ATLAS
HERRAMIENTA DE CARTOGRAFÍA WEB Y GEOCODIFICACIÓN PARA EL DESARROLLO DE SISTEMAS HÍBRIDOS EN ÁREAS URBANAS SOBRE J2EE Y POSTGRESQL

CARLOS ERNESTO ARTEAGA NOGUERA
MARIO RAÚL ERASO ORJUELA

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2008

ATLAS
HERRAMIENTA DE CARTOGRAFÍA WEB Y GEOCODIFICACIÓN PARA EL DESARROLLO DE SISTEMAS HÍBRIDOS EN ÁREAS URBANAS SOBRE J2EE Y POSTGRESQL

CARLOS ERNESTO ARTEAGA NOGUERA
MARIO RAÚL ERASO ORJUELA

Trabajo de grado presentado como requisito parcial para optar el título de ingeniero de sistemas

Dr. RICARDO TIMARÁN PEREIRA, Ph.D.

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2008

”Las ideas y las conclusiones aportadas en el presente trabajo son responsabilidad exclusiva de sus autores”

Artículo 1, acuerdo No. 324 de octubre 11 de 1966, emanado por el Honorable Consejo Directivo de la Universidad de Nariño.

Nota de aceptación:

Firma del presidente del jurado

Firma del jurado

Firma del jurado

San Juan de Pasto, 16 de abril del 2008.

CONTENIDO

pág.

1. INTRODUCCIÓN 27

1.1. TEMA 28

1.1.1. Título 28

1.1.2. Línea de investigación 28

1.1.3. Alcance y delimitación 28

1.2. PROBLEMA OBJETO DE ESTUDIO 28

1.2.1. Problema objeto de estudio 28

1.2.2. Formulación del problema 29

1.3. OBJETIVOS 29

1.3.1. Objetivo General 29

1.3.2. Objetivos específicos 29

1.4. JUSTIFICACIÓN 30

1.5. ORGANIZACIÓN DEL DOCUMENTO 30

2. CARTOGRAFÍA WEB 31

2.1. CARTOGRAFÍA 31

2.2. OGC 32

2.3. SISTEMAS DE COORDENADAS DE REFERENCIA 32

2.3.1. Sistemas de coordenadas geográficos 32

2.3.2. Sistemas de coordenadas proyectados 34

2.3.3. Proyecciones de mapa 34

2.3.4. Parámetros geodésicos EPSG 36

2.4. MAPAS DIGITALES 36

2.4.1. Formato raster 36

2.4.2. Formato vectorial 38

2.4.3. Renderización de mapas 40

2.5. CARTOGRAFÍA EN LA WEB 41

2.5.1. Tipos de mapas Web 41

2.5.2. Tecnologías de mapas Web del lado del servidor 42

2.5.3. Tecnologías de mapas Web del lado del cliente 43

2.6. WMS 44

2.6.1. Elementos básicos del servicio 45

2.6.2. Operaciones 46

2.7. APLICACIONES HÍBRIDAS 48

2.7.1. Estilos de mashup 48

3. GEOCODIFICACIÓN 50

3.1. EL PROCESO DE GEOCODIFICACIÓN 50

3.2. REFERENCIA LINEAL 51

3.3. NOMENCLATURA Y NUMERACIÓN URBANA 53

3.3.1. Numeración y nomenclatura urbana en San Juan de Pasto 53

3.4. ATLAS STREET CROSSING GEOCODER 55

3.4.1. Material de referencia. 55

3.4.2. Fases del proceso. 56

4. ANTECEDENTES 61

4.1. GEOSERVER 61

4.2. DEEGREE 63

4.3. MAPSERVER 68

4.4. ARCIMS 71

4.5. OPENLAYERS 74

4.6. GOOGLE MAPS 77

4.7. GOOGLE MOBILE 79

4.8. GOOGLE GEOCODER 82

5. SOFTWARE UTILIZADO EN LA CONSTRUCCIÓN DE ATLAS. 85

5.1. HERRAMIENTAS DE SOFTWARE 85

5.1.1. Netbeans IDE 85

5.1.2. Eclipse IDE 86

5.1.3. Quantum GIS 86

5.1.4. CAD2Shape 87

5.2. TECNOLOGÍAS JAVA 87

5.2.1. Principales características de Java 88

5.2.2. Java SE 90

5.2.3. Java ME 93

5.2.4. Java EE 95

5.2.5. JDBC 96

5.3. BIBLIOTECAS 97

5.3.1. Java topology suite 98

5.3.2. Geotools 98

5.3.3. C3PO 99

5.3.4. Hibernate 99

5.3.5. JDom 100

5.3.6. KXML 100

5.3.7. PNG encoder 100

5.4. GESTORES DE BASES DE DATOS 101

5.4.1. PostgreSQL 101

5.4.2. Postgis 102

6. ANÁLISIS, DISEÑO DE ATLAS 103

6.1. PROGRAMACIÓN EXTREMA 103

6.1.1. El objetivo de la programación extrema. 103

6.1.2. Valores de la programación extrema. 104

6.1.3. Principios de la programación extrema. 105

6.1.4. Actividades de la programación extrema. 106

6.1.5. Prácticas de la programación extrema. 108

6.2. MODELADO ÁGIL 110

6.2.1. Valores del modelado ágil. 111

6.2.2. Principios esenciales del modelado ágil. 111

6.2.3. Prácticas esenciales del modelado ágil. 113

6.3. ANÁLISIS UML. 115

6.3.1. Funciones. 115

6.3.2. Casos de uso del panel de control. 119

6.3.3. Casos de uso del Web Archive. 137

6.3.4. Casos de uso para los componentes de desarrollo. 139

6.3.5. Casos de uso el servidor Atlas 144

6.3.6. Diagramas de casos de uso del panel de control. 149

6.3.7. Diagramas de casos de uso del Web Archive. 152

6.3.8. Diagramas de casos de los componentes de desarrollo Atlas. 153

6.3.9. Diagramas de casos del servidor Atlas. 154

6.3.10. Diagramas de secuencia de sistema del panel de control. 155

6.3.11. Diagramas de secuencia de sistema del servidor Atlas. 158

6.3.12. Diagramas de secuencia de sistema de los componentes de desarrollo. 162

6.3.13. Diagramas de paquetes common. 163

6.3.14. Diagramas de paquetes del panel de control. 163

6.3.15. Diagramas de paquetes del Web Archive. 164

6.3.16. Diagramas de paquetes del servidor. 165

6.3.17. Diagramas de paquetes, biblioteca para desarrollo Java SE. 166

6.3.18. Diagramas de paquetes, biblioteca para desarrollo Java ME. 166

6.3.19. Diagramas de paquetes, biblioteca para desarrollo Web 2.0. 167

6.3.20. Diagramas de clases del dominio 169

6.3.21. Diagramas de clases para geocodificación 173

6.3.22. Diagramas de clases para la biblioteca de componentes de desarrollo 174

6.3.23. Diagramas de clases del lado del servidor 176

7. IMPLEMENTACIÓN DE LA HERRAMIENTA ATLAS 177

7.1. ARQUITECTURA DE ATLAS 177

7.2. MODULO DE UTILIDAD 179

7.2.1. Paquete clientdata 179

7.2.2. Paquete geocoding 185

7.2.3. Paquete hibernatedata 188

7.2.4. Paquete util 206

7.3. MODULO PANEL DE CONTROL 207

7.3.1. Paquete windows.configuration 207

7.3.2. Paquete windows.connections. 208

7.3.3. Paquete windows.controls. 208

7.3.4. Paquete windows.fonts. 211

7.3.5. Paquete windows.geocoding. 211

7.3.6. Paquete windows.icons. 212

7.3.7. Paquete windows.layers. 214

7.3.8. Paquete windows.projects. 214

7.3.9. Paquete windows.service. 215

7.3.10. Paquete windows.servicetest. 215

7.3.11. Paquete windows.sources. 216

7.3.12. Paquete windows.spatial_ref_sys. 218

7.3.13. Paquete windows.style. 219

7.3.14. Paquete windows.symbology. 220

7.4. MODULO DE SERVIDOR 227

7.4.1. Paquete exceptions. 227

7.4.2. Paquete geocoder. 228

7.4.3. Paquete wmsserver. 228

7.5. MODULO WEB ARCHIVE 229

7.5.1. Paquete tool 229

7.6. BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA STANDARD EDITION 231

7.6.1. Paquete gui. 231

7.6.2. Paquete events. 236

7.6.3. Paquete mashup. 237

7.7. BIBLIOTECA PARA DESARROLLO DE APLICACIONES WEB 2.0 CON JAVASCRIPT 238

7.7.1. Paquete Atlas. 238

7.8. BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA MOBILE EDITION 246

7.8.1. Paquete data. 246

7.8.2. Paquete events. 252

7.8.3. Paquete geocoding. 254

7.8.4. Paquete mobile. 254

8. PRUEBAS Y RESULTADOS 260

8.1. DATOS USADOS EN LAS PRUEBAS. 260

8.2. PRUEBAS SOBRE EL SERVIDOR DE CARTOGRAFÍA. 266

8.2.1. Pruebas de imágenes individuales. 266

8.2.2. Pruebas de simulación de clientes. 271

8.3. PRUEBAS SOBRE EL SERVIDOR DE GEOCODIFICACIÓN. 274

CONCLUSIONES 281

RECOMENDACIONES 283

ANEXOS

A1. GUÍA DE INSTALACIÓN DE PRERREQUISITOS. 284

A2. GUÍA DE INSTALACIÓN DE LAS HERRAMIENTAS ATLAS. 296

A3. GUÍA DE INSTALACIÓN DEL SERVIDOR ATLAS. 300

A4. GUÍA DE ADMINISTRACIÓN DEL SERVIDOR ATLAS. 307

A5. GUÍA DE DESARROLLO PLUGINS DE GEOCODIFICACIÓN. 331

A6. GUÍA DE DESARROLLO DE APLICACIONES JAVA SE. 334

A7. GUÍA DE DESARROLLO DE APLICACIONES JAVA ME. 355

A8. GUÍA DE DESARROLLO DE APLICACIONES WEB. 373

LISTA DE TABLAS

pág.

Tabla 1. Principales estándares del OGC. 32
Tabla 2. Clases básicas de simple feature Access. 40
Tabla 3. Parámetros de una URL de petición GetCapabilities. 46
Tabla 4. Parámetros principales de una URL de petición GetMap. 47
Tabla 5. Calculo de los valores M. 56
Tabla 6. Reemplazos en el proceso de análisis. 57
Tabla 7. Ejemplos direcciones analizadas. 59
Tabla 8. Clases generales de Java SE paquete Java.lang. 91
Tabla 9. Clases generales de Java SE paquete Java.io. 91
Tabla 10. Clases generales de Java SE paquete Java.net. 92
Tabla 11. Interfaces generales de Java SE paquete Java.util. 92
Tabla 12. Clases generales de Java SE paquete javax.swing. 93
Tabla 13. Resumen de la clase BBox. 180
Tabla 14. Resumen de la clase EpsgCRS. 181
Tabla 15. Resumen de la clase Layer. 182
Tabla 16. Resumen de la clase Project. 184
Tabla 17. Resumen de la clase Geocoder. 185
Tabla 18. Resumen de la clase ConfigurationDialog. 187
Tabla 19. Resumen de la clase Result. 187
Tabla 20. Resumen de la clase AtlasColor. 188
Tabla 21. Resumen de la clase AtlasStroke. 189
Tabla 22. Resumen de la clase GeocoderConfig. 191
Tabla 23. Resumen de la clase Icon. 192
Tabla 24. Resumen de la clase Layer. 193
Tabla 25. Resumen de la clase Project. 196
Tabla 26. Resumen de la clase Rule. 198
Tabla 27. Resumen de la clase Service. 199
Tabla 28. Resumen de la clase Source. 201
Tabla 29. Resumen de la clase Spatial_ref_sys. 204
Tabla 30. Resumen de la clase Style. 204
Tabla 31. Resumen de la clase Symbology. 205
Tabla 32. Resumen del paquete util. 206
Tabla 33. Resumen de la clase WMSExceptionUtil. 227
Tabla 34. Resumen de la clase geocoder.service. 228
Tabla 35. Resumen de la clase wmsserver.service. 229
Tabla 36. Resumen de la clase MapPanel. 231

Tabla 37. Resumen de la clase EventManager. 235
Tabla 38. Resumen de interfaces del paquete Event. 236
Tabla 39. Resumen de clases del paquete Event. 236
Tabla 40. Resumen de la clase Marker. 237
Tabla 41. Resumen de la clase Layer. 238
Tabla 42. Resumen de la clase CRS. 239
Tabla 43. Resumen de la clase Project. 239
Tabla 44. Resumen de la clase coordinateutils. 240
Tabla 45. Resumen de la clase BoundigBox. 240
Tabla 46. Resumen de la clase Point. 241
Tabla 47. Resumen de la clase Marker. 241
Tabla 48. Resumen de la clase Result. 242
Tabla 49. Resumen de la clase MapPanel. 243
Tabla 50. Resumen de la clase BBox. 247
Tabla 51. Resumen de la clase CRS. 248
Tabla 52. Resumen de la clase Layer. 249
Tabla 53. Resumen de la clase Project. 251
Tabla 54. Resumen de la clase FeatureInfoEvent. 252
Tabla 55. Resumen de la clase GeocodingEvent. 252
Tabla 56. Resumen de la clase MarkerTapEvent. 253
Tabla 57. Resumen de la clase PointerTapEvent. 253
Tabla 58. Interfaces del paquete events. 254
Tabla 59. Resumen de la clase Result. 254
Tabla 60. Resumen de la clase MapPanel. 254
Tabla 61. Resumen de la clase Marker. 259
Tabla 62. Características de los computadores usados en las pruebas. 260
Tabla 63. Tiempos en pruebas de imágenes individuales sin caché. 267
Tabla 64. Tiempos en pruebas de imágenes individuales con caché en formación. 268
Tabla 65. Tiempos en pruebas de imágenes individuales con caché formado. 270
Tabla 66. Tiempos en pruebas con clientes simulados A. 272
Tabla 67. Tiempos en pruebas con clientes simulados B. 273
Tabla 68. Tiempos de respuesta a peticiones simultáneas de geocodificación. 275
Tabla 69. Porcentajes de direcciones de la ciudad. 276
Tabla 70. Porcentajes de direcciones comuna centro. 277
Tabla 71. Direcciones geocodificadas en la ciudad. 278
Tabla 72. Direcciones geocodificadas en la comuna centro 279
Tabla 73. Bibliotecas en la construcción de geocoders. 331
Tabla 74. Modos de operación del MapPanel. 345
Tabla 75. Tipos de eventos generados por el MapPanel. 347
Tabla 76. Modos de operación del MapPanel Java ME. 364
Tabla 77. Modos de operación del MapPanel Web 2.0. 378

LISTA DE FIGURAS

pág.

Figura 1. Latitud phi (ϕ) y longitud lambda (λ). 33
Figura 2. Proyección de mapa. 35
Figura 3. Elipses de distorsión en proyección Mercator. 36
Figura 4. Codificación de una variable cuantitativa en formato raster. 37
Figura 5. Formato Vectorial. 38
Figura 6. Ejemplos de geometrías WKT. 40
Figura 7. Ejemplo de petición GetMap. 47
Figura 8. Mashup del lado del cliente. 48
Figura 9. Característica lineal con valores de medida. 52
Figura 10. Características lineales simples y complejas. 52
Figura 11. Calles y carreras de San Juan de Pasto. 53
Figura 12. Ubicación de una dirección en San Juan de Pasto. 54
Figura 13. Material de referencia del Atlas street crossing geocoder. 55
Figura 14. Máquina de estado para identificar el tipo de la dirección. 57
Figura 15. Máquina de estado para identificar el nombre de una vía. 58
Figura 16. Máquina de estado para identificar el número de la propiedad. 58
Figura 17. Formación de cuadras. 60
Figura 18. Logotipo de geoserver. 61
Figura 19. Bienvenida a la instalación de geoserver. 62
Figura 20. Progreso de la instalación de geoserver. 62
Figura 21. Metadatos WMS en geoserver. 63
Figura 22. Editor de almacén de datos de geoserver. 63
Figura 23. Logotipo de deegree. 63
Figura 24. XML descriptor de despliegue deegree. 65
Figura 25. XML de configuración servicio WMS deegree. 66
Figura 26. XML de administración de capas deegree. 67
Figura 27. Logotipo de MapServer. 68
Figura 28. Opciones de instalación de MapServer en Windows. 69
Figura 29. Progreso de la instalación de MapServer en Windows. 69
Figura 30. Código de ejemplo para capas WMS en MapServer. 70
Figura 31. Logotipo de ArcIMS. 71
Figura 32. Selección de componentes en la instalación de ArcIMS. 73
Figura 33. Opciones de post instalación de ArcIMS. 73
Figura 34. Selección de material en ArcIMS author. 73
Figura 35. Logotipo de OpenLayers. 74
Figura 36. Código de un visor simple de OpenLayers. 76

Figura 37. Apariencia de un visor simple de OpenLayers. 76
Figura 38. Logotipo de GoogleMaps. 77
Figura 39. Un ejemplo básico del manejo del API de GoogleMaps. 78
Figura 40. Mapa de calles de GoogleMaps. 78
Figura 41. Logotipo de GoogleMobile. 79
Figura 42. Teléfono celular con Google Mobile. 80
Figura 43. PDA con Google Mobile. 81
Figura 44. Logotipo de GoogleGeocoder. 82
Figura 45. Respuesta XML a una petición del geocoder del google. 83
Figura 46. Cliente GoogleMaps graficando una respuesta del geocoder del google. 83
Figura 47. Logotipo de Netbeans. 85
Figura 48. Logotipo de eclipse. 86
Figura 49. Logotipo de Quantum GIS. 87
Figura 50. Logotipo de Java. 88
Figura 51. Logotipo de vivid solutions. 98
Figura 52. Logotipo de geotools. 99
Figura 53. Logotipo de hibernate. 99
Figura 54. Logotipo de postgresql. 101
Figura 55. Logotipo de Postgis. 102
Figura 56. Diagrama A de casos de uso del panel de control. 150
Figura 57. Diagrama B de casos de uso del panel de control. 151
Figura 58. Diagrama general de casos de uso del Web archive. 152
Figura 59. Diagrama general de casos de uso de los componentes de desarrollo. 153
Figura 60. Diagrama general de casos de uso del servidor Atlas. 154
Figura 61. Diagrama de secuencia del sistema, conexión con un servidor. 155
Figura 62. Diagrama de secuencia del sistema, editar simbología por clases. 156
Figura 63. Diagrama de secuencia del sistema, editar simbología por rangos. 157
Figura 64. Diagrama de secuencia del sistema, petición GetCapabilities. 158
Figura 65. Diagrama de secuencia del sistema, petición GetMap. 159
Figura 66. Diagrama de secuencia del sistema, petición GetFeatureInfo. 160
Figura 67. Diagrama de secuencia del sistema, petición geocoder. 161
Figura 68. Diagrama de secuencia del sistema, visualizar mapa. 162
Figura 69. Paquetes del módulo common. 163
Figura 70. Paquetes del panel de control. 163
Figura 71. Paquete GUI en el panel de control. 164
Figura 72. Paquetes del Web Archive. 164
Figura 73. Paquetes del servidor. 165
Figura 74. Paquetes de la biblioteca para desarrollo Java SE. 166
Figura 75. Paquetes de la biblioteca para desarrollo Java ME. 167
Figura 76. Paquetes de la biblioteca para desarrollo de aplicaciones Web. 167
Figura 77. Diagrama de secuencia de interfaz del panel de control de ATLAS. 168
Figura 78. Diagrama A de clases del dominio. 169

Figura 79. Diagrama B de clases del dominio. 170
Figura 80. Diagrama C de clases del dominio. 171
Figura 81. Diagrama D de clases del dominio. 171
Figura 82. Diagrama de clases abstractas e interfaces de un geocoder. 173
Figura 83. Diagrama de clases del componente MapPanel. 174
Figura 84. Diagrama de clases de eventos. 174
Figura 85. Diagrama de clases de servlets. 176
Figura 86. Diagrama conceptual de arquitectura de la herramienta Atlas. 178
Figura 87. Apariencia de la clase FrmConfiguration. 207
Figura 88. Apariencia de la clase FrmManageGeocoderPlugins. 208
Figura 89. Apariencia de la clase FrmConnectionProps. 208
Figura 90. Apariencia de la clase PnlProjects. 209
Figura 91. Apariencia de la clase PnlLayers en pestaña de capas. 209
Figura 92. Apariencia de la clase PnlLayers en pestaña de leyendas. 210
Figura 93. Apariencia de la clase PnlViewer. 210
Figura 94. Apariencia de la clase AtlasColorChooser. 211
Figura 95. Apariencia de la clase FrmFonts. 211
Figura 96. Apariencia de la clase FrmAdminGeocoders. 212
Figura 97. Apariencia de la clase FrmAddIcons. 212
Figura 98. Apariencia de la clase FrmAdminIcons. 213
Figura 99. Apariencia de la clase IconsPanel. 213
Figura 100. Apariencia de la clase FrmLayerProps. 214
Figura 101. Apariencia de la clase FrmNewProyect. 214
Figura 102. Apariencia de la clase FrmServiceProps. 215
Figura 103. Apariencia de la clase FrmServerTest. 216
Figura 104. Apariencia de la clase FrmAddPostgis. 216
Figura 105. Apariencia de la clase FrmAddShapeFile. 217
Figura 106. Apariencia de la clase FrmLayerSrc. 217
Figura 107. Apariencia de la clase FrmSources. 218
Figura 108. Apariencia de la clase FrmSelectRefSys. 219
Figura 109. Apariencia de la clase PnlEditLineStyl. 219
Figura 110. Apariencia de la clase PnlEditPointStyle. 220
Figura 111. Apariencia de la clase PnlEditPolygonStyle. 220
Figura 112. Apariencia de la clase ClassRuleGenerator. 221
Figura 113. Apariencia de la clase RangeRuleGenerator. 222
Figura 114. Apariencia de la clase FixedRuleGenerator. 222
Figura 115. Apariencia de la clase FrmSymbLine con simbología fija. 223
Figura 116. Apariencia de la clase FrmSymbLine con simbología por rangos. 223
Figura 117. Apariencia de la clase FrmSymbLine con simbología por clases. 224
Figura 118. Apariencia de la clase FrmSymbPoint con simbología fija. 224
Figura 119. Apariencia de la clase FrmSymbPoint con simbología por rangos. 225
Figura 120. Apariencia de la clase FrmSymbPoint con simbología por clases. 225

Figura 121. Apariencia de la clase FrmSymbPoly con simbología fija. 226
Figura 122. Apariencia de la clase FrmSymbPoly con simbología por rangos. 226
Figura 123. Apariencia de la clase FrmSymbPoly con simbología por clases. 227
Figura 124. Apariencia de la clase DialogDB. 230
Figura 125. Apariencia de la clase DialogGeocoder. 230
Figura 126. Apariencia de la clase DialogWMS. 231
Figura 127. Apariencia de la capa Puntos.shp. 261
Figura 128. Apariencia de la capa Comunas.shp. 262
Figura 129. Apariencia de la capa Barrios.shp. 263
Figura 130. Apariencia de la capa Manzana.shp. 264
Figura 131. Apariencia de la capa Malla.shp. 265
Figura 132. Imagen compuesta e imagen individual. 266
Figura 133. Apariencia de la aplicación para pruebas individuales. 267
Figura 134. Tiempos en pruebas de imágenes individuales sin caché. 268
Figura 135. Tiempos en pruebas de imágenes individuales con caché en formación. 269
Figura 136. Tiempos en pruebas de imágenes individuales con caché formado. 270
Figura 137. Apariencia de la aplicación para pruebas de clientes. 271
Figura 138. Tiempos en pruebas con clientes simulados A. 272
Figura 139. Tiempos en pruebas con clientes simulados B. 274
Figura 140. Apariencia de la aplicación para pruebas de geocodificación. 275
Figura 141. Tiempos de respuesta a peticiones simultáneas de geocodificación. 276
Figura 142. Porcentajes de direcciones de la ciudad. 277
Figura 143. Porcentajes de direcciones comuna centro. 278
Figura 144. Direcciones geocodificadas en la ciudad. 279
Figura 145. Direcciones geocodificadas en la comuna centro 280
Figura 146. Home del sitio Web sobre Java en la página de Sun Microsystems. 284
Figura 147. Página Web de descargas de J2SDK. 284
Figura 148. Página Web de descarga del Apache Tomcat. 285
Figura 149. Contenido del archivo users.xml. 287
Figura 150. Ventana de propiedades del sistema en Windows. 287
Figura 151. Variables de entorno del sistema Windows. 288
Figura 152. Nueva variable del sistema Windows. 288
Figura 153. Éxito en la iniciación de Apache Tomcat. 289
Figura 154. Pantalla de bienvenida de Apache Tomcat. 289
Figura 155. Ventana database system ready. 293
Figura 156. Botón de conexión del PgAdmin III. 293
Figura 157. Ventana de conexión de PgAdmin III. 294
Figura 158. Creación de una base de datos con PgAdmin III. 294
Figura 159. Cuadro de diálogo, nueva base de datos. 295
Figura 160. Selección de idioma en el instalador de Atlas. 296
Figura 161. Pantalla de bienvenida en el instalador de Atlas. 296
Figura 162. Aceptación de licencia en el instalador de Atlas. 297

Figura 163. Selección de componentes en el instalador de Atlas. 297
Figura 164. Selección de ruta instalación en el instalador de Atlas. 298
Figura 165. Selección de grupo en el menú inicio en el instalador de Atlas. 298
Figura 166. Carpeta Atlas en el menú inicio de Windows. 299
Figura 167. Iniciar el panel de control Atlas. 300
Figura 168. Iniciar el WEB Archive. 300
Figura 169. Primer paso del asistente para configuración de archivos WAR. 301
Figura 170. Segundo paso del asistente para configuración de archivos WAR. 302
Figura 171. Tercer paso del asistente para configuración de archivos WAR. 303
Figura 172. Página principal de Tomcat. 304
Figura 173. Gestor de aplicaciones Web de Tomcat. 305
Figura 174. Sección, archivo WAR a desplegar. 305
Figura 175. Aplicación correctamente instalada en el servidor Tomcat. 306
Figura 176. Iniciar el panel de control Atlas. 307
Figura 177. Distribución de componentes del panel de control. 307
Figura 178. Barra de funciones del panel de control. 308
Figura 179. Barra de proyectos. 308
Figura 180. Herramientas de capas. 309
Figura 181. Sección de visualización. 309
Figura 182. Formulario, opciones de la herramienta. 310
Figura 183. Formulario, administración de plugins. 310
Figura 184. Botón connect en el panel de control. 311
Figura 185. Propiedades de la conexión en el panel de control. 311
Figura 186. Diagnósticos del servidor. 312
Figura 187. Información del servicio. 313
Figura 188. Botón WMS Service. 314
Figura 189. Formulario de nuevo proyecto. 315
Figura 190. Formulario selección de sistema de referencia espacial. 316
Figura 191. Botón administrar orígenes de datos. 317
Figura 192. Formulario, administrador de orígenes. 317
Figura 193. Agregar un origen de datos ShapeFile. 318
Figura 194. Formulario, importación de un ShapeFile. 318
Figura 195. Agregar un origen de datos Postgis. 319
Figura 196. Formulario, importación Postgis. 319
Figura 197. Administrador de geocoders. 320
Figura 198. Botones en la administración de geocoders. 320
Figura 199. Cuadro de selección de proyectos. 321
Figura 200. Formulario, nueva capa. 321
Figura 201. Lista de capas. 322
Figura 202. Botón, configurar orígenes de datos. 323
Figura 203. Formulario, asignación de orígenes. 323
Figura 204. Botón, administrar etiquetado. 324

Figura 205. Formulario, administrar etiquetas. 324
Figura 206. Botón editar simbología. 325
Figura 207. Alternativas de simbología. 325
Figura 208. Simbología por rangos. 326
Figura 209. Simbología por clases. 327
Figura 210. Estilos para geometrías tipo punto. 328
Figura 211. Estilos para geometrías tipo línea. 328
Figura 212. Estilos para geometrías tipo polígono. 329
Figura 213. Botones denominadores de escala. 329
Figura 214. Remover denominadores de escala. 330
Figura 215. Botones de subir y bajar capa. 330
Figura 216. Botón nuevo proyecto en NetBeans. 334
Figura 217. Nueva aplicación java en NetBeans. 334
Figura 218. Opciones para una nueva aplicación java en NetBeans. 335
Figura 219. Árbol de proyecto en NetBeans. 335
Figura 220. Menu agregar archivo jar en NetBeans. 336
Figura 221. Libraries en un proyecto NetBeans. 336
Figura 222. Creación de un formulario en NetBenans. 337
Figura 223. Nombre y ubicación de un proyecto NetBeans. 337
Figura 224. Árbol de proyecto en NetBeans con una clase nueva. 338
Figura 225. Vista de diseño en NetBeans. 338
Figura 226. Paleta de componentes Swing en java. 339
Figura 227. Arbol de componentes del formulario principal. 340
Figura 228. Apariencia de la clase Main. 340
Figura 229. Grupos de botones. 341
Figura 230. Tabla de propiedades del jPanel. 341
Figura 231. Código de inicialización para jPanel. 342
Figura 232. Import de la clase MapPanel. 342
Figura 233. Código de acceso a una instancia de MapPanel. 342
Figura 234. Constructor de la clase Main. 343
Figura 235. Imports en la clase Main para establecer conexión con un servidor. 343
Figura 236. Constructor de la clase Main, preparado para conectarse con un servidor. 344
Figura 237. Apariencia de la clase Main. 344
Figura 238. Eventos de un botón. 345
Figura 239. Métodos para interactuar con el Mapa. 346
Figura 240. Llamado a registrarEscuchadores en el constructor la clase Main. 347
Figura 241. Contenido el metodo registrar escuchadores. 348
Figura 242. Indicadores de posición y escala en Main. 348
Figura 243. Modificaciones al método regristrarEscuchadores para GetFeatureInfo. 349
Figura 244. Modificaciones al constructor para recibir respuestas GetFeatureInfo. 349
Figura 245. Escuchador del botón tglInfo. 350
Figura 246. Área de interés de una consulta GetFeatureInfo. 350

Figura 247. Resultado de una consulta GetFeatureInfo. 350
Figura 248. Modos para la clase Main. 351
Figura 249. Botones para modos adicionales en la clase Main. 351
Figura 250. Código en los botones para modos adicionales en la clase Main. 352
Figura 251. Modificaciones al método regristrarEscuchadores para marcadores. 352
Figura 252. Modificaciones al método regristrarEscuchadores para geocoding. 353
Figura 253. Componentes de Main, para peticiones al geocoder. 354
Figura 254. Geocodificación de una dirección. 354
Figura 255. Botón nuevo proyecto en NetBeans. 355
Figura 256. Nuevo proyecto java ME en NetBeans. 355
Figura 257. Opciones para una nueva aplicación java en NetBeans. 356
Figura 258. Selección de plataforma por defecto. 356
Figura 259. Configuraciones adicionales. 357
Figura 260. Árbol de proyecto en NetBeans. 357
Figura 261. Menu agregar recurso en NetBeans. 357
Figura 262. Resource en un proyecto NetBeans. 358
Figura 263. Creación de un Visual MIDlet en NetBeans. 358
Figura 264. Propiedades del visual MIDlet. 359
Figura 265. Árbol de proyecto en NetBeans con un MIDlet nuevo. 359
Figura 266. Vista Source en NetBeans. 360
Figura 267. Imports biblioteca Atlas para apliciones Java ME. 360
Figura 268. Implementación de CommandListener. 361
Figura 269. Atributos de la clase MIDlet. 361
Figura 270. Constructor de la clase Atlas. 361
Figura 271. Código de acceso a una instancia de MapPanel. 362
Figura 272. Código de acceso a una instancia de Formulario. 362
Figura 273. Código de lanzamiento del MIDlet. 362
Figura 274. Submenu para ejecutar la aplicación. 363
Figura 275. Apariencia de la aplicación MobileAtlas. 363
Figura 276. Código para administrar el manejo de modos. 364
Figura 277. Modificación al método startMIDlet para GetFeatureInfo. 365
Figura 278. Modificación código administrador de modos. 366
Figura 279. Modificación al metodo getMapPanel para FeatureInfo. 367
Figura 280. Área de interés de una consulta GetFeatureInfo. 367
Figura 281. Respuesta getFeatureInfo en formato texto plano. 368
Figura 282. Modificación al método startMIDlet para detectar toques en la pantalla. 369
Figura 283. Código para agremas el modo MODEGETCOORD. 369
Figura 284. Modificaciones al método startMIDlet para manejo de geocoding. 370
Figura 285. Código para agregar botón del geocoder. 370
Figura 286. Código para enviar la petición al servidor de geocodificación. 371
Figura 287. Apariencia de la aplicación respuesta geocoder. 372
Figura 288. Estructura de directorios aplicación web 373

Figura 289. Descomprimir el archivo atlas_web.zip 373
Figura 290. Crear archivo index.html. 374
Figura 291. Código básico de una pagina html. 374
Figura 292. Código para incluir la biblioteca atlas.js 374
Figura 293. Código de inicio para la carga de la página 375
Figura 294. Código del div del mapa. 375
Figura 295. Código conectarse al servidor 376
Figura 296. Apariencia de la aplicación Atlas Web. 377
Figura 297. Código botones de modo MapPanel. 378
Figura 298. Apariencia de la aplicación Atlas Web interactiva. 379
Figura 299. Código Escuchador mouse move. 379
Figura 300. Div de visualización de coordenadas. 380
Figura 301. Indicadores de posición del mapa. 380
Figura 302. Código configuración FeatureInfo. 381
Figura 303. Código para seleccionar el modo FeatureInfo 381
Figura 304. Área de interés de una consulta GetFeatureInfo. 382
Figura 305. Resultado HTML de la petición GetFeatureInfo. 382
Figura 306. Escuchador del evento clic del MapPanel 383
Figura 307. Código para seleccionar el modo MODEGETCOOR 383
Figura 308. Escuchador para la petición de geocoding. 384
Figura 309. Código para agregar botón del geocoder. 384
Figura 310. Dirección geocodifica. 385

Resumen

En este proyecto se muestra las etapas de análisis, diseño e implementación de Atlas, una herramienta de cartografía Web y geocodificación para el desarrollo de sistemas híbridos en áreas urbanas.

Esta herramienta está compuesta por un servidor de cartografía Web, compatible con el estándar internacional para publicación de mapas en Internet WMS. Además, cuenta con un servidor de geocodificación extensible mediante plugins, una herramienta gráfica para administrar el contenido de estos servidores y bibliotecas para el desarrollo de software compatible con los servidores para ambientes de escritorio, Web 2.0 y dispositivos móviles.

Se analiza y evalúa el desempeño de los servidores de cartografía y geocodificación, simulando diversas condiciones de concurrencia y configuración, además, se prueba la efectividad de un nuevo algoritmo, Atlas street crossing geocoder, propuesto para demostrar las capacidades del servidor de geocodificación.

Abstract

This project shows the stages of analysis, design and implementation of Atlas, a tool for Web mapping and geocoding, for the development of hybrid systems in urban areas.

This tool consists of a Web mapping server, compatible with the international standard for the distribution of maps on the Internet WMS. It has a geocoding server, extensible via plugins, a graphical tool for managing the content of these servers and libraries for the development of software compatible with the servers, for desktop, Web 2.0 and mobile devices.

It analyzes and evaluates the performance of Web mapping and geocoding servers under various simulated conditions of concurrency and configuration; also, it evaluates the effectiveness of the new algorithm Atlas street crossing geocoder, proposed to demonstrate the capabilities of the geocoding server.

1. INTRODUCCIÓN

Los mapas constituyen una útil representación de la realidad, de ahí que el hombre los ha empleado en la resolución de variados problemas. La cartografía impresa dominó el panorama durante largos años, a pesar de su utilidad, su alto costo permitía que solo unos pocos se beneficiaran de ella, con el paso del tiempo la aparición de las ciencias de la computación y su capacidad para mejorar distintos procesos tocó también a esta disciplina, formando la cartografía digital.

Con la aparición de las redes de computadores, la cartografía digital encontró el modo de llegar a más y más personas, ampliando no solo su cobertura, sino también enriqueciendo su contenido al recopilar información de distintas fuentes y agregando capacidades de interacción antes insospechadas.

Actualmente las organizaciones poseen información alfanumérica sobre diversos sucesos, dicha información requiere ser transformada antes de ser útil a un sistema de cartografía Web, proceso que puede demandar un gran esfuerzo. Es ahí donde cobra gran importancia el servicio de geocodificación, que permite automatizar dicha transformación para el caso de descripciones de ubicaciones como direcciones urbanas.

En este documento se presenta el trabajo de grado para optar por el título en ingeniería de sistemas, “ATLAS: HERRAMIENTA DE CARTOGRAFÍA WEB Y GEOCODIFICACIÓN PARA EL DESARROLLO DE SISTEMAS HÍBRIDOS EN ÁREAS URBANAS SOBRE J2EE Y POSTGRESQL”, desarrollado bajo software libre. Esta herramienta está compuesta por un servidor de cartografía Web, compatible con el estándar internacional para publicación de mapas en Internet WMS. Además, cuenta con un servidor de geocodificación extensible mediante plugins, una herramienta gráfica para administrar el contenido de estos servidores y bibliotecas para el desarrollo de software compatible con los servidores para ambientes de escritorio, Web 2.0 y dispositivos móviles. Se plantea e implementa un algoritmo de geocodificación titulado Atlas street crossing, creado para demostrar las capacidades del servidor de geocodificación y resolver direcciones en la nomenclatura de la zona centro de la ciudad de San Juan de Pasto.

1.1. TEMA

1.1.1. Título. ATLAS: Herramienta de cartografía Web y geocodificación para el desarrollo de sistemas híbridos en áreas urbanas sobre J2EE y PostgreSQL.

1.1.2. Línea de investigación. La propuesta de trabajo de grado, se encuentra inscrita bajo la línea de software y manejo de información.

1.1.3. Alcance y delimitación. ATLAS será una herramienta de cartografía en la Web (Web mapping) y geocodificación (geocoding). El propósito de ATLAS es proveer información cartográfica de áreas urbanas, que terceros combinarán con sus datos regulares y geográficos para ofrecer un servicio al público. Dicho proceso de combinación se denomina aplicaciones híbridas (mashup).

ATLAS empleará servicios Web para cartografía y geocodificación, además clientes que se conectarán a los servicios para formar aplicaciones híbridas. Dichos clientes funcionarán en dispositivos móviles, navegadores Web y aplicaciones de escritorio.

1.2. PROBLEMA OBJETO DE ESTUDIO

1.2.1. Problema objeto de estudio. Los sistemas de información que emplean datos geográficos, cobran creciente importancia, dado que tienen varias características favorables como:

- Permiten analizar y revelar información difícil de percibir con otros medios.
- Se estima que el 80% de los datos corporativos existentes en todo el mundo poseen este componente geográfico.
- Dado que proveen un marco común, que es la relación de los datos con el globo terrestre, permiten la integración de diversas fuentes de información.
- Pueden aprovechar las crecientes mejoras en las redes de información.
- Resultan muy útiles a la hora de administrar y gestionar recursos.

En el mercado existen varias herramientas para este propósito, como ArcIMS, Autodesk Map sin embargo, no son libres, sus licencias resultan demasiado costosas y están sujetas a ejecutarse en sistemas operativos no libres lo que incrementa aun más su costo dejándolas fuera de alcance para pequeñas y medianas organizaciones.

GoogleMap, GoogleEarth y YahooMap, permiten la integración de sus datos con otras fuentes, operan en Internet, pero carecen de información sobre Nariño y su carácter gratuito, pero no libre impide la adición de dicha información.

En el software libre existen herramientas, como gvSig, JUMP y Kosmo, que a pesar de ser independientes de plataforma y ser libres carecen de soporte WEB.

Lo anterior muestra la inexistencia de una herramienta que permita a las pequeñas y medianas organizaciones de la región administrar su información de forma georeferencial en Internet a un precio asequible.

Por esta razón, se plantea el desarrollo de una herramienta flexible, capaz de ejecutarse sobre sistemas operativos libres y privativos, escalable, de código abierto, que pueda trabajar en Internet y en aplicaciones de escritorio, que tenga información suficiente para ser útil en la región, que permita el ingreso de información de otros lugares y permita a terceros integrar sus datos en ella.

1.2.2. Formulación del problema. ¿Cómo facilitar a las pequeñas y medianas organizaciones de la región y el país administrar su información de forma georeferencial de manera económica?

1.3. OBJETIVOS

1.3.1. Objetivo General. Facilitar a las pequeñas y medianas organizaciones de la región y el país administrar su información de forma georeferencial a través del desarrollo de una herramienta de cartografía en la Web y geocodificación capaz de proveer información cartográfica de áreas urbanas.

1.3.2. Objetivos específicos

- Analizar las herramientas similares existentes que trabajen cartografía en la Web y geocodificación.
- Diseñar e implementar el módulo de cartografía en la Web y geocodificación.
- Diseñar e implementar los clientes para aplicaciones de escritorio, entornos Web y dispositivos móviles compatibles con CLDC.
- Probar los componentes de cartografía Web y geocodificación con datos reales y analizar los resultados.
- Construir demostraciones y documentación sobre el uso de los clientes.

1.4. JUSTIFICACIÓN

Los mapas constituyen una útil representación de la realidad de ahí que el hombre los ha empleado en la resolución de variados problemas. La cartografía impresa dominó el panorama durante largos años, a pesar de su utilidad su alto costo permitía que solo unos pocos se beneficiaran de ella, con el paso del tiempo la aparición de las ciencias de la computación y su capacidad para mejorar distintos procesos tocó también a esta disciplina formando la cartografía digital, que permitió la reducción de costos y abrió las puertas a que más personas accedan a los mapas.

No siempre resulta fácil extraer datos de una fuente única coherente y de calidad, con la aparición de las redes de computadores, la cartografía digital encontró el modo de llegar a más y más personas, ampliando no solo su cobertura, sino también enriqueciendo su contenido al recopilar información de distintas fuentes y agregando capacidades de interacción antes insospechadas.

La cartografía Web posibilita la difusión de la cartografía temática ofreciendo contenidos prácticamente a la medida de quien los consulta. La región requiere herramientas que le permitan la apropiación de estás tecnologías a un costo razonable, de ahí, la importancia de Atlas como un marco común, que permitirá a pequeñas y medianas organizaciones tratar su información de forma georeferencial.

Actualmente las organizaciones poseen información alfanumérica sobre diversos sucesos, dicha información requiere ser transformada antes de ser útil a un sistema de cartografía Web, proceso que puede demandar un gran esfuerzo. Es ahí donde cobra gran importancia el servicio de geocodificación, que permite automatizar dicha transformación para el caso de descripciones de ubicaciones como direcciones urbanas.

Atlas puede ser alimentado con información de otros lugares, hecho que incrementa su flexibilidad y potencia, constituyéndolo en un aporte valioso para la academia, el país y la humanidad.

1.5. ORGANIZACIÓN DEL DOCUMENTO

Este documento se organiza en capítulos, en el capítulo 2 explica los conceptos de cartografía Web, el capitulo 3 describe el proceso de geocodificación y el algoritmo de Atlas street crossing geocoder, el capitulo 4 analiza las herramientas afines al objetivo de Atlas, el capitulo 5 detalla el software utilizado en la construcción de Atlas, el capitulo 6 contiene el análisis y diseño de la herramienta, el capitulo 7 explica la implementación de la herramienta y sus componentes, finalmente el capitulo 8 describe las pruebas de rendimiento realizadas sobre la aplicación.

2. CARTOGRAFÍA WEB

En cartografía la tecnología ha cambiado continuamente a fin de satisfacer las demandas de las nuevas generaciones de creadores y usuarios de mapas, dando origen a lo que hoy se denomina cartografía Web.

2.1. CARTOGRAFÍA

La cartografía es una disciplina que integra ciencia, técnica y arte, que trata de la representación de la tierra sobre un mapa.

Al ser la tierra esférica ha de valerse de un sistema de proyecciones para pasar de la esfera al plano. Pero, además de representar los contornos de las cosas, las superficies y los ángulos, se ocupa también de representar la información que aparece sobre el mapa, según se considere que es relevante y que no. Esto, normalmente depende de lo que se quiera representar en el mapa y de la escala.

Los primeros mapas fueron construidos manualmente con pinceles y pergamino y en consecuencia, con variada calidad y limitada distribución. La aparición de dispositivos magnéticos, como la brújula y mucho más tarde de los dispositivos de almacenamiento, permitió la creación de mapas mucho más precisos y la capacidad de almacenarlos y manipularlos digitalmente.

Los avances en la tecnología como la litografía, han permitido la creación de mapas con pequeños detalles, sin distorsión en forma y resistencia a la humedad y el desgaste.

A fines del siglo 20 y principios del siglo 21 los avances en la tecnología electrónica han dado lugar a una nueva revolución en la cartografía. En concreto, los dispositivos de hardware como las pantallas, plotters, impresoras y escáneres (a distancia y en el documento) junto con la visualización, procesamiento de imágenes, análisis espacial y el software de bases de datos, han democratizado y ampliado en gran medida la elaboración de los mapas. La posibilidad de superponer variables situadas en el espacio en los mapas existentes, ha creado nuevos usos de los mapas y nuevas industrias para explorar y explotar este potencial.

2.2. OGC.

El Open Geospatial Consortium (OGC) fue creado en 1994 y agrupa a más de 250 organizaciones públicas y privadas. Su fin es la definición de estándares abiertos e interoperables dentro de los sistemas de información geográfica, los principales se muestran en la tabla 1. Persigue acuerdos entre las diferentes empresas del sector que posibiliten la interoperación de sus sistemas de geoprocesamiento y facilitan el intercambio de la información geográfica en beneficio de los usuarios.

Tabla 1. Principales estándares del OGC.

GML	Lenguaje de Marcado Geográfico
WFS	Web Feature Service o Servicio de entidades vectoriales que proporciona la información relativa a la entidad almacenada en una capa vectorial que reúnen las características formuladas en la consulta.
WMS	Web Map Service o Servicio de mapas en la Web que produce mapas en formato imagen a la demanda para ser visualizados por un navegador Web o en un cliente simple.
WCS	Coverage se refiere a datos espaciales continuos, como fotografía aérea, uso del suelo y datos de elevación.

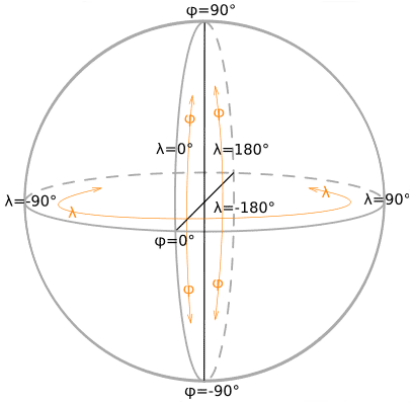
2.3. SISTEMAS DE COORDENADAS DE REFERENCIA

Para su correcto estudio e interpretación los datos consignados en los mapas están expresados respecto a un sistema de coordenadas de referencia, dichos sistemas pueden ser geográficos ó proyectados.

2.3.1. Sistemas de coordenadas geográficos. Un sistema de coordenadas geográfico usa una superficie esférica tridimensional para definir ubicaciones en la tierra. Un sistema de coordenadas geográfico incluye una unidad de medida angular, un primer meridiano y un datum.

Un punto es referenciado por sus valores de longitud y latitud. Longitud y latitud son ángulos medidos desde el centro de la tierra a un punto en la superficie. La relación entre los valores de latitud y longitud y el globo, se muestra en la figura 1.

Figura 1. Latitud phi (ϕ) y longitud lambda (λ).



Wikimedia Commons.

En el sistema esférico las líneas horizontales o líneas este a oeste, son líneas de igual latitud, o paralelos. Las líneas verticales, o líneas norte a sur, son líneas de igual longitud, o meridianos. Estas líneas envuelven el globo y forman una malla llamada retícula.

La línea de latitud en la mitad entre los polos es llamada ecuador. Esta define la línea de latitud cero. La línea de longitud cero es llamada el primer meridiano. Para la mayoría de sistemas de coordenadas geográficos, el primer meridiano es la longitud que pasa por Greenwich Inglaterra. Otros países usan líneas de longitud que pasan por Bern, Bogotá y Paris como primer meridiano. El origen de la retícula (0,0) está definido por la intersección entre el ecuador y el primer meridiano.

Tradicionalmente, los valores de latitud y longitud se miden en grados decimales o en grados, minutos y segundos. Los valores de latitud son medidos en relación al ecuador y van desde -90° en el polo sur, a +90° en el polo norte. Los valores de longitud están medidos en la relación al primer meridiano. Ellos van desde -180° al oeste a +180° al este.

Puede ser útil igualar los valores de longitud con X y los valores de latitud con Y. Los datos definidos en un sistema de coordenadas geográficas se muestran como si un grado fuese una unidad lineal de medida. Este método es básicamente el mismo que en la proyección Plate Carrée.

A pesar de que longitud y latitud pueden localizar posiciones exactas en la superficie del globo, no son unidades uniformes de medida. Solo a lo largo del ecuador la distancia representada por un grado de longitud se aproxima a la distancia representada por un grado de latitud. Esto es por el hecho de que el ecuador es el único paralelo tan largo como un meridiano. Bajo y sobre el ecuador, los círculos que definen los paralelos se hacen más pequeños hasta que forman un solo punto en los polos.

La forma y tamaño de la superficie de un sistema de coordenadas geográfico está definida por una esfera o esferoide. Una esfera está basada en un círculo, mientras que un esferoide está basado una forma elíptica. Aunque la tierra se representa mejor por un esferoide a veces se trata como una esfera para simplificar cálculos matemáticos. La selección entre una esfera ó un esferoide dependerá del tamaño del mapa y la precisión deseada.

Mientras que un esferoide aproxima la forma de la tierra, un datum define la posición del esferoide relativa al centro de la tierra. Un datum provee un marco de referencia para medir ubicaciones en la superficie de la tierra. Este define el origen y orientación de las líneas de latitud y longitud.

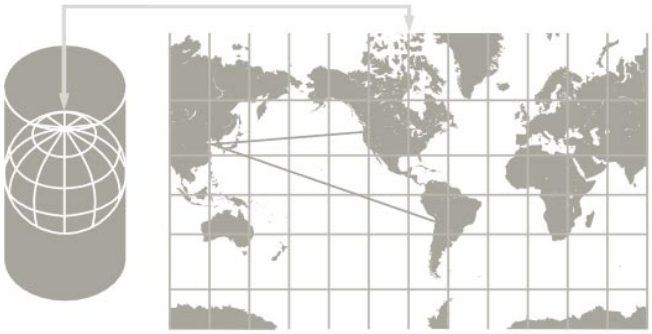
2.3.2. Sistemas de coordenadas proyectados. Un sistema de coordenadas proyectado está definido en una superficie plana bidimensional. A diferencia de los sistemas geográficos, un sistema proyectado tiene longitudes constantes. Un sistema proyectado siempre está basado en un sistema geográfico y una proyección.

En un sistema de coordenadas proyectado, las ubicaciones están identificadas por coordenadas x, y en una malla, con el origen en el centro de esta. Cada posición tiene dos valores que la referencian respecto a la localización central. Uno especifica la posición horizontal y otro la vertical.

En esta malla de líneas verticales y horizontales de igual separación, la línea horizontal se denomina eje x y la vertical eje y. Las líneas horizontales sobre el origen tienen valores positivos así como las líneas verticales a la derecha del origen, es decir, están orientados hacia el norte y el este. En ocasiones las proyecciones pueden indicar un orden y orientación diferentes. Por ejemplo, el sistema Hartebeesthoek94/Lo25 usado en el Sudáfrica tiene ejes que incrementan al oeste y al sur.

2.3.3. Proyecciones de mapa. Al tratar a la tierra como una esfera o como un esferoide se debe trasformar su superficie tridimensional para crear una hoja plana de mapa. Esta transformación matemática se conoce comúnmente como proyección de mapa. La figura 2, muestra un ejemplo conceptual sobre una proyección de mapa.

Figura 2. Proyección de mapa.



Wikimedia Commons.

Cuando se intenta ajustar un esferoide en una superficie plana, la forma del mapa es una distorsión de la verdadera configuración de la superficie terrestre. Un esferoide no se puede llevar a un plano por métodos triviales. Representar la superficie de la tierra en dos dimensiones causa distorsión en la forma, área, distancia o dirección de los datos.

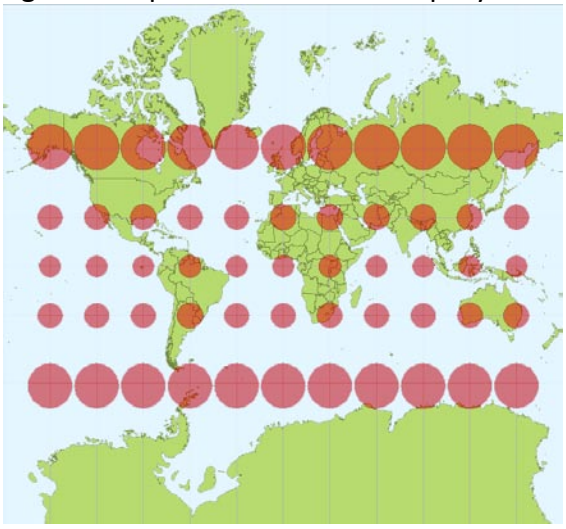
Una proyección de mapa usa fórmulas matemáticas para relacionar coordenadas esféricas en el globo a coordenadas en el plano.

Distintas proyecciones causan diferentes tipos de distorsión. Una proyección puede mantener el área, pero alterar la forma. De modo tal que la proyección a utilizar dependerá del propósito del mapa.

Por ejemplo, la proyección de Mercator se basa en el modelo ideal que trata a la tierra como un globo hinchable que se introduce en un cilindro y que empieza a inflarse ocupando el volumen del cilindro e imprimiendo el mapa en su interior, este cilindro cortado longitudinalmente y ya desplegado es el mapa con proyección de Mercator.

Esta proyección presenta una buena exactitud en su zona central, pero las zonas superior e inferior correspondientes a norte y sur presentan grandes deformaciones como puede apreciarse en la figura 3.

Figura 3. Elipses de distorsión en proyección Mercator.



Wikimedia Commons

Groenlandia aparece aproximadamente del tamaño de África, cuando en realidad el área de África es aproximadamente 14 veces el de Groenlandia.

2.3.4. Parámetros geodésicos EPSG. [1] El OGP surveying and positioning committee, mediante el sub comité de geodesia, mantiene y publica un conjunto de datos de los parámetros sobre sistemas de coordenadas de referencia y descripción de trasformación de coordenadas.

2.4. MAPAS DIGITALES

Los mapas digitales pueden ser almacenados en formatos raster o vectoriales, los formatos vectoriales son más populares en el mercado. No obstante, los formatos raster son muy utilizados en estudios que requieran la generación de capas continuas, necesarias en fenómenos no discretos y fenómenos medioambientales.

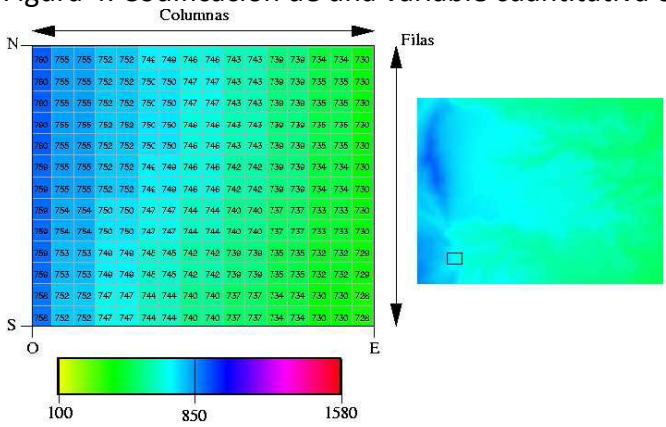
Por otra parte, es posible generar imágenes raster con base en datos vectoriales, transformación muy útil en campos como el Web mapping.

2.4.1. Formato raster. Consiste en filas y columnas de celdas donde en cada celda se almacena un valor. Los datos raster pueden ser imágenes donde cada píxel (o celda) contiene un valor de color. Las celdas pueden también contener valores discretos como uso del suelo, o continuos como temperatura.

Mientras una celda raster almacena un solo valor, este puede ser extendido al usar bandas raster para representar colores RGB, o una tabla de atributos con una fila por cada celda. La resolución de los datos raster es su ancho de celda en unidades de terreno.

Los datos raster son almacenados en varios formatos; desde archivos basados en estándares como JPEG, GIF, PNG, etc. a archivos binarios almacenados directamente en sistemas gestores de bases de datos. El almacenamiento en bases de datos correctamente indexadas, permite una rápida recuperación de los datos, pero puede requerir el almacenamiento de varios registros de gran tamaño. La figura 4, presenta en ejemplo de codificación de una variable cuantitativa en formato raster, donde el color de cada celda está asociado a un valor en una escala.

Figura 4. Codificación de una variable cuantitativa en formato raster.



Wikimedia Commons

- JPEG (joint photographic experts group). Es un algoritmo diseñado para comprimir imágenes con 24 bits de profundidad o en escala de grises. JPEG es también el formato de fichero que utiliza este algoritmo para comprimir imágenes. El formato de archivos JPEG se abrevia frecuentemente jpg debido a que algunos sistemas operativos sólo aceptan tres letras de extensión.

  JPEG es un algoritmo de compresión con pérdida. Esto significa que al descomprimir la imagen no se obtiene exactamente la misma imagen que se tenía antes de la compresión.

  Una de las características que hacen muy flexible al JPEG es el ajuste del grado de compresión. Una compresión muy alta pierde una cantidad significativa de calidad, pero obtiene archivos de pequeño tamaño. Una tasa de compresión baja obtiene una

calidad muy parecida a la original, y un archivo mayor. Este formato no admite transparencias de ninguna clase.

- GIF (graphic interchange format). Se limita a una paleta de 8 bits, o 256 colores. Esto hace que el formato sea conveniente para el almacenamiento de gráficos con un número relativamente reducido de colores simples, tales como diagramas, figuras, símbolos, etc. Además soporta animación, transparencia binaria y utiliza una compresión sin pérdida que es más eficaz cuando grandes zonas tienen un único color.

- PNG (Portable Network Graphics), fue desarrollado libre y de código abierto, para solventar varias de las limitaciones del GIF. Admite 16 millones de colores, sobresale cuando la imagen tiene grandes zonas de color uniforme, soporta transparencia alfa. Es adecuado para la distribución final de las imágenes fotográficas debido al menor tamaño de sus archivos. Algunos de los navegadores todavía no son totalmente compatibles con el formato.

2.4.2. Formato vectorial. Se centra en la precisión de localización de los elementos sobre el espacio y donde los fenómenos a representar son discretos, es decir, de límites definidos. Para modelar digitalmente las entidades del mundo real se utilizan tres objetos espaciales que pueden observarse en la figura 5. El punto se utiliza para características geográficas que puedan ser expresadas por una sola referencia, la línea utilizada para características que puedan ser modeladas a través de funciones lineales y el polígono para características que cubren una zona determinada de la superficie terrestre. Cada una de estas geometrías están vinculadas a una fila en una base de datos que describe sus atributos alfanuméricos.

Una ventaja de los formatos vectoriales es que pueden ser escalados tanto como se requiera sin pérdida de calidad de imagen.

Figura 5. Formato Vectorial.

Entre los formatos más difundidos se encuentran SHP, GML, WKT y WKB.

- SHP ESRI Shapefile. [2] Es un formato propietario y abierto de datos espaciales desarrollado por la compañía ESRI, quien desarrolla y comercializa software para sistemas de información geográfica como ArcGIS y ArcIMS. Originalmente se creó para la utilización con su producto ArcView GIS, pero actualmente se ha convertido en formato estándar de facto por la importancia que los productos ESRI tienen en el mercado.

  Un Shapefile es un formato vectorial de almacenamiento digital donde se guarda la localización de los elementos geográficos y los atributos asociados a ellos. El formato carece de capacidad para almacenar información topológica. Se compone de al menos tres archivos que tienen las siguientes extensiones:

  Shp. Es el archivo que almacena las entidades geométricas de los objetos.
  Shx. Es el archivo que almacena el índice de las entidades geométricas.
  Dbf. Es el dBASE, o base de datos, es el archivo que almacena la información de los atributos de los objetos.

- GML (Geography Markup Language). [3] Es la gramática XML definida por el Open Geospatial Consortium (OGC) para expresar características geográficas. Sirve como lenguaje de modelado para sistemas geográficos, así como un formato abierto de intercambio para transacciones geográficas en Internet. Los perfiles GML son restricciones lógicas pensadas para simplificar la adopción del estándar, puede decirse que definen sub conjuntos.

  Perfil de rasgos simples. Soporta una amplia gama de objetos vectoriales contenidos en un modelo de geometría de objetos lineales de 0, 1 y 2 dimensiones, todas basadas en interpolación lineal. Cuenta con un modelo de características simplificado que solo puede tener un nivel de profundidad, donde los tipos no geométricos deben ser tipos simples XML.

  Esquemas de aplicación. Para exponer los datos de una aplicación geográfica usando GML una comunidad u organización crea un esquema XML específico del dominio de la aplicación de interés. Este esquema describe los tipos de objetos en que está interesada la comunidad. Por ejemplo, una aplicación para turismo puede definir tipos de objetos incluyendo monumentos, lugares de interés y museos. Estos tipos de objeto a su vez referencian a los tipos primitivos definidos en el estándar GML.

- Simple Features Access. [4] Es una especificación del OGC también conocida como ISO 19125, describe una arquitectura común para geometría vectorial simple usando un modelo de objetos, los más básicos se presentan en la siguiente tabla.

Tabla 2. Clases básicas de simple feature Access.

Point	Representa una ubicación en el espacio de coordenadas x, y, también puede tener valores z y m.
MultiPoint	Es una colección geométrica restringida a puntos.
LineString	Curva con interpolación lineal entre puntos.
Line	LineString con exactamente dos puntos.
LinearRing	LineString cerrado y simple.
MultiLineString	Colección geométrica restringida a LineString.
Polygon	Superficie plana definida por un límite exterior y cero o más límites interiores. Cada límite interior define un agujero en el polígono.

La especificación define además formatos para el almacenamiento de las geometrías.

Well-known text (WKT). Es un formato de marcado de texto, para representar objetos de geometría vectorial en un mapa. Como se observa el la Figua 6.

Figura 6. Ejemplos de geometrías WKT.

```
POINT(6 10)
LINESTRING(3 4,10 50,20 25)
POLYGON((1 1,5 1,5 5,1 5,1 1),(2 2, 3 2, 3 3, 2 3,2 2))
MULTIPOINT(3.5 5.6,4.8 10.5)
MULTILINESTRING((3 4,10 50,20 25),(-5 -8,-10 -8,-15 -4))
```

Well-known binary (WKB). Equivalente binario para WKT usado para transferir y almacenar la misma información en bases de datos.

2.4.3. Renderización de mapas. Es el proceso de generar imágenes a partir de un modelo, para este caso consistente en las geometrías de los elementos del mapa, que están expresadas como estructuras de datos.

Dicho proceso toma en cuenta las geometrías e información como la porción del mapa que se desea renderizar, las dimensiones y formato de la imagen resultante, colores,

simbologías, escala y datos sobre el etiquetado de atributos entre otros. Distintos algoritmos de renderizado ofrecen variaciones sobre la calidad de la imagen final, su peso, y el tiempo que toma el proceso.

Estas variables se tienen en cuenta según el tipo de aplicación, por ejemplo, al tratarse de un servidor de imágenes para la Web, se busca imágenes de peso reducido y rápida generación, mientras que en una aplicación de escritorio es posible enfatizar en la calidad.

En función de estas variables cada algoritmo puede o no ofrecer características como, detección de colisión de etiquetas, etiquetas autoajustadas a la geometría, estilos de fuente, rasgos dependientes de la escala y alisado de bordes, entre otras.

2.5. CARTOGRAFÍA EN LA WEB

En inglés se traduce por Web Mapping. Se refiere a la visualización o provisión de datos geoespaciales a través de Internet.

La llegada de la cartografía Web puede ser considerada como una nueva e importante tendencia en la cartografía. Anteriormente, la cartografía se limitaba a unas pocas empresas, institutos y organizaciones cartográficas, requiriendo costosos y complejos equipos y programas informáticos, así como cartógrafos calificados. Con cartografía Web, las tecnologías cartográficas y datos geográficos de libre disposición, permiten a cualquier persona capacitada producir mapas en la Web.

La transferencia fácil y a bajo costo de datos geográficos a través de Internet permite la integración de fuentes de datos distribuidas, abriendo oportunidades que van más allá de las posibilidades del almacenamiento separado de datos.

2.5.1. Tipos de mapas Web. Una primera clasificación de los mapas Web fue hecha por Kraak, [5] quien clasificó los mapas Web en estáticos y dinámicos, así como interactivos y solo para visualización. Sin embargo, hoy a la luz de un número creciente de tipos de mapas Web, esta clasificación necesita alguna revisión.

- Estáticos. Únicamente para ver, sin animación ni interactividad. Se crean solo una vez, y se actualizan manualmente. Usualmente estos mapas son escaneados de mapas en papel y no han sido diseñados para pantalla.

- Dinámicos. Estos mapas son creados en demanda cada vez que el usuario recarga las páginas Web, usualmente desde fuentes de datos dinámicas, como bases de

datos. El servidor Web genera el mapa usando un servidor de mapas Web o un software a la medida.

- Mapas Web reusables. Son usualmente sistemas más complejos que ofrecen a otras personas librerías para reutilizar los mapas en sus páginas Web y productos. Un ejemplo de estos sistemas con librerías para reutilización es GoogleMaps con su Google Maps API.

- Mapas Web interactivos. La interactividad es una de las principales ventajas de los mapas basados en pantalla y los mapas Web. Esta ayuda a compensar sus desventajas. La interactividad ayuda a la exploración, permite cambiar los parámetros, navegar e interactuar con el mapa, revelar información adicional, enlaces a otros recursos, y mucho más.

2.5.2. Tecnologías de mapas Web del lado del servidor

- Servidor Web. El servidor es responsable del manejo de las peticiones http provenientes de los clientes. En el caso más simple sirven archivos estáticos como paginas HTML y archivos de imagen. Los servidores Web también administran autenticación, negociación de contenido, reescritura de URL y redirección de peticiones a recursos dinámicos como aplicaciones CGI.

- CGI (Common Gateway Interface). Las aplicaciones son ejecutables que están corriendo en el servidor Web en el ambiente y permisos del usuario respectivo. Pueden ser escritos en cualquier lenguaje de programación (compilado) o lenguaje de script. Una aplicación CGI implementa el protocolo de interfaz común de puerta de enlace, procesa la información enviada por el cliente y envía de vuelta el resultado en un formato legible.

- Un servidor de aplicaciones es un motor de software que entrega aplicaciones a clientes. Además, maneja la mayor parte, si no toda la lógica de negocio y acceso a datos de la aplicación. El principal beneficio de un servidor de aplicaciones es la facilidad de desarrollo y mantenimiento.

- Las bases de datos espaciales suelen ser bases de datos objeto-relacionales enriquecidas por tipos de datos, métodos y propiedades geográficas. Son necesarias siempre que una aplicación de cartografía Web tenga que tratar con datos dinámicos o con gran cantidad de datos geográficos. Las bases de datos espaciales permiten consultas espaciales, sub-selecciones, re-proyecciones, manipulación de geometría y ofrecen diversos formatos de importación y exportación.

2.5.3. Tecnologías de mapas Web del lado del cliente

- Navegador Web. En la configuración más simple, solo se requiere un navegador Web. Todos los navegadores modernos soportan el mostrar HTML e imágenes raster, algunas soluciones requieren extensiones adicionales. Para el desarrollo de aplicaciones de cartografía Web sobre navegadores son necesarias tecnologías como:

  El document object model en español modelo de objetos de documento, frecuentemente abreviado DOM, es una forma de representar los elementos de un documento estructurado (tal como una página Web HTML o un documento XML) como objetos que tienen sus propios métodos y propiedades. En efecto, el DOM es una API para acceder, añadir y cambiar dinámicamente contenido estructurado en documentos con lenguajes como javascript.

  Javascript es un lenguaje de scripting usado para desarrollo Web del lado del cliente, es una implementación del estándar ECMAScript. Sin embargo el DOM no hace parte de javascript y en la práctica las implementaciones de cada navegador difieren del estándar y entre ellas.

  Las hojas de estilo en cascada (cascading style sheets, CSS) son un lenguaje formal usado para definir la presentación de un documento estructurado escrito en HTML o XML. La idea que se encuentra detrás del desarrollo de CSS es separar la estructura de un documento de su presentación.

  AJAX, acrónimo de asynchronous javascript y XML, es una técnica de desarrollo Web para crear aplicaciones interactivas o RIA (rich Internet applications). Éstas se ejecutan en el cliente, es decir, en el navegador de los usuarios y mantienen comunicación asíncrona con el servidor en segundo plano. De esta forma es posible realizar cambios sobre la misma página sin necesidad de recargarla. Esto significa aumentar la interactividad, velocidad y usabilidad en la misma. Para ello se emplea manipulación DOM, CSS y el objeto XMLHttpRequest o iframes, sin embargo, en casos como aplicaciones hibridas las restricciones de seguridad impuestas sobre estos métodos llevan a los desarrolladores a buscar métodos alternativos como etiquetas script dinámicas y JSON (javascript object notation).

- Dispositivos móviles. Existen diferentes tecnologías que permiten el desarrollo de aplicaciones para dispositivos móviles entre ellas se encuentran:

Symbian. Está limitado a ciertos grupos de terminales y puede ser considerada como un sistema operativo, presenta acceso a elementos de bajo nivel.

.NET Compact Framework. Ideal para desarrollo en dispositivos Pocket-PC. Sin embargo no es de plataforma cruzada y está limitado a dispositivos Microsoft.

Flash Lite. Ideal para opciones gráficas pesadas en el mercado que pueda soportar el reproductor Flash Lite.

Java ME. Ideal para una solución portátil, en caso de que la plataforma Java ME proporcione la funcionalidad requerida. Existen bibliotecas específicas para muchos dispositivos y se utilizan para los juegos, lo que les hace no portátiles.

Los dispositivos Java ME implementan un perfil. Los más comunes son el MIDP (mobile information device profile) orientado a dispositivos como celulares y el PP (personal profile) orientado a dispositivos como PDAs.

Los perfiles son subconjuntos de las configuraciones, de aquellas actualmente existen dos: CLDC (connected limited device configuration) y CDC (connected device configuration).

- Aplicaciones de escritorio. Es posible desarrollar aplicaciones para cartografía Web que trabajen en ambientes de escritorio. La plataforma J2SE de sun, proporciona varios paquetes útiles para el trabajo con este tipo de aplicaciones:

  El paquete Java.net, suministra rutinas especiales IO para redes, permitiendo las peticiones HTTP, así como también otras transacciones comunes.

  El paquete Java.io contiene clases que soportan entrada/salida mediante flujos. En conjunto con las clases de Java.net permite acceso a archivos vía http.

  Java 2D es un API para dibujar gráficos en dos dimensiones usando Java. Contiene también las clases necesarias para el manejo de eventos del mouse y teclado.

  javax.swing es una colección de rutinas que se construyen sobre Java.awt, usa las rutinas de dibujado 2D para renderizar los componentes de interfaz de usuario en lugar de confiar en el soporte GUI nativo subyacente del sistema operativo.

2.6. WMS. [6] El servicio Web map service WMS definido por el OGC produce mapas de datos espaciales referidos de forma dinámica a partir de información geográfica. Este estándar internacional define un "mapa" como una representación de la información

geográfica en forma de un archivo de imagen digital conveniente para la exhibición en pantalla. Los mapas producidos por WMS se generan normalmente en un formato de imagen como PNG, GIF o JPEG, y ocasionalmente como gráficos vectoriales en formato SVG (scalable vector graphics).

Un WMS básico debe soportar las operaciones GetCapabilities y GetMap. Un WMS consultable debe soportar además la operación opcional GetFeatureInfo.

El estándar se presenta en diferentes versiones 1.0, 1.1, 1.1.1, y la actual 1.3, al establecer una comunicación entre cliente y servidor el primer paso se denomina negociación de versiones y consiste en hallar una versión mutuamente aceptable para ambos.

El estándar define la implementación de WMS en una plataforma de informática distribuida que comprende servidores que soporten http. Http soporta dos métodos de petición get y post, de los cuales get es obligatorio y post opcional.

La respuesta a una solicitud WMS es siempre un archivo que se transfiere a través de Internet desde el servidor al cliente. El archivo puede contener texto o imagen. El tipo del archivo retornado se indica mediante un tipo MIME.

El estándar usa dos clases principales de sistemas de coordenadas: un map CS aplicable al dibujo de mapa generado por el WMS, y un layer CRS para un boundig box aplicado en los datos de origen. Durante la operación de dibujo, un WMS convierte o transforma información geográfica de un layer CRS a un map CS.

2.6.1. Elementos básicos del servicio

- Map CS. Es un sistema de coordenadas de referencia para un mapa producido por un WMS. Un mapa WMS es una malla rectangular de píxeles presentada en una pantalla de computador. El map CS tiene un eje horizontal denotado por i, y un eje vertical denotado por j. El origen (i, j) = (0, 0) es el píxel en la esquina superior izquierda del mapa; i incrementa hacia la derecha y j incrementa hacia abajo.

  Un layer CRS es un sistema de coordenadas de referencia horizontal para la información geográfica que sirve como fuente para un mapa. Un WMS debe soportar al menos un CRS.

- Layer CRS. Tiene dos ejes, denotados x, y. El eje “x” es el primero en la definición del CRS el eje “y” es el segundo. Dependiendo del CRS en particular, el eje “x” puede o no estar orientado de oeste a este, y el eje “y” puede o no estar orientado

de sur a norte. La operación de dibujo debe tener en cuenta el orden, origen y dirección del layer CRS.

Muchos sistemas de coordenadas de referencia están orientados con un eje positivo al este y otro positivo al norte. Sin embargo, algunos CRS tienen coordenadas incrementando en otras direcciones.

- Bounding box. Especifican la porción de la tierra a mapear mediante dos pares de coordenadas en un layer CRS especificado. El primer par especifica los valores de coordenadas mínimos y el segundo los máximos.

2.6.2. Operaciones

1. GetCapabilities. Retorna metadatos del servicio, esta información se expresa en forma de un documento XML fiel a un esquema definido por el mismo estándar, declara las capacidades del servidor en lo referente al material del que dispone, los formatos en que puede entregarlo, las direcciones a las que debe recurrirse para hacer las peticiones e información de contacto entre otras, los parámetros de la petición se muestran en la Tabla 3.

Tabla 3. Parámetros de una URL de petición GetCapabilities.

Parámetro	**Obligatorio**	**Descripción**
VERSION=versión	No	Versión de la petición
SERVICE=WMS	Si	Tipo del servicio
REQUEST=GetCapabilities	Si	Nombre de la petición
FORMAT=tipo MIME	No	Formato de salida de los metadatos del servicio
UPDATESEQUENCE=cadena	No	Cadena para control de caché

2. GetMap. Retorna un mapa dados parámetros geográficos y dimensionales bien definidos (ver tabla 4), un ejemplo de esta petición se muestra en la figura 7.

Tabla 4. Parámetros principales de una URL de petición GetMap.

Parámetro	**Obligatorio**	**Descripción**
VERSION=1.3.0	Si	Versión de la petición
REQUEST=GetMap	Si	Nombre de la petición
LAYERS=lista	Si	Lista separada por comas de capas del mapa
STYLES=lista	Si	Lista separada por comas de estilos por capa
CRS=namespace:identificador	Si	Sistema de coordenadas de referencia
BBOX=minx, miny, maxx, maxy	Si	Esquinas del bounding box
WIDTH=ancho_salida	Si	Ancho en píxeles de la imagen
HEIGHT=alto_salida	Si	Alto en píxeles de la imagen
FORMAT=formato_salida	Si	Formato de salida del mapa
TRANSPARENT=TRUE|FALSE	No	Fondo transparente (por defecto=FALSE).
BGCOLOR=color	No	Valor hexadecimal, por defecto blanco
EXCEPTIONS=formato	No	Formato para las excepciones (por defecto=XML).

Figura 7. Ejemplo de petición GetMap.

```
http://192.168.10.111:8080/server/WMS?request=GetMap&version=1.3.0&styl
es=&format=image/PNG&crs=EPSG:21891&layers=3,13&width=256&height=25
6&bbox=977714.9347524977,625907.6674601873,977820.1373207929,626012
.8700284826
```

3. GetFeatureInfo. Es una operación opcional diseñada para proporcionar a los clientes de un WMS más información sobre las características de las imágenes de mapas que fueron devueltas en peticiones de mapas anteriores. El usuario ve la respuesta de una solicitud de mapa y elige un punto (I, J) para obtener más Información. La operación proporciona al cliente la capacidad de especificar el píxel, que capas deben ser investigadas, y en que formato debe devolverse la información. Como el protocolo carece de estado la petición GetFeatureInfo debe indicar al WMS lo que el usuario está viendo mediante la inclusión de la mayoría de los parámetros originales de la petición GetMap. El estándar no define el contenido de la respuesta a esta solicitud, es decir, queda a discreción del proveedor del servicio WMS.

2.7. APLICACIONES HÍBRIDAS

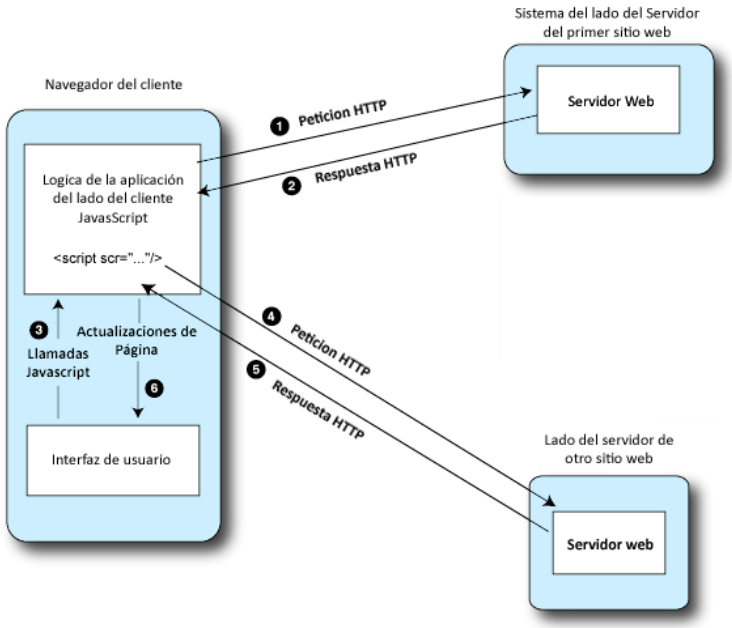
Es cada vez más común visitar sitios Web, que por ejemplo, muestren una lista de restaurantes en una ciudad, y entonces solicitar al sitio que muestre un mapa que resalte la ubicación de los restaurantes. Aunque se puede asumir que la lista y el mapa provienen de recursos dentro del sitio, los dos tipos de información usualmente provienen de sitios diferentes. De forma más específica, los servicios que generan los dos tipos de contenido están en diferentes sitios. Esta fusión de servicios y contenidos de múltiples sitios Web de forma integrada y coherente es denominada mashup, aplicaciones híbridas, en español. La mayoría de las aplicaciones híbridas hacen más que simplemente integrar servicios y contenido, típicamente agregan valor. La suma de los servicios resulta más rica que los servicios cada uno por separado. [7]

2.7.1. Estilos de mashup. Los dos estilos principales de mashup son mashup del lado del cliente y mashup del lado del servidor.

Lado del Servidor. Los servicios y contenidos se integran en el servidor. El servidor actúa como un proxy entre una aplicación Web en el cliente (típicamente un navegador), y el otro sitio que toma parte en el mashup.

Lado del cliente. En este caso los servicios y contenidos se integran en el cliente, es decir, este accede directamente a los datos o funcionalidades del otro sitio Web. (ver figura 8).

Figura 8. Mashup del lado del cliente.



1. El navegador hace la solicitud de la página Web al servidor.
2. El servidor carga la página en el cliente. La página usualmente incluye una librería del sitio de mashup para posibilitar el acceso a los servicios de este.
3. Alguna acción en la página llama a una función en la librería javascript provista por el sitio del mashup. La función crea un elemento <script> que apunta al sitio del mashup.
4. Con base en el elemento <script>, se hace una petición al sitio del mashup para cargar el script.
5. El sitio del mashup carga el script. Típicamente, una llamada local en la página del navegador se ejecuta con un objeto JSON enviado como parámetro.
6. La función de retorno actualiza la vista del cliente manipulando el DOM que representa la página.

3. GEOCODIFICACIÓN

Es el proceso de asignar una localización, usualmente en forma de coordenadas, a una dirección, comparando elementos descriptivos de tal dirección con aquellos presentes en el material de referencia. Las direcciones vienen en varias formas, en esencia, una dirección incluye cualquier información útil a la hora de identificar un lugar. [8]

En el mundo real, para encontrar lugares se usan datos descriptivos. Esto podría ser un número y nombre de la calle. Puede incluir el nombre de la ciudad, región, país, o características naturales.

El proceso de geocodificación comienza con una descripción textual de un lugar que se traduce en las coordenadas x, y que se pueden trazar en el mapa.

Es de vital importancia contar con el material adecuado. Es imposible localizar una dirección en una ciudad si solo de dispone de un mapa de otra, y no es posible localizar una dirección con toda exactitud si únicamente se dispone de un mapa de autopistas. El mapa debe tener suficiente detalle de la zona para poder identificar la ubicación que se está buscando.

Sucede de igual modo con un sistema de geocodificación, las capas que utiliza, conocidas como datos de referencia, también deben tener los detalles del punto concreto que se espera encontrar. Los principales datos de referencia por lo general consisten en una red de las calles, aunque también se puede usar un mapa de parcelas. Lo importante es que los datos tengan el nivel de detalle que se desea encontrar.

3.1. EL PROCESO DE GEOCODIFICACIÓN

El proceso de geocodificación incluye el tratamiento de direcciones alfanuméricas semiestructuradas de eventos (análisis), el establecimiento de una correspondencia entre la dirección estructurada y la base de datos de direccionamiento (correspondencia), y la asignación de coordenadas al evento (ubicación). Para realizar las tareas de análisis, correspondencia y ubicación, el proceso de geocodificación necesita acceso a una base de datos que almacene información sobre el sistema de direccionamiento. [9]

La fase de análisis consiste en el trabajo con campos de texto que contienen información sobre la dirección y su transformación a un formato estructurado y estandarizado. Esto puede lograrse mediante varios algoritmos orientados a cadenas de texto.

En el proceso también es usual tratar con omisiones y errores comunes en datos de dirección sin estructurar, buscando establecer una correspondencia entre ellos y los nombres “oficiales” de los lugares. Naturalmente la información de nombres de lugares debe estar disponible para consulta en la base de datos. También es útil tener información sobre nombres de lugares populares o desactualizados.

La fase de correspondencia trata de buscar en la base datos la referencia más precisa posible asociada con la dirección dada. Los resultados de la fase de análisis pueden estar incompletos, y dependiendo de la información faltante debe tomarse una decisión sobre cuál de las referencias disponibles a una ubicación generará los resultados más precisos.

Para que esto suceda debe haber dos grupos en base de datos. El primero se refiere a la estructura de direccionamiento actual. El segundo incluye cualquier información adicional que pueda usarse para resolver ambigüedades o referencias geográficas incompletas.

Finalmente la fase de ubicación recibe los resultados de la correspondencia y determina las coordenadas actuales de la dirección. El proceso puede ser trivial o incluir distintas operaciones geográficas.

3.2. REFERENCIA LINEAL

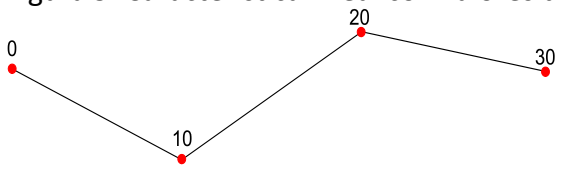
Referencia lineal es un método para manejar ubicaciones geográficas usando posiciones a lo largo de una característica lineal medida, por ejemplo, ubicaciones de incidentes de tránsito se describen usando convenciones como, “30 metros al este del kilómetro 25 vía sur”. [10]

Usualmente se requiere almacenar varios atributos adicionales sobre las vías, sin el uso de referencia lineal, esto podría requerir que tales vías sean dividas en muchos segmentos pequeños en cada lugar donde los valores de los atributos cambien. Como una alternativa, estas situaciones pueden ser manejadas como eventos de referencia lineal a lo largo de las vías.

Una ruta es una característica lineal que tiene un sistema de medida definido. Estas medidas pueden usarse para localizar eventos y condiciones a lo largo de su conjunto de características lineales.

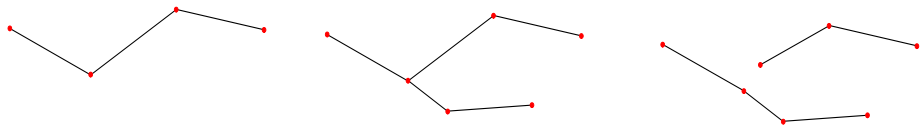
En las rutas los rasgos lineales tienen sus coordenadas (x, y) que describen la ubicación así como un valor de medida a lo largo de la línea (m), como se muestra en la figura 9.

Figura 9. Característica lineal con valores de medida.



Las características lineales simples son representadas por líneas de un camino, características lineales complejas se representan por líneas de múltiples caminos, como se muestra en la figura 10.

Figura 10. Características lineales simples y complejas.



Línea de un solo camino

Línea con múltiples caminos conectados

Línea con múltiples caminos desconectados

Es importante notar que a pesar de que muchas aplicaciones usan medidas para representar distancias incrementales a lo largo de características lineales, los valores de medida pueden incrementar, disminuir, o permanecer constantes a lo largo de una línea de forma arbitraria.

Los valores de las medidas son independientes de el sistema de coordenadas, es decir, no se requiere que los valores de medida estén en las mismas unidades que las coordenadas (x, y) de las líneas.

Las tablas de eventos contienen información sobre condiciones y eventos que pueden ser localizados a lo largo de rutas. Cada elemento en la tabla de eventos referencia un evento y su localización expresada como medida a lo largo de una ruta particular.

La segmentación dinámica es el proceso de calcular ubicaciones de eventos almacenadas y administradas en una tabla de eventos usando medidas de referencia lineal y mostrarlas en un mapa. El término segmentación dinámica es derivado del concepto de que las características lineales no necesitan se segmentadas cada vez que los valores en los atributos cambien, es decir, el segmento puede ser localizado dinámicamente.

Usando segmentación dinámica múltiples conjuntos de atributos se pueden asociar con cualquier porción de una característica lineal existente. Estos atributos pueden ser mostrados, consultados, editados y analizados sin afectar la subyacente geometría lineal.

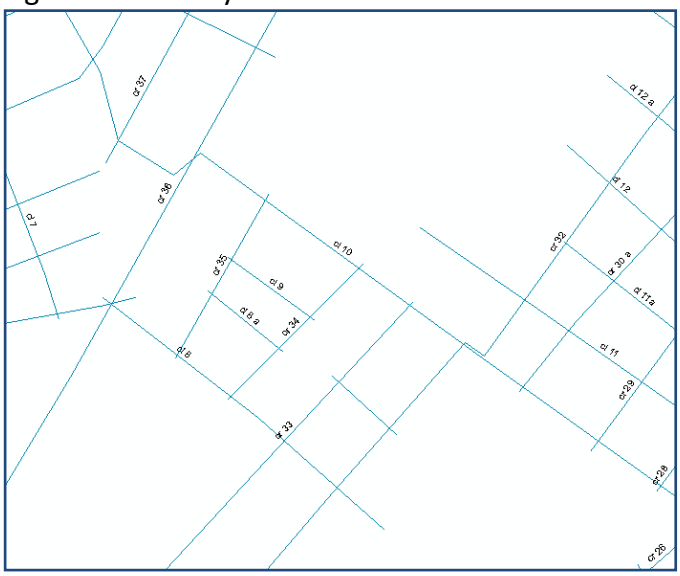
3.3. NOMENCLATURA Y NUMERACIÓN URBANA

La operación de nomenclatura y numeración urbana permite a las personas localizar un lote de terreno o una vivienda, es decir, definir su dirección por medio de un sistema de planos y letreros de calles que indican los números o los nombres de las calles y los edificios. Es también una base indispensable para el ordenamiento urbano.

3.3.1. Numeración y nomenclatura urbana en San Juan de Pasto. Para el municipio de San Juan de Pasto el ente encargado de la gestión, asignación y mantenimiento de la nomenclatura urbana es la alcaldía municipal mediante el departamento de planeación municipal.

Las vías de la ciudad se clasifican en calles y carreras, siendo las calles las que tienen números crecientes en sentido occidente a oriente y las carreras las vías cuya numeración avanza en sentido sur a norte, como se muestra en la siguiente figura.

Figura 11. Calles y carreras de San Juan de Pasto.



Según su tipo una vía se nombra con la palabra calle o carrera seguida de un número arábigo y en ciertos casos de una letra, por ejemplo, cuando se abre una vía en medio de un par de vías ya existentes.

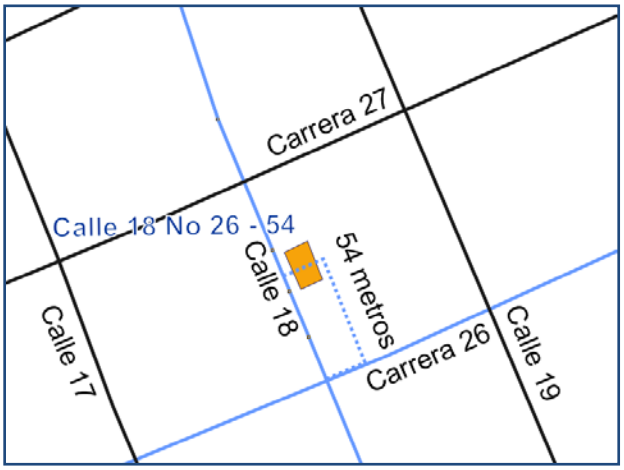
La numeración arábiga crece indefinidamente en sentido positivo sin embargo, de requerir nombrar vías con un número inferior a 1, como no se emplea el cero ni números

negativos las vías se nombran aumentado las palabras este o sur según corresponda al sector de la ciudad.

Una cuadra se origina de la intersección entre una vía y otras dos vías de orientación opuesta, por ejemplo, la cuadra formada en la calle 18 entre carreras 26 y 27, esta cuadra se denomina calle 18 No 26.

Para nombrar una propiedad se coloca el nombre de la cuadra en la que se encuentra, para diferenciar las propiedades dentro la misma cuadra se agrega un número correspondiente a la distancia en metros desde la puerta de propiedad hasta la esquina formada por la intersección con la vía de inferior valor, por ejemplo, la dirección calle 18 No 26 – 54, indica que la propiedad se encuentra sobre la calle 18 en la cuadra formada por la intersección de esta con la carrera 26 y la próxima carrera de valor superior, la dirección también indica que la propiedad se encuentra aproximadamente a 54 metros de la esquina con la carrera 26 como se muestra en la figura 12. Normalmente el último número es par para indicar casas a la derecha de la vía e impar para indicar casas a la izquierda, en este caso para determinar el sentido izquierdo o derecho debe recorrerse la vía en el sentido de avance de las vías de orientación contraria, por ejemplo para el caso de una calle debe recorrerse en el sentido de avance de las carreras.

Figura 12. Ubicación de una dirección en San Juan de Pasto.



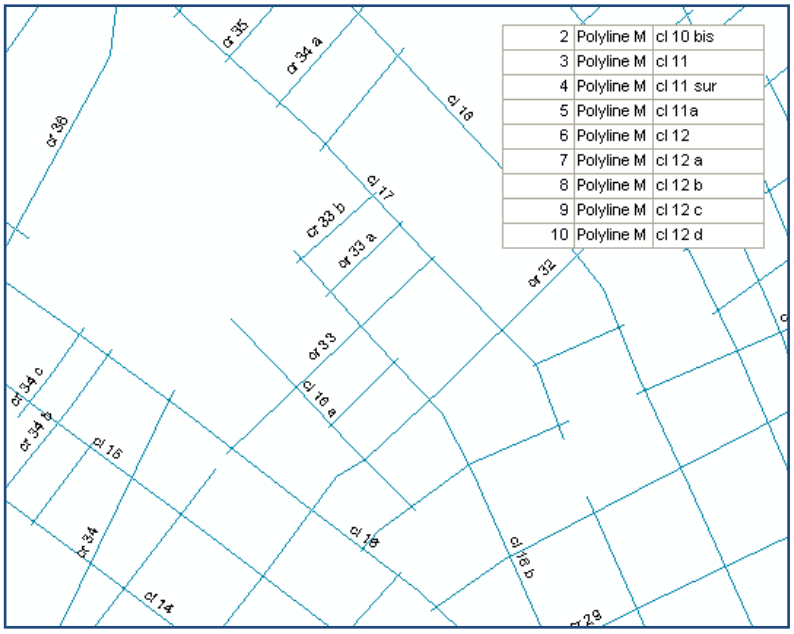
3.4. ATLAS STREET CROSSING GEOCODER

Se plantea un algoritmo de geocodificación titulado Atlas street crossing, creado específicamente para demostrar las capacidades de gestión de plugins de geocoders de la herramienta Atlas y resolver direcciones en la nomenclatura de la zona centro de la ciudad de San Juan de Pasto.

3.4.1. Material de referencia. El algoritmo requiere como información de referencia la malla vial de la ciudad representada por geometrías de tipo poli línea, donde cada geometría tiene asociado un valor textual único en el conjunto de datos que contiene el nombre de la vía, así como un identificador numérico único. La geometría está expresada en metros. (ver figura 13).

El nombre la vía debe iniciar con las letras CL o CR según se trate de una calle o una carrera, luego un espacio y el número de la vía, seguido por un espacio y una letra o la palabra ESTE o la palabra SUR.

Figura 13. Material de referencia del Atlas street crossing geocoder.



Dada la geometría se procede a calcular los valores M según lo indicado por las técnicas de referencia lineal usando la distancia geométrica acumulada entre punto y punto como se muestra en la tabla 5, en el caso geometrías con más de un segmento, el primer punto

del primer segmento iniciará con el valor M de 0, los siguientes segmentos iniciarán con el valor M mayor del segmento anterior.

Tabla 5. Calculo de los valores M.

Segmento	X	Y	M
Segmento 1	10	5	0
	20	3	10,19804
	12	20	28,98633
Segmento 2	15	23	28,98633
	20	35	41,98633
	23	42	49,60211

Una vez calculados lo valores M se procede a calcular las intersecciones de cada vía. Consignando por cada intersección el identificador de la vía y el valor M del lugar de la intersección, por cada vía los datos de sus intersecciones se deben ordenar ascendentemente respecto a la columna de valores M.

Los datos anteriores deben estar disponibles al momento de hacer una consulta a geocoder, inclusive, se recomienda mantenerlos pre calculados y almacenados, ya que constituyen la base de geocoder y son invariables entre diferentes consultas.

3.4.2. Fases del proceso. El algoritmo define procesos que deben realizarse en cada fase del proceso de geocodificación.

- **Durante la fase de análisis.** se reemplazan los términos comunes como calle, cll, cle o términos del léxico de algoritmo como CL o CR en caso de carreras, así mismo se remueven caracteres de puntuación, en la tabla 6 se muestran todos los símbolos y sus remplazos respectivos, la cadena de entrada se convierte a mayúsculas, se eliminan los espacios iniciales y finales y las ocurrencias de más de un espacio consecutivo se reemplazan por un solo espacio, también se remueven símbolos que puedan hacer referencia número y se remueven las tildes de las vocales.

  El objetivo es tomar una cadena de entrada como Calle 9 No 34 – 30 Las acacias y transformarla en CL 9 34 30 LAS ACACIAS.

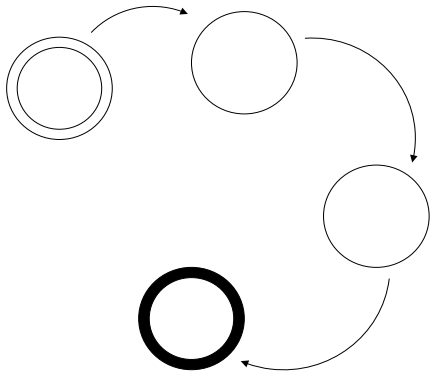
Tabla 6. Reemplazos en el proceso de análisis.

Símbolo	**Reemplazo**
CALLE, CLE, CLLE, CAL, CLL CL	CL
CRA, KRA, KR, CARRERA, CRRA, CR	CR
n0, #, no, num, n	Espacio
CON, DE, EL, EN, LA, LAS, LO, LOS	Espacio
.,;:-_	Espacio
Á	A
É	E
Í	I
Ó	O
Ú	U

Una vez con la cadena en este formato se procede a identificar y separar sus partes.

Tipo de la vía principal. El tipo de la vía sobre la que se encuentra ubicada la propiedad, es la primera parte de la cadena que cumpla con la expresión presentada en siguiente figura.

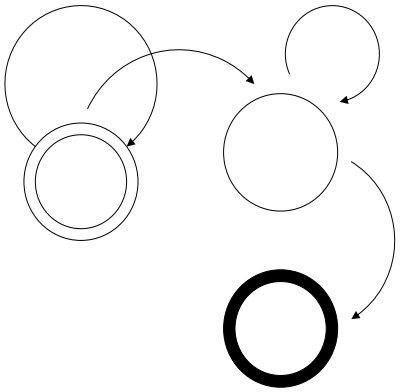
Figura 14. Máquina de estado para identificar el tipo de la dirección.



Nombre de la vía principal. El nombre de la vía sobre la que se encuentra ubicada la propiedad, es la primera parte de la cadena a partir del tipo que cumpla la expresión presentada en la figura 15, el dígito que lleva al estado final no hace parte del nombre y debe ser regresado, solo se usa para terminar. De no existir un

carácter de espacio entre la parte numérica y alfanumérica del nombre, dicho espacio debe ser agregado.

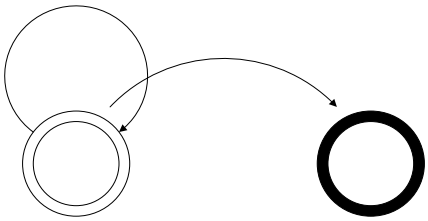
Figura 15. Máquina de estado para identificar el nombre de una vía.



Nombre de la vía de intersección. El nombre de la vía que intercepta la vía principal es la primera parte de la cadena a partir del nombre de la vía principal, que cumple la anterior expresión, el dígito que lleva al estado final no hace parte del nombre y debe ser regresado, solo se usa para terminar.

Número de la propiedad. El número de metros entre la esquina de la cuadra y la puerta de la propiedad, la naturaleza par o impar de este número indicara el lado del la vía sobre la que se ubica la propiedad. Será la primera sub cadena después del nombre la vía de intersección que cumpla con la siguiente expresión. (ver figura 16)

Figura 16. Máquina de estado para identificar el número de la propiedad.



Aplicando lo anterior debe obtenerse resultados como los mostrados en los ejemplos de la siguiente tabla. Al final del proceso debe obtenerse al menos el tipo de la dirección y los nombres de la vía principal y la vía de intersección, de no ser así se dirá que la dirección no cumple con los requisitos mínimos para el proceso.

Tabla 7. Ejemplos direcciones analizadas.

Dirección Ingresada	Dirección estandarizada	Vía principal	Intersección	Número de casa
Calle 9 No 34 - 30	CL 9 34 30 LAS ACACIAS	CL 9	34	30
Ed. Madrigal Cl. 24 # 25 - 10 OF 301	ED MADRIGAL CL 21 25 10 OF 301	CL 21	25	10
Cll 18 26 - 54 Centro	CL 18 26 54 CENTRO	CL 18	26	54
Cra 21A - 12 - 53	CR 21 A 12 53	CR 21 A	12	53

- **Durante la fase de correspondencia.** Se trabaja con las partes generadas en la operación anterior, el propósito aquí es ubicar la vía principal y la vía de intersección en el material de referencia. El tipo de la vía de intersección no se encuentra explícito en la dirección, sin embargo, se sabe que si la vía principal es una calle la intersección será una carrera y viceversa, hecho que permite completar esta información. Entonces es posible comparar con el material de referencia, solicitando las vías que coincidan en, tipo, la parte numérica del nombre y la parte alfanumérica del nombre, si de este modo no se hallan resultados, se flexibiliza la búsqueda al ignorar la parte alfanumérica del nombre de la vía.

  Al final de la fase se ha obtenido 2 listas, la primera con una o más posibles vías principales y la segunda con una o más vías de intersección, si no es posible formar estas listas se dirá que no se cuenta con material de referencia suficiente para procesar la dirección.

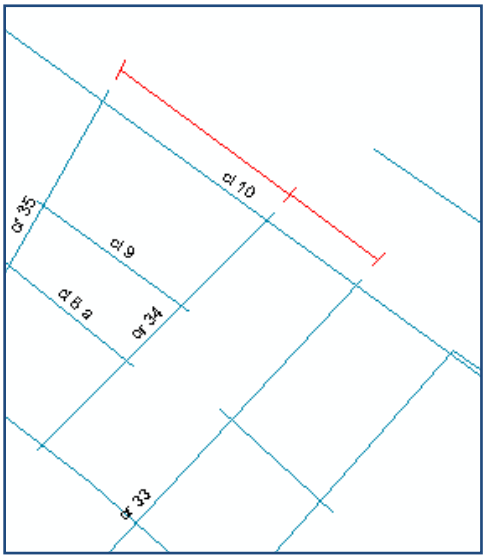
- **Durante la fase de ubicación.** Se trabaja con las partes generadas en la fase de análisis y las dos listas de la fase de correspondencia, así como la lista de intersecciones de cada vía que se encuentra en el material de referencia. El objetivo aquí es generar un listado de coordenadas correspondientes con la dirección textual solicitada.

  Por cada elemento en la lista de vías principales se toma cada elemento de la lista de vías de intersección, por cada par, se toma la lista de intersecciones de la vía principal y se busca en ella la vía de intersección, de encontrarla en la lista se toma también la vía de intersección anterior y la siguiente.

  De modo que, si por ejemplo, se tiene como vía principal la CL 10 y como vía de intersección la CR 34 al consultar la tabla de intersecciones de la CL 10 se tomará 3 elementos, la CR 33, la CR 34 y la CR 35.

Con estos 3 elementos se forman dos cuadras sobre la CL 10, como se muestra en la siguiente figura.

Figura 17. Formación de cuadras.



Por la definición de la nomenclatura urbana de la ciudad se sabe que la cuadra en que se encuentra la propiedad es la formada entre la intersección denotada en la dirección y la intersección con otra vía de mayor valor, para este caso, la cuadra de interés es la formada por las carreras 34 y 35.

Para determinar la vía mayor se deben remover todos los caracteres no numéricos del nombre de la vía, y luego comparar los valores enteros resultantes.

Una vez identificada la cuadra, se suma el número de la casa al valor M de la intersección principal, para este caso, entre la calle 10 y la carrera 34.

Este valor M se convierte a coordenadas (x, y) usando las coordenadas (x, y, m) de la geometría de la vía principal y esta es la ubicación de la dirección solicitada.

4. ANTECEDENTES

En este capítulo se realiza una recopilación de las distintas herramientas que trabajan con cartografía Web y herramientas que prestan el servicio geocodificación, se encuentran clasificadas en 3 categorías, servidores de cartografía, bibliotecas de desarrollo y servicios de geocodificación.

4.1. GEOSERVER

Es un servidor de código abierto distribuido bajo licencia GNU enfocado en conectar información existente referenciada geográficamente a la Web, usando estándares abiertos. [11] El logotipo de geoserver se muestra en la figura 18.

Figura 18. Logotipo de geoserver.

http://www.photon.com/Images/index/GeoServer.jpg

Se inició en una organización sin ánimo de lucro denominada TOPP (del inglés The Open Planning Project), para ayudar a construir una infraestructura interoperable de datos espaciales promocionando un software de alta calidad, fácil de usar, de código abierto, a potenciales proveedores de datos.

Ha evolucionado para convertirse en un método fácil para conectar la información existente a mundos virtuales como Google Earth y NASA World Wind, así como también a sitios de cartografía Web, como Google Maps, Windows Live Local y Yahoo Maps.

GeoServer es también la implementación de referencia para servidores que cumplan el estándar Web Feature Server en su versión 1.0, también implementa el estándar Web Map Server en su versión 1.1.1.

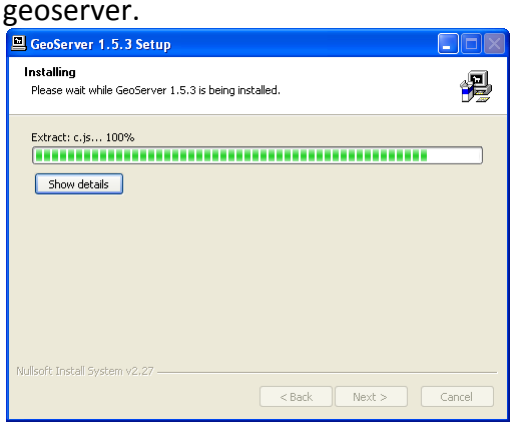
Al implementar estándares abiertos geoserver produce formatos como JPEG, GIF, SVG, PNG, entre otros. Además es capaz de leer una variedad de formatos de datos incluyendo PostGIS, Oracle Spatial, ArcSDE, Shapefiles, etc.

Requisitos Software. Geoserver es una aplicación escrita 100% en Java y para su instalación requiere que esté instalado en el equipo de despliegue Java JSDK (1.4 o 1.5), como también un contenedor del servlets.

Instalación. La versión actual de Geoserver se encuentra disponible en http://geoserver.org/display/GEOS/Latest el proceso de instalación depende del sistema operativo, para usuarios Windows cuenta con un asistente que instala el servidor por defecto y todos sus prerrequisitos en un ambiente pre configurado como se muestra en las figuras 19 y 20.

Figura 19. Bienvenida a la instalación de geoserver.

Figura 20. Progreso de la instalación de geoserver.



Para usuarios Linux se encuentra un archivo compreso con los fuentes de la aplicación y el lanzador del servidor por defecto, es decir con los prerrequisitos y un ambiente pre configurado.

Si se desea utilizar el servidor en un ambiente ya existente, la instalación debe hacerse manualmente siguiendo el procedimiento requerido por el contenedor de servlets.

Manejo de la Herramienta. Geoserver cuenta con un conjunto de interfaces Web, diseñadas para administrar, configurar y demostrar el funcionamiento del servidor.

Las interfaces de administración liberan los recursos consumidos por el servidor como la memoria JAI y memoria RAM, las interfaces de configuración como su nombre lo indica establecen las propiedades del servidor, los parámetros de los servicios WFS, WMS (ver figura 21), así como el ingreso de fuentes de datos al servidor (ver figura 22), las interfaces de demostración son enlaces a la visualización de los datos, documentación, peticiones de ejemplo, entre otras.

Figura 21. Metadatos WMS en geoserver.

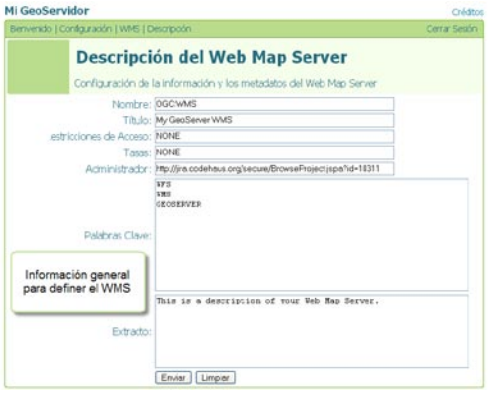


Figura 22. Editor de almacén de datos de geoserver.

Ventajas.

- Tiene interfaces para la administración.
- Es independiente de plataforma.
- Permite la carga de fuentes de datos más utilizadas.
- Se distribuye con datos de ejemplo.
- Implementa estándares internacionales.

Desventajas.

- No implementa la última versión de WMS 1.3.0.
- No ofrece características que faciliten la creación de módulos de geocodificación.
- Las interfaces de administración se orientan a la información, más no la visualización.

4.2. DEEGREE

Es un marco Java que ofrece bloques principales para la construcción de infraestructuras de datos espaciales. Toda su arquitectura se desarrolla mediante las normas del Open Geospatial Consortium, incluye servicios Web, así como clientes. [12] El logotipo de deegree se muestra en la figura 23.

Figura 23. Logotipo de deegree.

http://www.deegree.org/deegree/images/deegree/logo-deegree.PNG

Deegree ha evolucionado desde el proyecto JaGo, se publica bajo los términos de la licencia GNU Lesser Public License (GPL-L) de la fundación de software libre. La mejora generalizada de la arquitectura, el incremento del modelo de objetos y el soporte de Java 5 hace a deegree una arquitectura robusta para el desarrollo de aplicaciones SIG.

Deegree implementa los estándares para servicios Web, Web Map Service (WMS) 1.1.1, Web Feature Service (WFS) 1.1.0, Web Coverage Service (WCS) 1.0.0, Catalogue Service Web-Profile (CSW) 2.0.0.

Entre sus características más destacadas se encuentran:

- Soporte para PostGIS 1.0 y Oracle spatial/locator (9i/10g).
- Múltiples orígenes de datos para capas WMS.
- Renderizado dinámico de reglas a través de SLD.

Requisitos Software. Deegree al ser desarrollado en Java requiere Java JSDK 1.4 o superior, como también el contendor de servlets.

Instalación. La versión más reciente de Deegree se encuentra disponible en el sitio Web oficial http://www.deegree.org bajo el link de downloads, debe descargarse el archivo apropiado para la plataforma y descomprimirlo.

El proceso de instalación y configuración exige la edición de diferentes archivos XML que controlan la funcionalidad del servidor como por ejemplo, la localización de las bibliotecas y la configuración central de deegree en la operación de despliegue del contenedor con el archivo war, se debe indicar en el descriptor de despliegue Web.xml que se muestra en la figura 24.

Figura 24. XML descriptor de despliegue deegree.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>
    <servlet>
        <servlet-name>services</servlet-name>
        <servlet-class>
           org.deegree.enterprise.servlet.OGCServletController
        </servlet-class>
        <init-param>
            <param-name>services</param-name>
            <param-value>WMS</param-value>
        </init-param>
        <init-param>
            <param-name>WMS.handler</param-name>
            <param-value>org.deegree.enterprise.servlet.WFSHandler</param-value>
        </init-param>
        <init-param>
            <param-name>WMS.config</param-name>
            <param-value>/WEB-INF/conf/wfs/wfs_capabilities.xml</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>services</servlet-name>
        <url-pattern>/services</url-pattern>
    </servlet-mapping>
</web-app>
```

Todos la campos deben ser editados para la configuración del servicio

Manejo de la Herramienta. La configuración permite la utilización de todo el conjunto de parámetros que se pueden utilizar para el control de Deegree WMS (ver figura 25). Esto incluye los parámetros mencionados en la especificación OGC WMS 1.1.1 para el documento getcapabilities, así como también los parámetros técnicos específicos para describir los orígenes de datos de las capas (ver figura 26), en el archivo de configuración denominado WMS_configuration.xml.

Ventajas.

- Es la implementación WMS de referencia del OGC.
- Es independiente de plataforma.
- Se distribuye con datos de ejemplo.

Desventajas.

- No implementa la última versión de WMS 1.3.0.
- No ofrece características que faciliten la creación de módulos de geocodificación.
- No presenta ninguna interfaz gráfica para la instalación, configuración, carga de datos o administración del servicio.

Figura 25. XML de configuración servicio WMS deegree.

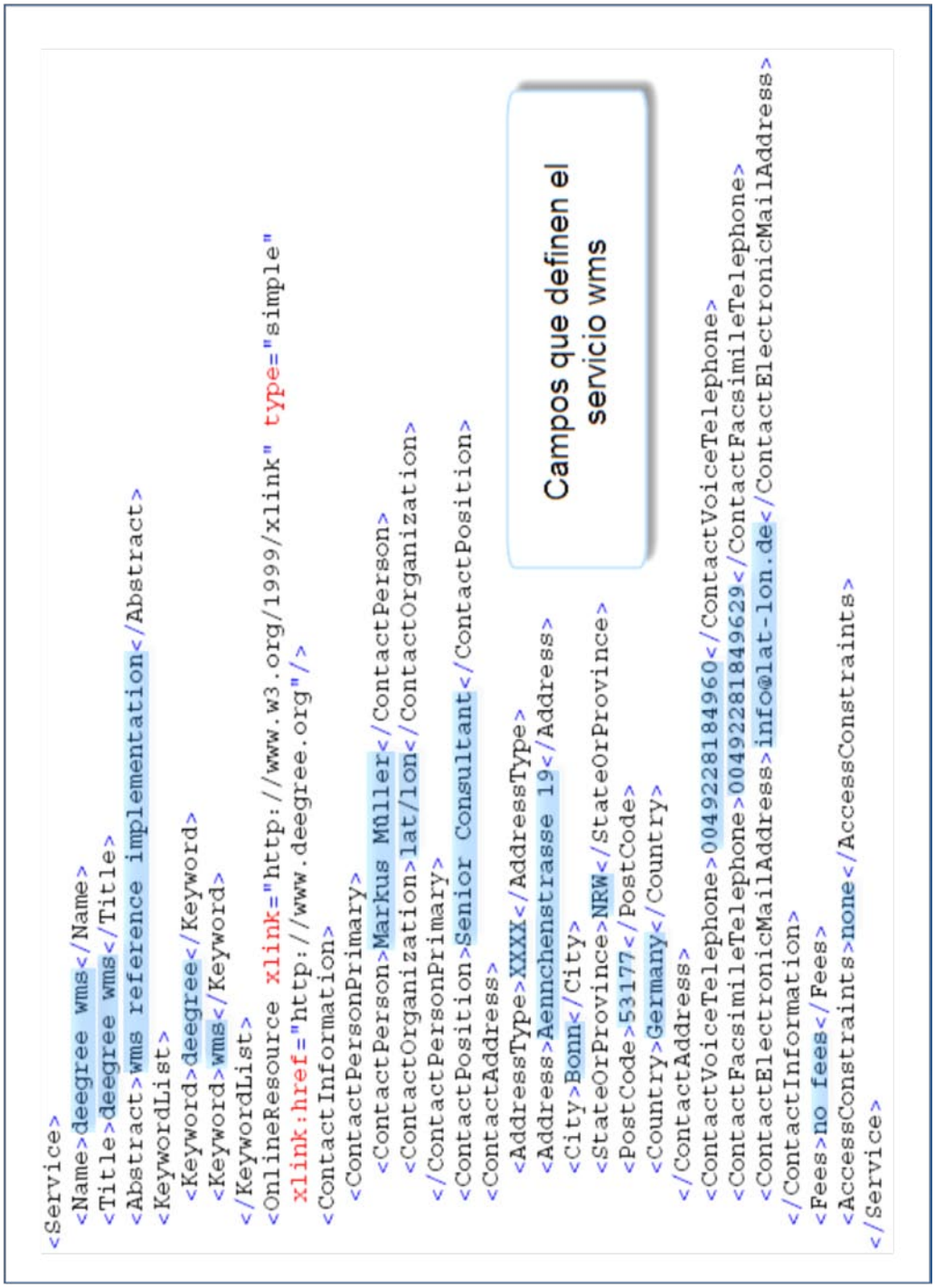


Figura 26. XML de administración de capas deegree.

```
<Layer xmlns:app="http://www.deegree.org/app" queryable="1" cascaded="0" opaque="1" noSubsets="1">
  <Name>Europe</Name>
  <Title>Datenlayer 1</Title>
  <Abstract>first testing layer</Abstract>
  <deegree:DataSource xmlns:deegree="http://www.deegree.org/wms"
      failOnException="1" queryable="1">
    <!-- default = equals the layer name and must be equal to the feature type
      or the layer name of the conected OWS -->
    <deegree:Name>app:Europe</deegree:Name>
    <!--name of the property that contains the geometries for this layer .
      this element will only be solved if the data source is a LOCALWFS or a
      REMOTEWFS, the value is case sensitive
      default = 'app:GEOM' -->
    <deegree:GeometryProperty>app:GEOM</deegree:GeometryProperty>
    <!--possible values: LOCALWFS, LOCALWCS, REMOTEWFS, REMOTEWCS,
    REMOTEWMS -->
    <!-- default = LOCALWFS -->
    <deegree:Type>LOCALWFS</deegree:Type>
    <!-- default is %Type%_capabilities.xml at the directory
    %RootDirectory%/WEB-INF/xml;
      default can only be used if data source type is LOCALXXX otherwise
      reference to the capabilities of the remote service must be set -->
    <deegree:OWSCapabilities>
      <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
          type="simple"
          xlink:href="http://www.lat-lon.de/documents/capabilities.xml"/>
    </deegree:OWSCapabilities>
    <deegree:GeometryProperty>app:the_geom</deegree:GeometryProperty>
    <!-- default is %Type%_capabilities.xml at the directory
    %RootDirectory%/WEB-INF/xml;
      default can only be used if data source type is LOCALXXX otherwise
      reference to the  capabilities of the remote service must be set -->
    <deegree:OWSCapabilities>
      <OnlineResource xmlns:xlink="http://www.w3.org/1999/xlink"
          type="simple"
      xlink:href="http://www.lat-lon.de/documents/another_capabilities_file.xml"/>
    </deegree:OWSCapabilities>
    <deegree:ScaleHint min="0" max="1000"/>
  </deegree:DataSource>
</Layer>
```

Campos que define una capa

4.3. MAPSERVER

MapServer es un ambiente de desarrollo de código abierto para construir aplicaciones de Internet habilitando características espaciales. [13] El logotipo de MapServer se muestra en la figura 27.

Figura 27. Logotipo de MapServer.

http://mapserver.gis.umn.edu/img/mapserver.GIF

No pretende ofrecer todas las funcionalidades de un SIG. Puede ejecutarse como un programa CGI o mediante Mapscript que soporta varios lenguajes de programación. MapServer fue desarrollado por la universidad de Minnesota, originalmente con el apoyo de la NASA, la cual necesitaba publicar sus imágenes satelitales.

MapScript se utiliza de forma independiente a MapServer CGI, este es un módulo que agrega capacidades de MapServer a un lenguaje de script. MapScript existe actualmente en PHP, Perl, Python, Ruby, Tcl, Java, C#, proporcionando una interfaz de scripting para la construcción de aplicaciones Web.

Más allá de la navegación de datos SIG, MapServer permite crear "imágenes de mapas geográficas", es decir, mapas de contenido que puedan dirigirse a los usuarios. Con el fin de visualizar, consultar y analizar información geográfica a través de la red. Por ejemplo, Minnesota DNR proporciona a los usuarios más de 10000 páginas Web, informes y mapas a través de una aplicación simple. La misma aplicación sirve como un "artefacto de mapa" para otras partes del sitio, proporcionando contexto espacial donde sea necesario.

MapServer maneja estándares del Open Geospatial Consortium (OGC), incluyendo Web Map Service (WMS) y Web Feature Service (WFS). Funciona con postgresql incluyendo la extensión postgis, soporta formatos SIG, como el formato ESRI shapefile.

Sus características principales, son:

- Se ejecuta bajo plataformas Linux, Windows, Mac OS X, Solaris.
- Soporta formatos vectoriales como ESRI shapefiles, PostGIS, ESRI ArcSDE y otros.
- Soporta formatos raster como JPG, PNG, GIF, TIFF, EPPL7 y otros.
- Manejo de etiquetas con detección de colisiones.

Requisitos Software. Para el correcto funcionamiento de MapServer se requiere de un Servidor HTTP como apache o Microsoft Internet Information Server, un lenguaje de scripting como PHP. Si desea utilizar orígenes de datos compatibles con bases de datos se deberá instalar postgresql con extensión postgis y un editor de texto.

Instalación. MapServer se encuentra disponible en la página oficial en la sección de descarga http://mapserver.gis.umn.edu/download/current, el proceso de instalación depende del sistema operativo, para usuarios Windows cuenta con el asistente MS4W (ver figura 28 y 29) que contiene una instalación pre configurada de Apache, PHP y MapServer, además incluye el módulo de soporte para WMS.

Figura 28. Opciones de instalación de MapServer en Windows.

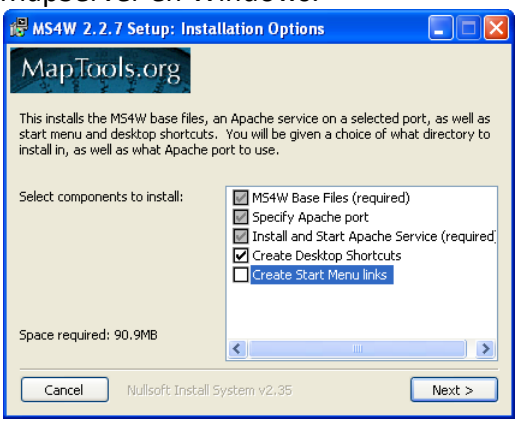
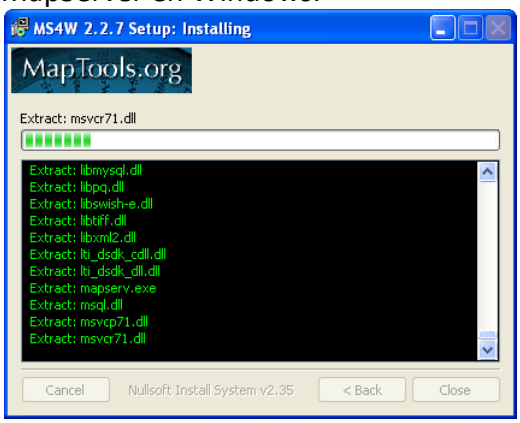


Figura 29. Progreso de la instalación de MapServer en Windows.



Si se desea utilizar el servidor en un ambiente ya existente la instalación debe hacerse manualmente, lo cual implica copiar ejecutables, carpetas y bibliotecas a rutas especificas así como la edición de archivos de configuración, para más información sobre el procedimiento, puede consultar las guías de instalación de MapServer en http://mapserver.gis.umn.edu/docs/howto/win32_compiling.

Para usuarios Linux la instalación es manual, aquí se debe instalar y configurar todos los prerrequisitos obligatorios y opcionales como gd, proj4, gdal y ogr, postgis, curl, libpng, WMS antes de compilar MapServer, mayor información sobre este procedimiento de instalación puede ser consultada en la guía de instalación para Linux disponible en http://mapserver.gis.umn.edu/docs/howto/compiling_on_unix.

Manejo de la Herramienta. El archivo mapfile es el corazón de MapServer. Define las relaciones entre los objetos, indica a MapServer donde se encuentran los datos y define

como se dibujan las capas. En la siguiente figura se presenta un ejemplo de un archivo mapfile para definir una capa WMS en MapServer.

Figura 30. Código de ejemplo para capas WMS en MapServer.

```
MAP
  IMAGETYPE      PNG24
  EXTENT         201621.4941 -294488.2853 14518.0222 498254.5115
  SIZE           400 300
  SHAPEPATH      "../data/"
  SYMBOLSET      "../symbols/symbols35.sym"

  PROJECTION
    Equal-Area
    "init=epsg:2163"
  END

  WEB
    IMAGEPATH "/ms4w/tmp/"
    IMAGEURL  "/tmp/"
    LOG "/ms4w/log/mslog"
  END

  LAYER NAME sample_jpg
    TYPE         RASTER
    OFFSITE      0 0 0
    STATUS       OFF
    CONNECTIONTYPE WMS
    CONNECTION "http://wms.jpl.nasa.gov/wms.cgi?"

    METADATA
      "wms_srs" "EPSG:4326"
      "wms_name" "modis"
      "wms_server_version" "1.1.1"
      "wms_format" "image/jpeg"
    END

    PROJECTION
      "init=epsg:4326"
    END
END
```

Para utilizar MapServer como servidor WMS se debe instalar un módulo de soporte adicional. Cada instancia de servidor WMS necesita tener su propio mapfile e incluir en él los metadatos del servicio a fin de producir una salida válida para las peticiones WMS.

Ventajas.

- Existen versiones para diferentes plataformas.
- Fue uno del los primeros software para la presentación de datos geográficos en Internet.
- Se distribuye con datos de ejemplo.

Desventajas.

- No implementa la última versión de WMS 1.3.0.
- No ofrece características que faciliten la creación de módulos de geocodificación.
- No presenta interfaz gráfica para la instalación, configuración y carga de datos.
- No soporta WMS de forma nativa.
- Perdida de rendimiento al ser una basada en CGI.

4.4. ARCIMS

ArcIMS es un servidor de mapas Web desarrollado por ESRI. Se trata de una aplicación que está diseñada para servir mapas a través de Internet. A veces estos mapas sólo son imágenes que permiten hacer operaciones de zoom y desplazamiento, mientras que otras son más complejas. [14] El logotipo de ArcIMS se muestra en la figura 31.

Figura 31. Logotipo de ArcIMS.

http://www.researchplanning.com/images/arcims.GIF

Ejemplos de mapas interactivos servidos con ArcIMS incluyen mapas con capas que se pueden activarse y desactivarse, o con atributos que pueden ser consultados.

Un visitante a un sitio manejado por ArcIMS no necesita nada más que un navegador Web, los sistemas de información geográfica y base de datos se mantienen en el lado del servidor.

ArcIMS es la solución para la entrega mapas dinámicos, datos SIG y servicios a través de la Web. Proporciona un marco de alta escalabilidad para publicaciones SIG. Los servicios ArcIMS pueden ser utilizados por una amplia gama de clientes, incluyendo aplicaciones Web personalizadas, aplicaciones de escritorio y dispositivos móviles inalámbricos.
El núcleo de ArcIMS es un servidor espacial donde la mayoría de los mapas relacionados con el servicio se procesan. Por el lado del servidor, ArcIMS conector se encuentra en la parte superior del servidor Web, los componentes ArcIMS y de aplicaciones del servidor funcionan en un segundo plano. En el lado del cliente, los visores pueden ser clientes ligeros, personalizables o aplicaciones de escritorio como ArcMap, ArcExplorer, o ArcPad.

Utiliza ArcXML para recibir y atender las solicitudes del cliente. Los datos detrás de ArcIMS normalmente se almacenan en formato Shapefile o en una base de datos ArcSDE.

DDE (del inglés, Data Delivery Extension) es una extensión para ArcIMS que entrega datos en diferentes formatos y sistemas de coordenadas con el fin de tener acceso a los datos en un formato compatible para las aplicaciones SIG.

Características de ArcIMS.

- Creación de aplicaciones fáciles de usar, para tareas que usen contenido geográfico.
- Desarrollo fácil de aplicaciones personalizadas en ambientes de desarrollo Web.
- Compartir datos con otros para lograr tareas.
- Implementación portales SIG.

Requisitos Software. Para ejecutar ArcIMS, se necesita un servidor Web y un contenedor de servlets. Otros requerimientos más específicos dependen de la plataforma y configuración del servidor Web, para más información puede obtenerse en http://support.esri.com/index.cfm?fa=knowledgebase.systemRequirements.matrix&pName=ArcIMS&productID=16&pvName=9.2&versionID=115&PID=16&PVID=351

Instalación. La instalación de ArcIMS puede generalizarse en cinco pasos.

1. Determinar el ambiente de configuración que coincida con los requerimientos del sistema de ArcIMS. (ver figura 32).

2. Establecer un sitio de configuración aceptable. Se pueden instalar los componentes de ArcIMS es una sola máquina, o se pueden decidir agregar diferentes componentes en máquinas separadas.

3. Distribuir los componentes de ArcIMS, así como instalar el Applicaton Server Conector, también incluir Web Server, Servlet Engine (si es necesario).

4. Configurar el Web Server y el Servlet Engine para comunicarse con ArcIMS. (ver figura 33).

5. Colocar los parámetros de ambiente de configuración de ArcIMS. Para la plataforma Windows, incrementar la memoria para JAVA.

Figura 32. Selección de componentes en la instalación de ArcIMS.

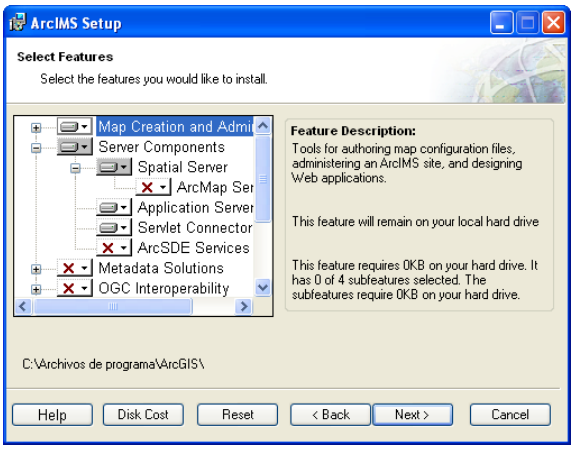
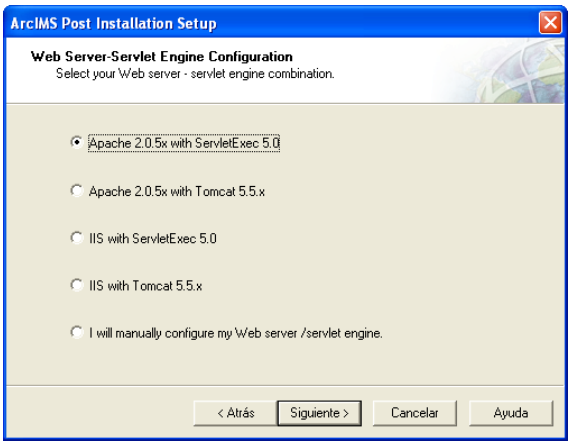
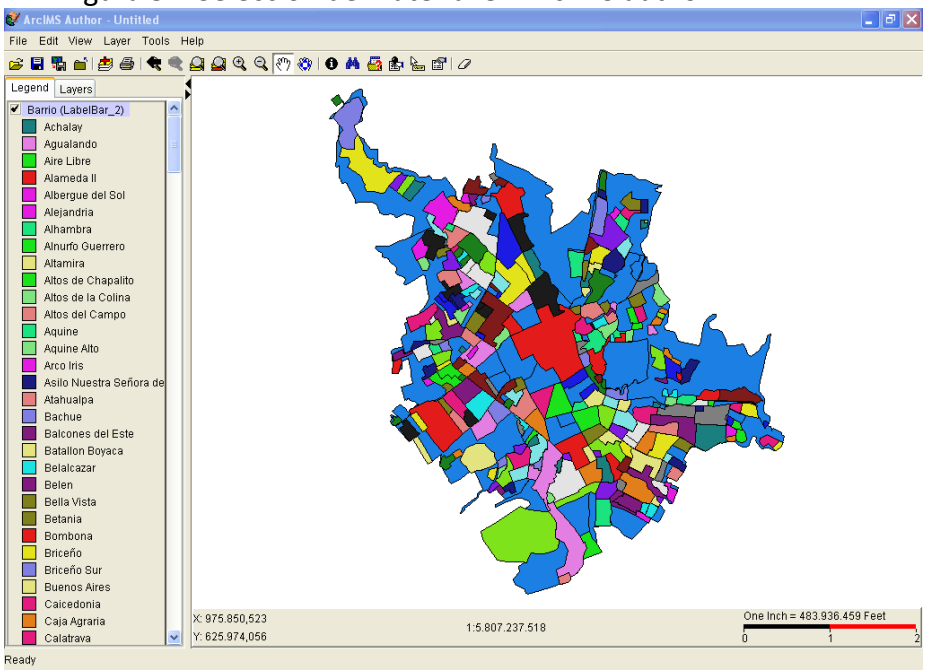


Figura 33. Opciones de post instalación de ArcIMS.



Manejo de la herramienta. Una vez instalados todos los componentes en las ubicaciones designadas puede procederse a publicar información, la principal herramienta para ello es el ArcIMS author que permite seleccionar la capas para un proyecto así como establecer la forma en que estas serán presentadas, como se muestra en la figura 34. Una vez establecidas las opciones deseadas con el author se procede a instalarlas en el servidor, con ello el servicio queda listo y disponible para los clientes. ArcIMS también incorpora herramientas que permiten la creación de sitios Web que sirven como clientes del servicio.

Figura 34. Selección de material en ArcIMS author.



Ventajas.

- Maneja Interfaces gráficas para administrar el servicio.
- Maneja herramientas para el realizar el proceso de geocodificación.
- Está fuertemente integrado con otros productos de la suite.

Desventajas.

- Solo está disponible para la plataforma Windows.
- El costo en licenciamiento es elevado.
- No es software libre.

4.5. OPENLAYERS

OpenLayers es una biblioteca Java script orientada a objetos de código abierto para visualizar mapas dinámicos en navegadores Web. Ofrece un API para la creación de aplicaciones geográficas y se distribuye bajo licencia BSD. [15] El logotipo de OpenLayers se muestra en la figura 35.

Figura 35. Logotipo de OpenLayers.

http://www.openlayers.org/images/OpenLayers.trac.PNG

Fue creado por MetaCarta y liberado como software de código abierto por MetaCarta Labs quien mantiene un papel activo en el soporte y la administración del proyecto.

Como un marco, OpenLayers se destina a separar las herramientas de mapa de los datos geográficos a fin de que todas las herramientas puedan operar sobre todas las fuentes de datos, además puede mostrar marcadores cargados desde diferentes fuentes. Esta biblioteca se encuentra aún en fase de desarrollo, por lo que esperan muchos cambios.

Tiene la cualidad de acceder a información geoespacial ya sea de servidores que cumplan con el estándar WMS y WFS del OpenGeospatial Consortium (OGC) o de los principales servicios comerciales que hay en la actualidad como son Google Maps, Virtual Earth, TerraServer y WorldWind. OpenLayers es una opción de código abierto similar a la API de Google Maps, pero con la libertad de acceder a más fuentes de datos.

Requisitos Software. Al ser una biblioteca desarrollada totalmente en javascript necesita de un navegador Web el cual debe habilitar el uso de javascript y una fuente de datos compatible.

Instalación. OpenLayers se encuentra disponible en http://openlayers.org/download el proceso de instalación se reduce a descargar el archivo y descomprimirlo en una ruta al alcance de la página Web. Dentro debe incluirse una etiqueta javascript dirigida a la biblioteca.

Adicionalmente, se puede usar la biblioteca incluyendo el script con la ruta http://www.openlayers.org/api/OpenLayers.js en la página, para siempre tener la versión más reciente.

Manejo de la Herramienta. El API de OpenLayers tiene dos conceptos que son importantes para construir un mapa, mapa y capa. El mapa almacena la información sobre la proyección por defecto, medidas, unidades, entre otras. Dentro de un mapa, los datos se visualizan a través de capas. Una capa es una fuente de datos que define como deben solicitarse los datos para ser mostrados.

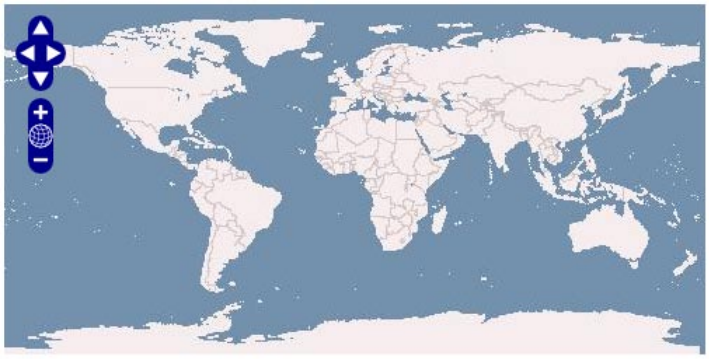
Construir un visor OpenLayers requiere construir el código HTML en el cual el visor será presentado. OpenLayers puede ubicar un mapa dentro de cualquier elemento en cualquier nivel, esto significa que se puede ubicar un mapa en casi cualquier elemento HTML de la página.

Con el fin de crear el visor, se debe crear un objeto mapa. El constructor OpenLayers.Map requiere un argumento que debe ser el identificador del elemento HTML en donde se ubicará el mapa, el siguiente paso es agregar una capa. Finalmente para mostrar el mapa se debe asignar el centro y el nivel zoom. EL código fuente de ejemplo para crear un visor simple con OpenLayers se muestra en la figura 36, la apariencia del visor en un navegador Web se presenta en la figura 37.

Figura 36. Código de un visor simple de OpenLayers.

```
<html>
  <head>
    <title>OpenLayers Example</title>
    <script src="http://openlayers.org/api/OpenLayers.js"></script>
  </head>
  <body>
      <div style="width:100%; height:100%" id="map"></div>
      <script defer="defer" type="text/javascript">
        var map = new OpenLayers.Map('map');
        var wms = new OpenLayers.Layer.WMS( "OpenLayers WMS",
          "http://labs.metacarta.com/wms/vmap0", {layers: 'basic'});
      map.addLayer(wms);
      map.zoomToMaxExtent();
      </script>
  </body>
</html>
```

Figura 37. Apariencia de un visor simple de OpenLayers.



Ventajas.

- Soporta varios formatos de entrada.
- Es una biblioteca construida totalmente en javascript.
- Al ejecutarse el lado del cliente, resulta Independiente de la tecnología del lado del servidor.

Desventajas.

- En ocasiones genera una excesiva demanda sobre el servidor al solicitar imágenes muy distantes respecto al área visible actual y que probablemente no serán necesarias.
- Se encuentra en estado inicial de desarrollo.

4.6. GOOGLE MAPS

Es un servicio gratuito para aplicaciones de cartografía Web proporcionado por Google que potencia muchos servicios basados en mapas, como el sitio Web de Google Maps, Google Ride Finder y páginas de terceros que utilizan mapas a través del API de Google Maps. [16] El logotipo de GoogleMaps se muestra en la figura 38.

Figura 38. Logotipo de GoogleMaps.

http://code.google.com/apis/maps/logo.GIF

GoogleMaps ofrece la posibilidad de que cualquier propietario de una página Web integre muchas de las características del API, además ofrece mapas de calles, planes de rutas, localizadores de puntos urbanos, imágenes de mapas desplazables, así como fotos satelitales del mundo entero.

Está disponible para diversos navegadores Web como Internet Explorer, Mozilla Firefox, Opera, Safari, entre otros, para lograr la conectividad sin sincronía con el servidor, Google aplicó el uso de AJAX dentro de la aplicación.

El API de Google Maps facilita la integración en sitios Webs con datos propios. Proporciona una serie de utilidades para la manipulación de mapas y contenidos a través de una variedad de servicios, lo que le permite crear aplicaciones robustas de mapas en su sitio Web. Como la mayoría de aplicaciones Web de Google, usa un gran número de archivos javascript, permite el uso de marcadores como un indicador en forma de pin, el cual es una imagen transparente sobre el mapa.

Requisitos Software. Al ser una biblioteca desarrollada en javascript y XML necesita de un navegador Web el cual debe habilitar el uso de javascript y el servidor de mapas de Google.

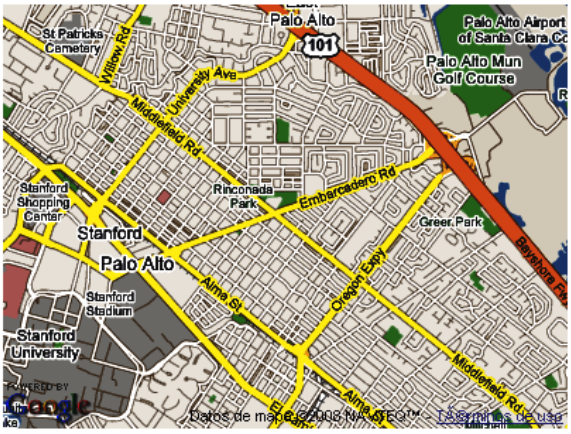
Instalación. Google Maps es un API AJAX que puede usarse en las páginas Web, para usarlo solo se necesita incluir una simple etiqueta <script> en la parte superior de la página Web con la clave del API para Google. Para más información sobre la clave de visite http://code.google.com/apis/ajaxsearch/signup.html.

Manejo de la Herramienta. Una vez que el API de Google AJAX está cargado, se especifica los módulos que se desea utilizar en la página con el comando google.load. Después de llamar a este comando puede utilizar todos los módulos cargados en la página Web. Luego se debe crear un objeto mapa. El constructor requiere un argumento, este argumento debe ser un elemento HTML. Después debe asignarse el centro de mapa con el objeto google.maps.LatLng. Para finalizar se usa google.setOnLoadCallback para registrar la función en el manejador que se ejecutará cuando el documento cargue. El código fuente de un ejemplo básico para utilizar el API de GoogleMaps se muestra en la figura 39, la apariencia del visor en un navegador Web se presenta en la figura 40.

Figura 39. Un ejemplo básico del manejo del API de GoogleMaps.

```
<html>
  <head>
    <script src="http://www.google.com/jsapi?key=ABCDEFG"></script>
    <script type="text/javascript">
      google.load("maps", "2");

      function initialize() {
         var ele = document.getElementById("map");
         var map = new google.maps.Map2(ele);
         map.setCenter(new google.maps.LatLng(37.4419, -122.1419), 13);
      }
      google.setOnLoadCallback(initialize);
    </script>
  </head>
  <body>
    <div id="map" style="width: 500px; height: 400px"></div>
  </body>
</html>
```

Figura 40. Mapa de calles de GoogleMaps.



Más información se encuentra disponible en la página oficial, en la sección de documentación http://code.google.com/apis/ajax/documentation/#GettingStarted.

Ventajas.

- Biblioteca construida totalmente en javascript.
- Permite su uso en combinación con diferentes tecnologías del lado del servidor.
- Soporta tecnologías Web 2.0.

Desventajas.

- Solo soporta servidores de mapas google, que se administran de forma privada.
- El uso de la biblioteca es gratuito, sin embargo su código fuente no es libre.
- El uso del servicio y de la biblioteca están sujetos a diversas restricciones, mayor información se puede obtener en http://code.google.com/apis/maps/terms.html

4.7. GOOGLE MOBILE

Google para móviles es una aplicación descargable que permite ver mapas e imágenes de satélite, encontrar puntos de referencia y obtener instrucciones de cómo llegar a un lugar en dispositivos de recursos restringidos como celulares y PDAs. Incluye compatibilidad con GPS para los modelos que disponen de esta función. [17] El logotipo de GoogleMobile se muestra en la figura 41.

Figura 41. Logotipo de GoogleMobile.

http://www.google.com/mobile/gmm/logo.GIF

Requisitos. Google Maps ofrece servicio para los siguientes dispositivos móviles:

- La mayoría de teléfonos habilitados para Java ME.
- Dispositivos Palm con Palm OS 5 y posterior.
- Todos los dispositivos BlackBerry con pantalla de color.
- Dispositivos Windows Mobile con Windows Mobile 2003 Second Edition, 5.0 y superior.

Instalación. Para celulares se debe visitar la página www.google.com/gmm desde el navegador Web del móvil para descargar Google Maps. Para PDAs, la versión móvil de Google Maps es una aplicación Java desarrollada por Google, la instalación depende del

sistema operativo, para usuarios Windows mobile descargue el archivo CAB en http://www.google.com/gmm/GoogleMaps.CAB y realice ActiveSync. Para usuarios del Palm OS descargue el archivo PRC en http://www.google.com/gmm/GoogleMaps.prc y realice HotSync.

Manejo de la Herramienta en teléfonos celulares.

Para manejar la herramienta GoogleMobile en teléfonos celulares se deben seguir los pasos que se listan a continuación (ver figura 42).

Los botones y del teléfono se usan para abrir y cerrar el menú de la aplicación.

Las teclas de flecha del teclado o bien se usan para desplazarse por los elementos del menú y mover el mapa.

Se pulsa intro en el teclado o bien se hace clic en para seleccionar elementos del menú, direcciones de salida/llegada y para acercar el zoom.

Se puede navegar por los elementos del menú, los resultados de búsqueda y las rutas con las teclas 1-9 del teclado o del teléfono.

Figura 42. Teléfono celular con Google Mobile.

http://www.google.com/gmm/images/phone_en.PNG

Manejo de la herramienta para PDAs.

Para manejar la herramienta GoogleMobile en PDAs se deben seguir los pasos que se listan a continuación (ver figura 43).

Se usa el apuntador para desplazarse por el menú de la aplicación o para tocar los botones de función del mapa.

Un toque en el botón  aleja la visualización del mapa.

Un toque en el botón  acercar la visualización del mapa.

Figura 43. PDA con Google Mobile.

http://www.google.com/gmm/images/phone_en.PNG

Ventajas.

- Soporta una completa gama de dispositivos.
- Soporta varios sistemas operativos como Palm OS Garnet y Windows Mobile.

Desventajas.

- Solo soporta servidores de mapas google, que se administran de forma privada.
- El uso de la aplicación es gratuito, sin embargo su código fuente no es libre y está sujeto a diversas restricciones.

- Se trata de una aplicación cerrada, es decir no permite la adición de características distintas a las que ya tiene, por tanto, es inapropiada en el campo de las aplicaciones a medida.

4.8. GOOGLE GEOCODER

El API de Google Maps incluye un servicio de geocodificación que puede accederse directamente a través de una petición HTTP o usando un objeto GClientGeocoder. Cabe señalar que la tarea de geocodificación consume tiempo y recursos, por lo que es aconsejable realizar el proceso de Geocodificación y almacenar los resultados usando algún sistema de caché. [18] El logotipo de GoogleGeocoder se muestra en la figura 44.

Figura 44. Logotipo de GoogleGeocoder.

http://code.google.com/apis/maps/documentation/services.GIF

Requisitos Software. Al ser una biblioteca desarrollada en javascript y XML necesita de un navegador Web el cual debe habilitar el uso de javascript y el servidor de mapas de Google.

Instalación. Google Maps es un API AJAX que puede usarse en las páginas Web, para usarlo solo se necesita incluir una simple etiqueta <script> en la parte superior de la página Web con la clave del API para Google. Más información sobre la clave se encuentra en http://code.google.com/apis/ajaxsearch/signup.html.

Manejo de la herramienta. Para acceder al geocoder directamente se puede enviar una petición a http://maps.google.com/maps/geo? con los siguientes parámetros:

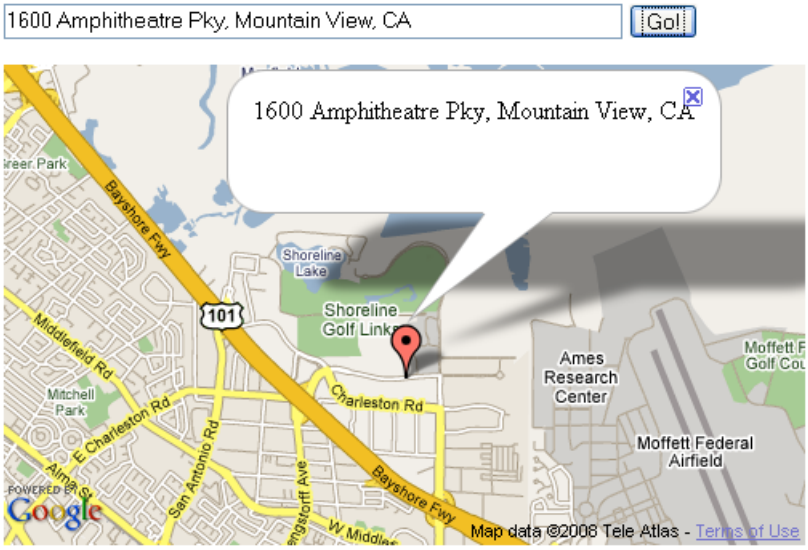
q	La dirección que se quiere geocodificar.
key	La clave de acceso al API.
Output	El formato de salida, las opciones son XML, KML, CSV, o JSON.

La respuesta a esta petición en formato XML (ver figura 45), debe ser interpretada por el cliente google map que obtiene los valores de longitud y latitud y puede crear un marcador del punto obtenido. (ver figura 46)

Figura 45. Respuesta XML a una petición del geocoder del google.

```
<kml xmlns="http://earth.google.com/kml/2.0">
  <Response>
    <name>1600 amphitheatre mountain view ca</name>
    <Status>
      <code>200</code>
      <request>geocode</request>
    </Status>
    <Placemark>
      <address>
        1600 Amphitheatre Pkwy, Mountain View, CA 94043, USA
      </address>
      <AddressDetails Accuracy="8">
        <Country>
          <CountryNameCode>US</CountryNameCode>
          <AdministrativeArea>
            <AdministrativeAreaName>CA</AdministrativeAreaName>
           <SubAdministrativeArea>
             <SubAdministrativeAreaName>Santa Clara</SubAdministrativeAreaName>
             <Locality>
               <LocalityName>Mountain View</LocalityName>
               <Thoroughfare>
                 <ThoroughfareName>1600 Amphitheatre Pkwy</ThoroughfareName>
               </Thoroughfare>
             </Locality>
       </Country>
     </AddressDetails>
     <Point>
       <coordinates>-122.083739,37.423021,0</coordinates>
     </Point>
   </Placemark>
  </Response>
</kml>
```

Figura 46. Cliente GoogleMaps graficando una respuesta del geocoder del google.



Ventajas.

- Biblioteca construida totalmente en javascript.
- Soporta tecnologías Web 2.0.
- Puede trabajar en conjunto con otros servicios de google.

Desventajas.

- Solo soporta servidores de geocodificación google, que se administran de forma privada.
- El uso de la aplicación es gratuito, sin embargo su código fuente no es libre y está sujeto a diversas restricciones.

5. SOFTWARE UTILIZADO EN LA CONSTRUCCIÓN DE ATLAS.

En este capítulo se realiza una descripción del software usado en el desarrollo de la herramienta Atlas, se encuentran clasificados en 3 categorías, herramientas software, tecnologías Java y bibliotecas.

5.1. HERRAMIENTAS DE SOFTWARE

Esta categoría describe el software utilizado en la construcción de la herramienta Atlas, así como también el software para la construcción de los datos de prueba y datos reales.

5.1.1. Netbeans IDE. Un entorno de desarrollo integrado (en inglés Integrated Development Environment, IDE) es un programa compuesto por un conjunto de herramientas que facilitan la tarea de programación. El entorno de desarrollo elegido para la implementación de Atlas es Netbeans IDE [19], el logotipo de Netbeans se muestra en la figura 47.

Figura 47. Logotipo de Netbeans.

http://tec.fresqui.com/files/images/netbeans-logo.GIF

Netbeans es una plataforma para el desarrollo de aplicaciones utilizando Java, que permite editar programas, compilarlos, ejecutarlos, depurarlos y construir interfaces gráficas entre otras. Además es libre, de código abierto, distribuido bajo la licencia CDDL, ideal para crear aplicaciones de escritorio y aplicaciones para dispositivos móviles.

Netbeans plugins permite ampliar la funcionalidad del IDE, así como cualquier otra aplicación construida sobre la plataforma. Al instalar un plugin se obtiene el soporte para las nuevas tecnologías, marcos y especificaciones, por ejemplo el plugin de UML agrega funcionalidad para la construcción de diagramas.

5.1.2. Eclipse IDE. Eclipse es un entorno de desarrollo integrado para desarrolladores Java, de código abierto, independiente de plataforma, distribuido bajo su propia licencia pública de eclipse (en inglés, Eclipse Public License) [20], el logotipo de Eclipse se muestra en la figura 48.

Figura 48. Logotipo de eclipse.

http://dec.gesbit.com/archives/images/logotipo-eclipse.PNG

Eclipse es desarrollado por la fundación eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios, dispone de características como edición de texto, resaltado de sintaxis, compilación en tiempo real, pruebas unitarias con JUnit, control de versiones con CVS y asistentes para creación de proyectos, clases, pruebas, etc.

Eclipse puede extender sus capacidades por medio de plungins compatibles con su marco, por ejemplo, Aptana es un plugin que agrega un conjunto de funciones para desarrollar aplicaciones Web, incluyendo asistentes de código para JavaScript, HTML y CSS.

5.1.3. Quantum GIS. Es una herramienta libre, de código abierto para sistemas de información geográfica (SIG) que puede ejecutarse en sistemas operativos como Linux, Unix, Mac y Windows, soporta formatos vectoriales, raster y base de datos, es distribuido bajo licencia GNU [21], el logotipo de Quantum GIS se muestra en la figura 49.

Figura 49. Logotipo de Quantum GIS.

http://www.freemacware.com/wp-content/images/qgis.PNG

QuantumGIS tiene un pequeño tamaño en archivos en comparación con software SIG comerciales y requiere menos poder de procesamiento y memoria RAM, por lo que puede ser usado con hardware limitado o simultáneamente con otras aplicaciones.

QuantumGIS permite navegar y crear mapas, soporta muchos formatos comunes para datos espaciales vectoriales como ShapeFile, ArcInfo coverages, Mapinfo, GRASS, entre otros y raster como GRASS GIS, GeoTIFF, TIFF, JPG, etc. soporta extensiones espaciales para PostgreSQL, PostGIS. Finalmente soporta plugins para agregar nuevas funcionalidades al software.

5.1.4. CAD2Shape. Es una herramienta de software distribuida mediante shareware, utilizada para convertir archivos de autocad a shape files de ESRI [22]. Sus características principales son:

- Funciona en Windows 2000/2003/ XP/Vista.
- Soporta archivos provenientes de AutoCAD versión 2008.
- Conversión de texto a poli líneas.
- Correcta transformación de agujeros en polígonos.
- Filtrado por capa, área, color o entidad.
- Traducción de datos de entidad extendidos (Xdata).
- Traducción de atributos.

5.2. TECNOLOGÍAS JAVA

El lenguaje de programación para la implementación de la herramienta Atlas es Java, el cuál es en sí una plataforma independiente tanto de hardware como de software gracias al soporte de la Máquina Virtual de Java (en ingles Java Virtual Machine, JVM) [23], el logotipo de Java se muestra en la figura 50.

Figura 50. Logotipo de Java.

http://www.mundoatletismo.com/site/images/Java.jpg

Cualquier aplicación que se desarrolle se apoya en un gran número de clases preexistentes del API de Java en sus diversas especificaciones.

Otras de ellas son desarrolladas por la comunidad o medida por el propio usuario, formando un gran grupo de clases extensibles para cada aplicación.

Java incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores como manejo de hilos, ejecución remota, componentes, seguridad, acceso a bases de datos, entre otros.

Por eso es un lenguaje ideal para la informática moderna, porque incorpora todos estos conceptos de un modo estándar, mucho más sencillo y claro que con las citadas extensiones de otros lenguajes. Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

La compañía Sun describe el lenguaje Java como “simple, orientado a objetos, distribuido, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico”.

5.2.1. Principales características de Java.

- **Lenguaje Orientado a Objetos.** Es una característica en la que el software se diseña de tal forma, que los distintos tipos de datos que usa están organizados de una forma natural lógica.

Una clase es una agrupación de datos y funciones que operan sobre esos datos. A estos datos y funciones pertenecientes a una clase se les denomina variables y métodos.

Los objetos son entidades de clase que combinan estado, comportamiento e identidad. El estado está compuesto de datos, serán uno o varios atributos a los que se habrán asignado unos valores concretos. El comportamiento está definido por los procedimientos o métodos con que puede operar dicho objeto, es decir, qué operaciones se pueden realizar con él. La identidad es una propiedad de un objeto que lo diferencia del resto, dicho con otras palabras, es su identificador.

Entre las características más importantes de los objetos se encuentran las siguientes.

Abstracción. Cada objeto en el sistema sirve como modelo de un agente abstracto que puede realizar trabajo, informar y cambiar su estado, y comunicarse con otros objetos en el sistema sin revelar cómo se implementan estas características. Los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

Encapsulamiento. Significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema.

Principio de ocultación. Cada objeto está aislado del exterior, es un módulo natural, y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas.

Polimorfismo. Comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando.

Herencia. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de clases preexistentes.

La programación orientada a objetos se basa en la programación de clases. Un programa se construye a partir de un conjunto de clases. Esto hace posible que

grandes proyectos sean fáciles de gestionar y manejar, mejorando como consecuencia su calidad y reduciendo el número de fallas.

- **Independencia de plataforma.** Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, es importante conseguir una herramienta independiente del tipo de hardware utilizado. El desarrollo de un código "neutro" que no depende del tipo de CPU utilizada permitió lograr lo que luego se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere".

  De esta manera se logra obtener un código capaz de ejecutarse en cualquier tipo de máquina, el cual una vez compilado no necesita de ninguna modificación por el hecho de cambiar de hardware o de ejecutarlo en otra máquina. La clave consiste en desarrollar un código preparado para ser ejecutado sobre una Máquina Virtual de Java (JVM).

  Es esta JVM quien interpreta este código neutro convirtiéndolo a código particular de la CPU, evitando así el tener que realizar un programa diferente para cada plataforma. La JVM es el intérprete de Java. Ejecuta los bytecodes ficheros compilados con extensión *.class creados por el compilador de Java. Tiene numerosas opciones entre las que destaca la posibilidad de utilizar el denominado JIT (Just In Time Compiler) que interpreta el bytecodes generado convirtiéndolo a instrucciones máquina del código nativo.

- **El recolector de basura.** Uno de los aspectos más importantes en la programación orientada a objetos es la forma en la cual son creados y eliminados los objetos. La eliminación de los objetos es realiza por el "garbage collector", quien automáticamente libera la memoria ocupada por un objeto cuando no existe ninguna referencia apuntando a él. El programador determina cuándo se crean los objetos y el entorno en tiempo de ejecución de Java es el responsable de gestionar el ciclo de vida de los objetos.

5.2.2. Java SE. Java SE (en ingles Java Platform, Standard Edition) es una plataforma ampliamente utilizada para la programación en el lenguaje Java. Es la plataforma Java utilizada para desplegar aplicaciones portables de uso general. [24]

En términos prácticos, Java SE consiste en una máquina virtual, que debe ser utilizada para ejecutar programas Java, un conjunto de bibliotecas o paquetes necesarios para permitir el uso de sistemas de archivos, redes, interfaces gráficas, etc.

A continuación se presentan descripciones de algunos de los paquetes principales de Java SE que hacen parte de Java Foundation Classes, utilizados en la construcción de Atlas.

El paquete **Java.lang** contiene clases fundamentales e interfaces fuertemente relacionadas con el lenguaje y el sistema de ejecución. Esto incluye las clases raíz que forman la jerarquía de clases, tipos relacionados con la definición del lenguaje, excepciones básicas, funciones matemáticas, hilos, funciones de seguridad, así como también alguna información sobre el sistema nativo subyacente. Las clases principales del paquete Java.lang se muestran en la tabla 8.

Tabla 8. Clases generales de Java SE paquete Java.lang.

Object	Clase raíz de toda la jerarquía de clases.
Enum	Clase base para las clases enumeration.
Class	Clase raíz del sistema de reflexión Java.
Throwable	Clase base de la jerarquía de clases de excepciones.
Error **Exception** **RuntimeException**	Clases base de cada tipo de excepción.
Thread	Clase que permite operaciones con hilos.
String	Clase para cadenas String y literales.
StringBuffer	Clase para realizar manipulación de Strings.
Comparable	Interfaz que permite comparación genérica y ordenamiento.
Iterable	Interfaz que permite iteración genérica usando el bucle for mejorado.
ClassLoader **System**	Suministran operaciones del sistema que gestionan el enlazado dinámico de clases, creación de procesos externos, investigaciones del entorno y refuerzo de políticas de seguridad.
Math	Suministra funciones matemáticas básicas.

El paquete **Java.io** contiene clases que soportan entrada/salida. Las clases del paquete son principalmente streams; sin embargo, se incluye una clase para ficheros de acceso aleatorio. Las clases principales del paquete Java.io se muestran en la tabla 9.

Tabla 9. Clases generales de Java SE paquete Java.io.

InputStream **OutputStream**	Clases abstractas base para leer y escribir flujos de bytes.

ByteArrayInputStream **ByteArrayOutputStream**	Clases que contienen buffers de bytes que pueden leerse y escribirse de un flujo.
FileInputStream **FileOutputStream**	Clases que obtienen flujos de lectura y escritura para un archivo.
BufferedInputStream **BufferedOutputStream**	Clases que añade funcionalidad de buffers a los flujos.

El paquete **Java.net** suministra rutinas especiales IO para redes, permitiendo las peticiones HTTP, así como también otras transacciones comunes. Algunas de clases del paquete Java.net se muestran en la tabla 10.

Tabla 10. Clases generales de Java SE paquete Java.net.

HttpURLConnection	Clase de conexión con soporte para el protocolo HTTP.
URL	Clase que representa un localizador uniforme a un recurso en Web.
URLClassLoader	Clase para cargar recursos o nuevas clases desde una URL.
URLDecoder **URLEncoder**	Clases de utilidad para codificar y decodificar una URL.

El paquete **Java.util** suministra principalmente estructuras de datos. En el paquete está incluida la API Collections, una jerarquía organizada de estructura de datos. Algunas de las interfaces del paquete Java.util se muestran en la tabla 11.

Tabla 11. Interfaces generales de Java SE paquete Java.util.

List<E>	Una colección ordena.
Iterator<E>	Un iterador sobre una colección
Map<K,V>	Un objeto de correspondencia clave valor.
Set<E>	Una colección que no contiene elementos duplicados.

El paquete **javax.swing** es una colección de rutinas que se construyen sobre Java.awt para suministrar controles independientes de plataforma. Usa las rutinas de dibujado 2D para renderizar los componentes de interfaz compatibles con pluggable looks and feels. Algunas de las clases del paquete javax.swing se muestran en la tabla 12.

Tabla 12. Clases generales de Java SE paquete javax.swing.

JComponent	Clase base para todos los componentes swing.
JFrame	Clase contenedora de alto nivel que proporciona ventanas.
Jpanel	Clase contenedora de propósito general para componentes ligeros, utiliza un controlador de distribución.
JTabbedPane	Clase contenedora que permite cambiar entre un grupo de componentes a otros en el mismo espacio de trabajo.
JTree	Clase que muestra un conjunto de datos jerárquicos en un árbol, no contiene datos, simplemente es un vista de ellos.
JScrollPanne	Clase que proporciona una vista desplazable de un componente. Para componentes con tamaño dinámico.
JSplitPane	Clase que contiene dos componentes, separados por un divisor, para especificar qué cantidad de área pertenece a cada componente.
JTable	Clase para representar tabla, no contiene ni almacena datos, simplemente es una vista de los datos.
JFileChooser	Clase que proporciona un diálogo opcionalmente modal para elegir un archivo de una lista.
JtextArea	Clase que representa un área de texto que muestra múltiples líneas, permitiendo edición.
JradioButton	Clase que representa grupos de botones de selección única.
JcomboBox	Clase contenedora que permite elegir un elemento desde una lista.
JSpinner	Clase que permite seleccionar un valor de una secuencia configurable.

5.2.3. Java ME. Java ME (en ingles Java Platform, Micro Edition) es una especificación de un subconjunto de la plataforma Java, que provee una colección de APIs para el desarrollo de software para dispositivos con restricciones de tamaño y recursos, como lo son los teléfonos celulares y asistentes personales digitales (en inglés PDAs) entre otros. Java ME se ha convertido en una opción para la creación de aplicaciones para dispositivos móviles, ya que estas pueden ser emuladas en un computador durante la fase de desarrollo. [25]

Java ME se divide en configuraciones, perfiles y APIs opcionales. Una configuración define un tipo de dispositivo en función de las características de su hardware, sus limitaciones, sus capacidades, etc y le asigna una máquina virtual y un conjunto de APIs adecuados a ese hardware.

La configuración para dispositivos con conexión limitada (en inglés Connected Limited Device Configuration, CLDC) es una configuración limitada que está presente en la mayoría de los dispositivos móviles.

Configuración para Dispositivos con Conexión Limitada. Contiene un subconjunto estricto de bibliotecas Java, y es el mínimo necesario para que la máquina virtual Java funcione. Básicamente CLDC define configuraciones para una gran variedad de dispositivos.

Una configuración provee el conjunto más básico de bibliotecas y características de máquina virtual que deben estar presentes en cada implementación. En conjunto con un o más perfiles, la configuración de CLDC da a los desarrolladores una plataforma solida para la creación de aplicaciones.

Dentro de una configuración, un perfil define ciertas características concretas, como la interfaz de usuario. Existen tres perfiles para la configuración CLDC **MIDP** Perfil para dispositivos de información móvil (en ingles Mobile Information Device Profile), que es la usada en los teléfonos móviles, **IMP** (Information Module Profile) que es una versión de la anterior sin interfaz de usuario, y **DoJa**, destinado a un tipo de móviles japoneses.

Perfil para Dispositivos de Información Móvil. Es un perfil que se apoya en CLDC y que proporciona los paquetes y clases necesarios para el desarrollo de aplicaciones. MIDP está orientado principalmente a teléfonos móviles, Sin embargo, existen implementaciones para PalmOS (versión 3.5 y superiores) y PocketPC, por lo que es también utilizable en casi cualquier PDA.

Se denomina **Midlet** a las aplicaciones Java ME realizadas usando la especificación de MIDP, es decir, un midlet será una aplicación que puede usar la funcionalidad aportada por MIDP y por CLDC. Un midlet siempre estará compuesto, al menos, por una clase principal que hereda directamente de la clase javax.microedition.midlet.MIDlet.

El API de MIDP 2.0 se compone de las clases y paquetes heredados de CLDC y de otra serie de paquetes situados en la jerarquía javax.microedition entre los cuales se destacan.

Paquete de Ciclo de vida de las Aplicaciones (javax.microedition.midlet): Este paquete permite a las aplicaciones MIDP (midlets) interactuar con el entorno, sobre el cual la aplicación se está ejecutando.

Paquete de Interfaz (javax.microedition.lcdui): Este es la parte del API dedicada al interfaz de usuario. Proporciona un conjunto de características para la implementación de interfaces en MIDP.

Paquete de Red (javax.microedition.io): MIDP proporciona soporte de red basándose en CLDC.

Paquete de Sonido: (javax.microedition.media): El API Media de MIDP 2.0 es un bloque directamente compatible con la especificación Mobile Media API (MMAPI). MMAPI extiende la funcionalidad de J2ME proporcionando audio, video y otras características multimedia.

Paquetes principales: Java.lang y Java.util que contienen clases del lenguaje y utilidades incluidas en el perfil, provenientes de J2SE.

Interfaz de Programación de Aplicaciones. Finalmente, cada dispositivo móvil puede incluir soporte para distintas APIs opcionales. Por ejemplo, un dispositivo con Bluetooth puede disponer de la Bluetooth API (JSR 82), como estas capacidades varían de dispositivo en dispositivo, hay que acudir a las especificaciones de cada uno para ver qué APIs soportan.

Esto implica que cada fabricante realice las implementaciones de las APIs Java ME en sus dispositivos móviles. Aunque para ello se basan en la implementación de referencia de Sun, en la práctica se pueden encontrar diferencias significativas en el comportamiento de distintos dispositivos para una misma función, fallando en unos dispositivos lo que funciona en otros. En ocasiones, esto supone la necesidad de desarrollar más de una versión de la misma aplicación.

5.2.4. Java EE. Java EE (en ingles Java Platform, Enterprise Edition), es una plataforma de programación para desarrollar y ejecutar software de aplicaciones en lenguaje de programación Java con arquitectura de varios niveles distribuida, basándose ampliamente en componentes de software modulares ejecutándose sobre un servidor de aplicaciones.

La plataforma Java EE está definida por una especificación. Similar a otras especificaciones del Java Community Process, esta incluye varias especificaciones de API, tales como JDBC, RMI, e-mail, JMS, Servicios Web, XML, etc. y define cómo coordinarlos. [26]

Java EE también configura algunas especificaciones únicas para componentes. Estas incluyen Enterprise JavaBeans, servlets, portlets, JavaServer Pages y varias tecnologías de servicios Web. Esto permite al desarrollador crear una aplicación de empresa portable entre plataformas y escalable, a la vez que integrable con tecnologías anteriores.

Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, operaciones de seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse en los componentes de lógica de negocio en lugar de en tareas de mantenimiento de bajo nivel.

Servlet. La palabra servlet deriva de otra anterior, applet, que se refiere a pequeños programas escritos en Java que se ejecutan en el contexto de un navegador Web. Por otra parte, un servlet es un programa que se ejecuta en un servidor.

Un servlet es una clase del lenguaje de programación Java, que es usada para extender la capacidad del servidor que hospeda las aplicaciones, accediéndolas por medio del modelo de programación petición respuesta. Aunque los servlets pueden responder cualquier tipo de petición, son comúnmente usados para extender las aplicaciones hospedadas por servidores Web. Para este tipo de aplicaciones la tecnología Java servlet soporta el protocolo HTTP.

Los paquetes javax.servlet y javax.servlet.http proveen interfaces y clases para manejar servlets, todos los servlets deben implementar la interfaz servlet, la cual define los métodos del ciclo de vida. Cuando se implementa un servicio genérico, puede usar o extender la clase GenericServlet que se encuentra en las APIs de Java Servlet, la clase HttpServlet proporciona métodos, como doGet y doPost, para manejar servicios específicos de HTTP. Además al implementar esta interfaz el servlet es capaz de interpretar los objetos de tipo HttpServletRequest y HttpServletResponse que contienen la información de la página que invocó al servlet y un medio para enviar la respuesta respectivamente.

Un servlet como objeto que se ejecuta en un servidor o contenedor Java EE, fue especialmente diseñado para ofrecer contenido dinámico desde un servidor Web. Otras opciones que permiten generar contenido dinámico son con los JSP que resultan más apropiados para producir documentos en leguaje de marcado.

5.2.5. JDBC. JDBC es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la librería de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión, para ello provee en localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar con cualquier tipo de tareas con la base de datos

a las que tenga permiso: consultas, actualizaciones, creado modificado y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Tipos de manejadores JDBC. Los manejadores JDBC se ubican en cuatro categorías:

1. Un manejador puente JDBC-ODBC provee el API JDBC mediante uno o más manejadores ODBC. Nótese que generalmente esto requiere la instalación de código nativo en cada máquina cliente que use este tipo de manejador. De ahí que generalmente es más apropiado cuando la instalación automática de la aplicación Java no es importante.

2. Un manejador de API nativo parcialmente Java convierte los llamados JDBC en llamados al API del cliente para Oracle, Sybase, Infomix, DB2 y otros gestores. Nótese que tal como en el driver de puente, este estilo de manejador requiere que algo de código binario sea cargado en cada máquina cliente.

3. Un manejador de protocolo de red completamente Java, traduce los llamados al API JDBC a un protocolo de red independiente del gestor de base de datos que luego es traducido al protocolo del gestor por un servidor. Este middleware de servidor de red es capaz de conectar todos sus clientes a diferentes bases de datos. El protocolo específico usado depende el proveedor. En general esta es la alternativa más flexible.

4. Un manejador de protocolo nativo completamente construido en Java, convierte las llamadas JDBC a llamadas del protocolo de red usado por el gestor, directamente. Esto permite una llamada directa desde la máquina cliente al servidor de base de datos. Es la solución más práctica para acceso en intranets. Dado que muchos de estos protocolos son propietarios los proveedores serán la fuente primaria de este estilo de manejador.

5.3. BIBLIOTECAS

Las bibliotecas usadas en la construcción de Atlas son necesarias para agregar características específicas y funcionalidad a la herramienta, a continuación se lista y se describen cada una de ellas.

5.3.1. Java topology suite. La suite de topología en Java (en ingles Java Topology Suite, JTS) es una API que proporciona un modelo de objetos espaciales y funciones fundamentales geométricas 2D. JTS ha sido desarrollada por la empresa Vivid Solutions y esta implementada íntegramente en el lenguaje de programación Java, distribuyéndose bajo licencia LGPL [27], el logotipo de la empresa vivid solutions se muestra en la figura 51.

Figura 51. Logotipo de vivid solutions.

http://www.vicoug.org/images/vivid1.jpg

Cumple con la especificación de rasgos simples para SQL publicada por el Open Geospatial Consortium, proporcionando una implementación completa, consistente y robusta de algoritmos espaciales bidimensionales.

La especificación de rasgos simples (en ingles Simple Features Specification) es un estándar openGIS que especifica el almacenamiento digital de datos geográficos como punto, línea, polígono, multipunto, multi línea, etc. con atributos espaciales y no espaciales. Se basa en geometrías 2D con interpolación lineal entre los vértices. En general, una geometría 2D es simple si no se intercepta consigo misma. Además define diferentes operadores espaciales, que se puede utilizar para generar nuevas geometrías a partir de geometrías existentes.

JTS es una biblioteca ampliamente utilizada en el software SIG de código libre con funciones de análisis espacial, consultas avanzadas y creación de topología. Existe una versión de esta biblioteca en C++, llamada GEOS.

5.3.2. Geotools. Es una biblioteca de código abierto, desarrollada en Java y distribuida bajo licencia LGPL, soporta estándares, y proporciona métodos para la manipulación de datos geoespaciales, utilizados ampliamente en los sistemas de información geográfica, además implementa especificaciones del OGC a medida que se van desarrollando [28], el logotipo de GeoTools se muestra en la figura 52.

Figura 52. Logotipo de geotools.

http://media.xircles.codehaus.org/_projects/ small.PNG

Entre sus diversas características se encuentra que GeoTools soporta una gran variedad de formatos para datos vectoriales como Shapefile, GML, WFS, Postgis, Oracle Spatial, MapInfo, etc. y datos raster como ArcGrid, Image y GeoTIFF.

GeoTools implementa funcionalidades para la transformación de coordenadas, funciones útiles en el proceso de proyección de mapas como Mercator, Transverse Mercator, Lambert Conformal, Conic entre otras, maneja filtros proporcionando un subconjunto de datos para trabajar sobre ellos, y herramientas para la validación de geometrías.

5.3.3. C3PO. Biblioteca Java compatible con hibernate que administra un pool de conexiones de la base de datos con el fin de que estás puedan ser utilizadas cuando la base de datos recibe nuevas consultas, mejorando su rendimiento. Se distribuye bajo LGPL.

5.3.4. Hibernate. Es una herramienta de manejo objeto relacional de alto rendimiento para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos en formato XML que permiten establecer dichas relaciones [29], el logotipo de hibernate se muestra en la tabla 53.

Figura 53. Logotipo de hibernate.

http://www.hibernate.org/tpl/jboss/img/01_oben_logo.GIF

La licencia LGPL permite el uso de hibernate en proyectos libres y comerciales. Permite desarrollar clases persistentes siguiendo la filosofía orientada a objetos incluyendo asociación, herencia, polimorfismo, composición y colecciones. Busca solucionar el problema de la diferencia entre los dos modelos usados hoy en día para organizar y manipular datos: El usado en la memoria de la computadora (orientación a objetos) y el

usado en las bases de datos (modelo relacional). Para lograr esto permite al desarrollador detallar cómo es su modelo de datos, qué relaciones existen y qué forma tienen. Con esta información hibernate le permite a la aplicación manipular la información de la base de dato operando sobre objetos, con todas las características de la programación orienta a objetos.

Hibernate convierte los datos entre los tipos utilizados por Java y los definidos por SQL. Además genera las sentencias SQL y libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias, manteniendo la portabilidad entre todas las bases de datos con un ligero incremento en el tiempo de ejecución.

Hibernate está diseñado para ser flexible en cuanto al esquema de tablas utilizado, para poder adaptar su uso sobre una base de datos ya existente. También tiene la funcionalidad de crear la base de datos a partir de la información disponible.

Hibernate ofrece también un lenguaje de consulta de datos llamado HQL (en inglés, Hibernate Query Language), al mismo tiempo que una API para construir las consultas programáticamente (conocida como "criteria"). Hibernate puede ser utilizado en aplicaciones Java independientes o en aplicaciones Java EE, mediante el componente Hibernate Annotations que implementa el estándar JPA, que es parte de esta plataforma.

5.3.5. JDom. Es una biblioteca para manipular datos en formato XML optimizada para Java, con el fin de que pueda aprovechar las características del lenguaje, incluyendo sobrecarga de métodos, colecciones, etc.

5.3.6. KXML. Es una pequeña pero completa biblioteca Java para manipular leguajes de marcado como xml, específicamente diseñada para dispositivos con restricciones como teléfonos celulares y PDAs

5.3.7. PNG encoder. Es una biblioteca Java especializada en la codificación de imágenes formato PNG que sobresale por sus altas prestaciones en comparación con otras alternativas existentes como JAI. Se distribuye bajo una licencia particular que hace posible su uso en cualquier tipo de proyecto, por no autoriza el acceso a su código fuente.

5.4. GESTORES DE BASES DE DATOS

El propósito general de los gestores de base de datos es el de manejar de manera clara, sencilla y ordenada un conjunto de datos que posteriormente se convertirán en información relevante, Atlas utiliza un gestor de base de datos dadas sus capacidades de rendimiento y organización.

5.4.1. PostgreSQL. PostgreSQL [30] es un potente sistema de base de datos relacional orientado a objetos, de código abierto, liberado bajo la licencia BSD, Tiene más de 15 años de desarrollo activo y brinda una arquitectura probada que se ha ganado una sólida reputación de fiabilidad, integridad de datos y correcciones, funciona en los principales sistemas operativos incluyendo Linux, UNIX (AIX, BSD, HP-UX, SGI IRIX, Mac OS X, Solaris, Tru64), y Windows, el logotipo de postgresql se muestra en la figura 54.

Figura 54. Logotipo de postgresql.

http://blog.pucp.edu.pe/media/44/20060322-postgres.jpg

PostgreSQL tiene un soporte completo para claves foráneas, uniones, vistas, disparadores y procedimientos almacenados. Incluye SQL92 y SQL99 para la definición de tipos de datos, incluyendo integer, numeric, boolean, char, varchar, date, interval, y timestamp.

También soporta el almacenamiento de grandes objetos binarios, incluyendo imágenes, sonidos o vídeo. Tiene interfaces nativas de programación para C / C++, .Net, Perl, Python, Ruby, Tcl, ODBC, entre otros, además de una amplia documentación. En el caso particular de la conectividad con Java se ofrece un manejador nativo tipo IV.

El desarrollo de PostgreSQL no es manejado por una sola compañía, es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (en inglés PostgreSQL Global Development Group).

Mediante un sistema denominado Acceso concurrente multiversión (en inglés Multiversion concurrency control, MVCC) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos. Cada usuario obtiene una visión consistente de lo último sobre lo que se hizo commit. Esta estrategia es superior al uso de bloqueos por tabla o por filas común en otras bases, eliminando la necesidad del uso de bloqueos explícitos.

5.4.2. Postgis. Es un módulo que añade soporte de objetos geográficos a la base de datos objeto relacional PostgreSQL para su utilización en sistemas de información geográfica [31], el logotipo de postgis se muestra en la figura 55.

Figura 55. Logotipo de Postgis.

http://da2i.univ-lille1.fr/2007/doc/postgis.GIF

Se publica bajo licencia GNU, un aspecto a tener en cuenta es que PostGIS ha sido certificado en 2006 por el Open Geospatial Consortium, lo que garantiza la interoperabilidad con otros sistemas también interoperables. PostGIS es un producto que ha demostrado versión a versión su eficiencia.

Entre sus características se encuentra que puede usar todos los objetos que aparecen en la especificación OpenGIS como puntos, líneas, polígonos, multilíneas, multipuntos, y colecciones geométricas, además implementa un conjunto de operadores espaciales como unión, diferencia, diferencia simétrica y buffers, maneja índices espaciales y de selectividad que mejoran la velocidad de consultas espaciales.

6. ANÁLISIS, DISEÑO DE ATLAS

En un proceso de desarrollo de software las tareas de análisis y diseño, comprenden gran parte del esfuerzo total requerido, y son definitivas para lograr el éxito del proyecto. Un nombre apropiado para la etapa de análisis es, análisis de requisitos ya que hace énfasis en la investigación del problema y los requisitos en lugar de centrarse en hallar una solución. De igual forma un nombre apropiado para la etapa de diseño, es diseño de objetos o diseño de bases de datos, ya que pone énfasis en hallar una solución conceptual que satisfaga los requisitos.

6.1. PROGRAMACIÓN EXTREMA

La programación extrema o eXtreme Programming (XP) [32] es un enfoque de la ingeniería de software formulado por Kent Beck, autor del primer libro sobre la materia, Extreme Programming Explained: Embrace Change (1999). Es la más destacada de los procesos agiles de desarrollo de software. Al igual que éstos, la programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.

Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos. Creen que ser capaz de adaptarse a los cambios de requisitos en cualquier punto de la vida del proyecto es una aproximación mejor y más realista que intentar definir todos los requisitos al comienzo del proyecto e invertir esfuerzos después en controlar los cambios en los requisitos.

Se puede considerar la programación extrema como la adopción de las mejores metodologías de desarrollo de acuerdo a lo que se pretende llevar a cabo con el proyecto, y aplicarlo de manera dinámica durante el ciclo de vida del software.

6.1.1. El objetivo de la programación extrema.

- Un intento de reconciliar humanidad y productividad.
- Un mecanismo para el cambio social.
- Un camino al mejoramiento.
- Un estilo de desarrollo.
- Una disciplina de desarrollo de software.

El principal objetivo de la programación extrema es reducir el costo del cambio. En metodologías tradicionales de desarrollo los requerimientos para el sistema son determinados al inicio del proyecto y usualmente permanecen fijos desde este punto. Esto implica que el costo de cambiar los requerimientos a una etapa posterior será alto.

La programación extrema propone reducir el costo de cambio al introducir valores, principios y prácticas básicas. Al aplicar programación extrema, un proyecto de software deber ser más flexible respecto al cambio.

6.1.2. Valores de la programación extrema.

Comunicación. La construcción de un sistema de software requiere comunicar los requerimientos del sistema a los desarrolladores. En metodologías de desarrollo formales, esta tarea se logra mediante documentación. Las técnicas de programación extrema pueden ser vistas como métodos rápidos para construir y diseminar conocimiento entre los miembros de un equipo de desarrollo.

El objetivo es dar a todos los desarrolladores una visión común del sistema que coincida con la visión del los usuarios. Para este fin la programación extrema va a favor de diseños simples, metáforas comunes, colaboración entre usuarios y programadores, comunicación verbal frecuente y realimentación.

Simplicidad. La programación extrema invita a iniciar con la solución más simple la funcionalidad adicional puede agregarse después. La diferencia entre este enfoque y los métodos más convencionales de desarrollo es el énfasis en que se debe diseñar y codificar para las necesidades de hoy en lugar de aquellas del futuro. Los proponentes de la programación extrema reconocen que la desventaja de esto es que puede acarrear más esfuerzo futuro para cambiar el sistema; ellos afirman que esto es más que compensado por la ventaja de no invertir en posibles requerimientos futuros que podrían cambiar antes de que se vuelvan relevantes. La codificación y el diseño para requisitos futuros inciertos implican el riesgo de gastar recursos en algo que podría no necesitarse.

Realimentación. En programación extrema la realimentación toma diferentes dimensiones del desarrollo del sistema.

Realimentación del sistema. Al escribir pruebas de unidad o correr pruebas de integración periódicas los programadores tienen realimentación directa del estado del sistema después de implementar los cambios.

Realimentación del cliente. Las pruebas funcionales son escritas por el cliente y el equipo de pruebas. Ellos recibirán realimentación concreta sobre el estado actual de su sistema. Esta revisión es planeada una vez cada dos o tres semanas así que el cliente puede dirigir fácilmente el desarrollo.

Realimentación del equipo. Cuando los clientes llegan con nuevos requerimientos al juego de planeación, el equipo directamente entrega una estimación del tiempo que tomará implementarlos.

La realimentación está cercanamente relacionada con la comunicación y la simplicidad. Las fallas en el sistema son fácilmente comunicadas escribiendo pruebas de unidad que comprueben que cierta parte de código fallará.

La realimentación directa del sistema dice a los programadores que deben re codificar esta parte. Un cliente está en capacidad de probar el sistema periódicamente de acuerdo a los requerimientos funcionales, conocidos como historias de usuario.

Valor. Varias prácticas implican valor. Una es el mandato de diseñar y codificar para hoy y no para mañana. El valor permite a los desarrolladores sentirse cómodos con la re factorización de su código cuando se necesita. Esto significa revisar el sistema existente y modificarlo de modo que los cambios futuros puedan implementarse más fácilmente.

Otro ejemplo de valor es saber cuándo descartar código; valor para remover código fuente que es obsoleto, sin importar cuánto esfuerzo fue colocado el su creación. El valor también implica persistencia; un programador podría estar atascado en un problema complejo un día entero, y entonces resolverlo rápidamente al día siguiente solo si es persistente.

Respeto. El respeto se manifiesta en varias formas. En programación extrema, los miembros del equipo se respetan mutuamente por que nunca envían cambios que interfieran con la compilación, que hagan que las pruebas de unidad fallen, o que retrasen el trabajo de sus colegas. Los miembros respetan su trabajo al buscar siempre la alta calidad y el mejor diseño para la solución.

6.1.3. Principios de la programación extrema.

Los principios que forman la base de la programación extrema están fundamentados en los valores anteriores y están pensados para ser decisiones adoptadas en un proyecto. Los principios son más concretos que los valores y se pueden traducir de forma más fácil en guías ante una situación práctica.

Realimentación. Es más útil cuando se hace rápidamente. El tiempo entre una acción y su realimentación es crítico para el aprendizaje y la realización de cambios. En programación extrema, el contacto con el cliente ocurre en pequeñas iteraciones. El cliente tiene una comprensión clara del sistema en desarrollo y puede dar realimentación y dirigir el desarrollo conforme se necesite.

Las pruebas de unidad contribuyen también con el principio de realimentación rápida. Al escribir código, las pruebas de unidad proveen realimentación directa sobre la forma en que el sistema reacciona a los cambios que alguien ha hecho. Si por ejemplo, los cambios afectan un área de sistema fuera del alcance del programador, quien hizo el cambio no se dará cuenta del error hasta que corra las pruebas de unidad.

Asumir la simplicidad. Trata sobre abordar cada problema como si la solución fuese extremadamente simple. Los métodos tradicionales dicen que debe planearse para el futuro y codificar para la reutilización. La programación extrema rechaza estas ideas.

Los defensores de la programación extrema dicen que hacer grandes cambios juntos una vez no funciona. La programación extrema aplica cambios incrementales; por ejemplo, un sistema podría generar pequeñas entregas cada tres semanas, al dar pequeños pasos cada vez, el cliente tiene más control sobre el proceso de desarrollo y el sistema que es desarrollado.

Adoptar el cambio. Trata sobre no trabajar contra los cambios sino adoptarlos. Por ejemplo, si en uno de los encuentros iterativos parece que los requerimientos del cliente han cambiado dramáticamente, los programadores deben aceptar esto e incluir los cambios para la próxima iteración.

6.1.4. Actividades de la programación extrema.

La programación extrema describe cuatro actividades básicas que se realizan durante el proceso de desarrollo de software.

Codificar. Los defensores de la programación extrema argumentan que el único producto verdaderamente importante de un proceso de desarrollo es el código.

La codificación puede ser la creación de diagramas que generen código, generar scripts en un sistema Web o codificar un programa que requiere compilación.

La codificación también puede ser usada para hallar la solución apropiada. Por ejemplo, la programación extrema podría sugerir que al enfrentarse a varias alternativas a un

problema de programación, simplemente se podría codificarlas todas y determinar cuál es mejor mediante pruebas automatizadas.

La codificación también puede ayudar a comunicar pensamientos sobre problemas de programación. Un programador enfrentado con un problema complejo que encuentra difícil explicar la solución a sus compañeros puede usar código para demostrar lo que quiere decir.

Probar. No se puede estar seguro de algo a menos que se haya probado, en un proceso de desarrollo puede existir incertidumbre en varios campos.

Puede existir incertidumbre respecto a si lo que fue codificado corresponde con lo que se quería decir. Para probar esta incertidumbre se usan las pruebas de unidad. Estas son pruebas automatizadas que prueban el código. El programador escribirá las pruebas que piense que pueden romper el código que está escribiendo, si todas las pruebas pasan con éxito, entonces el código está terminado.

Puede existir también incertidumbre respecto a que lo que se dice sea lo que debe decirse. Para probar esta incertidumbre se usan pruebas de aceptación basadas en los requerimientos dados por el cliente en la fase de exploración de la planeación de entrega.

Escuchar. Los programadores no necesariamente conocen todo sobre el lado del negocio de un sistema en desarrollo. La función del sistema es dada por el lado del negocio. Para que los programadores puedan determinar la funcionalidad que el sistema debe tener deben escuchar dicho lado.

Los programadores deben escuchar lo que el cliente necesita, también deben tratar de entender el problema del negocio y dar al cliente realimentación sobre su problema. La comunicación entre el cliente y el programador es tratada más adelante en el juego de planeación.

Diseñar. Desde el punto de vista de la simplicidad, se puede decir que un sistema no necesita más que codificación, pruebas y escucha. Si estas actividades se realizan bien, el resultado siempre será un sistema que funciona. En la práctica esto no es cierto, puede pasar mucho tiempo sin diseñar, pero en algún punto dado el proceso se quedará atascado; el sistema se torna demasiado complejo y las dependencias dejan de ser claras. Esto se puede prevenir creando un diseño estructural que organice la lógica en el sistema.

6.1.5. Prácticas de la programación extrema.

La programación extrema tiene doce prácticas agrupadas en cuatro áreas.

Realimentación

- **Programación en parejas.** Es una técnica de desarrollo de software en la cual dos programadores trabajan juntos en un teclado. Uno digita el código mientras el otro revisa cada línea escrita. Quien escribe se denomina guía, quien revisa se denomina observador. Los dos programadores cambian roles con frecuencia.

- **Juego de planeación.** Es un encuentro que ocurre una vez por iteración, típicamente una vez por semana.

  El proceso de planeación de divide en dos partes

  Planeación de entregas. Se enfoca en la determinación de los requerimientos a cubrir en las entregas más cercanas, y cuando deben entregarse. Los clientes y desarrolladores toman parte en esto. La planeación de entregas consta de tres fases.

  Fase de exploración. Aquí, el cliente provee una breve lista de requerimientos de alto valor para el sistema. Estos serán transcritos en tarjetas de historias de usuario.

  Fase de acuerdo. Aquí gente del negocio y del desarrollo se acuerda la funcionalidad que se incluirá en la fecha de la próxima entrega.

  Fase de dirección. Aquí, el plan puede ser ajustado, se pueden agregar nuevos requisitos u otros requisitos existentes pueden modificarse o removerse.

  Planeación de iteración. Aquí se planea las actividades y tareas de los desarrolladores. En este proceso no se involucra al cliente, la planeación de la iteración también consiste en tres fases.

  Fase de exploración. Aquí los requerimientos se traducen en diferentes tareas que son almacenadas en tarjetas de tareas.

  Fase de acuerdo. Las tareas se asignan a los programadores y el tiempo que toman será estimado.

Fase de dirección. Las tareas son realizadas y el resultado final es comparado con la historia de usuario original.

- **Desarrollo orientado a pruebas.**

  Las pruebas de unidad están automatizadas y sirven para probar la funcionalidad de partes del código como clases o métodos. Las pruebas de unidad son escritas antes que el código para estimular al programador a pensar en las condiciones en que su código puede fallar. Se dice que un programador ha terminado una parte de código cuando no puede generar ninguna otra condición en que este pueda fallar.

- **Equipo completo.** El cliente es quien realmente usa el sistema. El cliente debe estar disponible para las preguntas que puedan surgir. Por ejemplo, un equipo desarrollando un sistema de administración financiera debe incluir un administrador financiero.

Proceso continuo.

- **Integración continua.** El equipo siempre debe trabajar en la última versión del software. Como los distintos miembros del equipo pueden tener versiones guardadas localmente con cambios y mejoras, deben procurar subir su versión actual al repositorio en términos de horas o cuando se presente un cambio significativo. La integración continua evitará retrasos en el ciclo del proyecto debidas a problemas de integración.

- **Mejoras de diseño.** Como se sugiere programar solo lo que se necesita hoy, e implementarlo tan simple como sea posible, a veces esto termina atascando el sistema. Uno de los síntomas es la necesidad de mantenimiento dual o múltiple, los cambios funcionales requieren modificaciones en múltiples copias del código igual o similar. Otro síntoma es que al cambiar una parte del código se afecta muchas partes. Cuando esto sucede el sistema está diciendo que debe re fabricarse el código cambiando la arquitectura para hacerla más simple y genérica.

- **Pequeñas entregas.** Las entregas de software están predeterminadas. El plan de entregas es determinado al iniciar el proyecto. Usualmente cada entrega llevará un pequeño segmento del software total el cual puede correr sin depender de componentes que se construirán en el futuro. Las entregas pequeñas ayudan al cliente a ganar confianza en el progreso del proyecto.

Comprensión colectiva

- **Estándares de código.** Es un conjunto aceptado de reglas que el equipo entero acepta seguir a lo largo del proyecto. El estándar especificará un estilo y formato consistente de código fuente de acuerdo al leguaje seleccionado.

- **Propiedad colectiva del código.** Significa que todos son responsables por todo el código y cualquiera está autorizado a cambiar cualquier parte de este. Una ventaja de la propiedad colectiva del código es que agiliza el desarrollo ya que si un error ocurre cualquiera puede arreglarlo.

  Al darle a cualquier programador el derecho de cambiar el código existe el riesgo de errores introducidos por quienes que creen saber lo que hacer pero no prevén ciertas dependencias. Las pruebas de unidad bien definidas resuelven el problema.

- **Diseño simple.** Los programadores deben aplicar el enfoque "simple es mejor" al diseño del software. Siempre que se escribe una parte de código los autores deben preguntarse si han hallado la forma más simple de agregar una funcionalidad.

- **Metáfora del sistema.** Es un concepto de nombrado para clases y métodos que debe hacer fácil para un miembro del equipo suponer la funcionalidad de una clase o método particular solo con base en su nombre.

Bienestar del programador

- **Ritmo sostenible.** Indica que los programadores o desarrolladores no deben trabajar más de 40 horas por semana, y que si hay alguna irregularidad en una semana la próxima debe ser normal. Como los ciclos de desarrollo son frecuentes, cortos y de integración continua, los proyectos de programación extrema no siguen el típico tiempo critico que otros proyectos requieren.

6.2. MODELADO ÁGIL

Modelado ágil es una colección de valores, principios y prácticas para modelar software que pueden ser aplicadas a proyectos de desarrollo en una forma efectiva y ligera. Está ideada para trabajar en conjunto con otras metodologías como programación extrema, permitiendo desarrollar un proceso de software que realmente cumpla las necesidades. [33]

6.2.1. Valores del modelado ágil.

Comunicación. Los modelos promueven la comunicación entre los inversionistas y el equipo así como entre los desarrolladores del equipo.

Simpleza. Es importante que los desarrolladores comprendan que los modelos son críticos a la hora de simplificar el software y el proceso de software. Se considera más efectivo explorar una idea y mejorarla mediante la creación de algunos diagramas que escribir oraciones o código.

Realimentación. El optimismo es un riesgo profesional de la programación. Al comunicar ideas mediante diagramas se obtiene realimentación rápida que permite actuar sobre este hecho.

Valor. El valor es importante porque es necesario para cambiar la dirección, descartando o modificando el trabajo cuando se prueba alguna decisión como inadecuada.

Humildad. Los mejores desarrolladores tienen la humildad para reconocer que no lo saben todo y que otros desarrolladores y sus clientes tienen sus propias áreas de experticia y pueden agregar valor al proyecto.

6.2.2. Principios esenciales del modelado ágil.

Crear modelos con un propósito. Al crear un artefacto debe tenerse en cuenta la razón por la que se crea y para quien se crea. Es posible crear modelos para mejorar la compresión de algún aspecto del software, para comunicar un enfoque, o para describir el sistema a quienes se encargarán de la operación o el mantenimiento. Si no es posible determinar el propósito de un modelo posiblemente no haga falta construirlo.

Maximizar la inversión. Los inversionistas en el proyecto están aportando recursos para tener un software desarrollado para satisfacer sus necesidades. Ellos merecen invertir sus recursos de la mejor manera posible sin que el equipo los desperdicie.

Viajar ligero. Cada artefacto creado que se decide conservar necesitará ser mantenido sobre el tiempo. Si se decide mantener siete modelos, entonces siempre que un cambio ocurra se debe considerar el impacto del cambio en los siete modelos.

Múltiples modelos. Potencialmente se requiere el uso de múltiples modelos para desarrollar software, ya que cada modelo cubre solo un aspecto. Considerando la complejidad del software actual se necesita tener un amplio rango de herramientas intelectuales de modelado para ser efectivo. Un punto importante es que no se necesita

desarrollar todos estos modelos para cualquier sistema, esto depende de la naturaleza del software.

Realimentación rápida. El tiempo entre una acción y la realimentación sobre ella es crítico. Al trabajar con otros en un modelo, particularmente cuando se trabaja con tecnología de modelado compartido, se obtiene realimentación instantánea de las ideas.

Trabajar cerca al cliente para entender los requerimientos, para analizar esos requerimientos o para desarrollar una interfaz de usuario, provee oportunidades de obtener una realimentación rápida.

Adoptar la simplicidad. Al desarrollar se debe asumir que la solución más simple es la mejor, no debe sobre construirse el software, o en el caso de modelado ágil, no colocar más información en los modelos de la que se necesita hoy.

Afrontar el cambio. Los requerimientos evolucionan con el tiempo. La compresión de las personas sobre los requerimientos cambia. Los interesados en el proyecto pueden cambiar durante la ejecución, se agregan nuevas personas y quienes estaban se retiran. Los inversionistas pueden cambiar su punto de vista, cambiando los criterios de éxito. La implicación es que el ambiente del proyecto cambia y el enfoque del desarrollo debe reflejar esta realidad.

Cambio incremental. Un concepto importante que debe comprenderse respecto al modelado es que no es necesario hacerlo perfecto la primera vez, de hecho, no podría hacerse aunque se quisiera. Es más, los modelos no necesitan capturar cada detalle individual, solo se necesita que sean suficientemente buenos en el momento. En lugar de tratar infructuosamente el desarrollo de un modelo total desde el principio, es posible desarrollar un modelo pequeño, tal vez de alto nivel y hacer que evolucione con el tiempo o descartarlo si ya no hace falta.

Trabajo de calidad. A nadie le gusta el trabajo mediocre. A quienes lo hacen les disgusta por que no es algo de lo que puedan sentirse orgullosos, a quienes llegan después para re fabricar les desagrada porque es difícil de entender y actualizar, y a los usuarios finales no les gusta porque parece fragil y/o no alcanza sus expectativas.

Software funcional es el principal objetivo. El objetivo del desarrollo de software es producir software de alta calidad que alcance las necesidades del proyecto de forma efectiva. El objetivo principal no es producir documentación, artefactos o modelos extraños. Cualquier actividad que no contribuya con este objetivo debe ser evaluada y evitada si no puede justificarse a esta luz.

Facilitar el próximo esfuerzo es el objetivo secundario. El proyecto puede considerarse una falla aun cuando el equipo entregue un sistema funcional a los usuarios, parte de satisfacer las necesidades de los usuarios es asegurarse de que el sistema sea lo suficientemente robusto para extenderse con el tiempo.

El próximo esfuerzo puede ser el desarrollo de una entrega mayor o el soporte de la versión actual. Para facilitar esto no solo es deseable desarrollar software de calidad sino también crear suficiente documentación y material de soporte para que quienes se ocupen del próximo esfuerzo puedan ser efectivos. Los factores que deben considerarse incluyen la permanencia de los miembros del equipo actual en el próximo esfuerzo y la naturaleza del esfuerzo en si misma.

6.2.3. Prácticas esenciales del modelado ágil.

Participación efectiva del cliente. Esto incluye la participación de usuarios directos, administradores, y personal de soporte. La participación del cliente puede promoverse fácilmente al adoptar técnicas de modelado inclusivas.

Modelar con otros. Al construir un modelo con un propósito usualmente se encuentra que se modela para entender algo, para comunicar algo, o para construir una visión común del proyecto. Esta es una actividad de grupo en la que de desea el aporte de varias personas. El equipo de desarrollo necesita trabajar en grupo para desarrollar el conjunto de modelos esenciales del proyecto.

Aplicar los artefactos correctos. Cada artefacto tiene sus aplicaciones específicas. Por ejemplo, un diagrama UML de actividad es útil para describir procesos de negocio, mientras que la estructura estática de la base de datos se representa mejor por un modelo de datos físicos o un modelo de persistencia. Con frecuencia un diagrama es una mejor elección que el código fuente.

Iterar a otro artefacto. Cuando al trabajar en un artefacto de desarrollo, como un caso de uso, un diagrama de secuencia o inclusive código fuente se encuentra que se está atascado debe considerarse el trabajar con otro artefacto. Cada artefacto tiene sus fortalezas y debilidades.

Demostrarlo con código. Un modelo es una abstracción, para determinar si funcionará, el modelo debe probarse con código. Nunca debe olvidarse que con un enfoque iterativo de software la norma para una amplia mayoría de proyectos es que el modelado es solo una de las muchas tareas que deben realizarse.

Usar las herramientas más simples. La gran mayoría de modelos se pueden dibujar en un tablero o en papel. Siempre que se desee guardar uno de esos diagramas puede tomarse una fotografía digital de el o simplemente trascribirlo en papel. Esto funciona porque la mayoría de diagramas son desechables, su valor esta dado al dibujarlos para pensar en un tema, una vez el tema está resuelto el diagrama no tiene gran valor.

Modelar en incrementos pequeños. El desarrollo incremental, en el que se organiza un gran esfuerzo en pequeñas porciones que se entregan con el tiempo, incrementa la agilidad al poder entregar resultados rápidamente.

Única fuente de información. La información debe ser almacenada en un solo lugar, no solo se debe elegir el artefacto correcto, también se debe modelar el mismo concepto una y otra vez, almacenando la información en el mejor lugar posible.

Propiedad colectiva. Todos pueden trabajar en cualquier modelo y de hecho, en cualquier artefacto en el proyecto.

Crear varios modelos en paralelo. Como cada tipo de modelo tiene sus fortalezas y debilidades un solo modelo no es suficiente. En combinación con la práctica de iterar a otro artefacto los modeladores agiles descubren con frecuencia que son más productivos al trabajar con varios modelos en lugar enfocarse solo en uno.

Crear contenido simple. El contenido actual de los modelos debe mantenerse tan simple como sea posible mientras cumpla las necesidades. No debe agregarse detalles adicionales a los modelos a menos que sea justificable.

Representar los modelos con simplicidad. Cuando se considera los diagramas que podrían aplicarse, rápidamente se nota que la mayor parte del tiempo solo hace falta un subconjunto de la notación de diagramación disponible. Un modelo simple que muestra las características claves que se pretende entender, tal vez un modelo de clases describiendo las responsabilidades principales de estas y sus relaciones usualmente son suficientes.

Mostrar los modelos públicamente. Esto habilita una comunicación honesta en el equipo porque todos los modelos actuales son fácilmente accedidos por todos.

6.3. ANÁLISIS UML.

Para el análisis y diseño de Atlas se utilizó Extreme Programming como metodología de desarrollo, que pone un mayor énfasis en la adaptabilidad que en la previsibilidad, característica deseable en este tipo de proyectos, además, propone estrategias aplicables a todo el proceso de desarrollo. Para modelar el sistema se utiliza el enfoque de análisis y diseño orientado a objetos que modela un sistema como un grupo de objetos que interactúan entre sí, dichos modelos se representan con la notación visual estándar UML y se administran de acuerdo a la estrategias planteadas por el modelado ágil.

6.3.1. Funciones. Para tener una visión general del proyecto, se lista en la siguiente tabla los requerimientos obligatorios y opcionales de Atlas.

Ref	Función	Cat.	Atributo	Detalles y restricciones	Cat.
R1	Generar archivos WAR para el contenedor de servlets.	Evidente.	Interfaz.		Obligatorio.
R1.1	Incluir datos para el despliegue y conexión.	Evidente.	Interfaz.		Obligatorio.
R1.2	Administrar el listado de plugins de geocodificación.	Evidente.	Interfaz.	Debe descartar archivos JAR que no sean plugins.	Obligatorio.
R1.3	Generar un archivo WAR con base en la platilla de configuración.	Oculto.	Información.		Obligatorio.
R2	Responder peticiones vía HTTP.	Evidente.	Interfaz.		Obligatorio.
R2.1	Servir contenido de acuerdo al estándar WMS.	Evidente.	Interfaz.	En las versiones 1.1.0, 1.1.1 y 1.3.0	Obligatorio.
R2.1.1	Responder peticiones GetCapabilities.	Evidente.	Interfaz.		Obligatorio.
R2.1.2	Responder peticiones GetMap.	Evidente.	Interfaz.	En formatos JPEG y PNG.	Obligatorio.

R2.1.3	Responder peticiones GetFeatureInfo.	Evidente.	Interfaz.	En formatos texto plano, GML y XML.	Opcional.
R2.2	Responder peticiones simultáneamente.	Oculto.	Información.		Opcional.
R2.3	Responder peticiones GetLeyend.	Evidente.	Interfaz.		Opcional.
R2.4	Responder peticiones GetEPSGInfo.	Evidente.	Interfaz.	En formato XML.	Obligatorio.
R2.5	Responder peticiones de geocodificación.	Evidente.	Interfaz.		Obligatorio.
R2.5.1	Permitir adicionar geocoders como plugins.	Oculto.	Información.	Los plugins siguen una especificación definida por el proyecto.	Opcional.
R2.6	Validar el acceso al servidor.	Evidente.	Interfaz.		Obligatorio.
R2.7	Mejorar la capacidad de respuesta manteniendo buffers en memoria	Oculto.	Información.		Opcional.
R2.8	Mejorar la capacidad de respuesta manteniendo un caché de las imágenes generadas.	Oculto.	Información.	Restringido a un tamaño dado.	Opcional.
R2.9	Validar la forma de las peticiones y reportar errores.	Evidente.	Interfaz.		Obligatorio.
R3	Responder peticiones desde el panel de control.	Evidente.	Interfaz.		Obligatorio.
R3.1	Establecer conexión con el panel de control.	Oculto.	Información.	Solo desde el panel de control.	Obligatorio.
R3.2	Reiniciar el servicio.	Oculto.	Información.		Obligatorio.
R3.3	Codificar las comunicaciones con información privada.	Evidente.	Interfaz.	Sumas MD5 y AES.	Opcional.

R4	Administrar el contenido del servidor Atlas con un panel de control.	Evidente.	Interfaz.	Aplicación de escritorio.	Obligatorio.
R4.1	Establecer conexión con el servidor Atlas.	Evidente.	Interfaz.		Obligatorio.
R4.2	Establecer los metadatos del servicio WMS.	Evidente.	Interfaz.		Obligatorio.
R4.3	Administrar los iconos del servidor Atlas.	Evidente.	Interfaz.	Formato SVG.	Obligatorio.
R4.4	Administrar los proyectos del servidor Atlas.	Evidente.	Interfaz.		Obligatorio.
R4.5	Administrar los orígenes de datos por proyecto.	Evidente.	Interfaz.		Obligatorio.
R4.5.1	Agregar orígenes de datos ShapeFile.	Evidente.	Interfaz.		Obligatorio.
R4.5.2	Agregar orígenes de datos Postgis.	Evidente.	Interfaz.	En la base de datos del sistema	Opcional.
R4.6	Administrar geocoders por proyecto.	Evidente.	Interfaz.		Obligatorio.
R4.7	Administrar capas por proyecto.	Evidente.	Interfaz.		Obligatorio.
R4.8	Administrar visualización de las capas.	Evidente.	Interfaz.		Obligatorio.
R4.8.1	Cambiar orden de presentación capas.	Evidente.	Interfaz.		Opcional.
R4.8.2	Administrar el etiquetado por capa.	Evidente.	Interfaz.		Obligatorio.
R4.8.3	Detectar colisiones entre etiquetas.	Evidente.	Interfaz.	Tipos de fuentes Java.	Obligatorio.
R4.9	Administrar simbología por capa.	Evidente.	Interfaz.	Para capas tipo punto, línea y polígono.	Obligatorio.
R4.9.1	Administrar simbología fija.	Evidente.	Interfaz.		Obligatorio.
R4.9.2	Administrar simbología por clases.	Evidente.	Interfaz.	Identificadas en los registros de la capa.	Obligatorio.

R4.9.3	Administrar simbología por rangos.	Evidente.	Interfaz.	Formados desde un atributo numérico.	Obligatorio.
R4.10	Visualizar los efectos de los cambios en el contenido.	Evidente.	Interfaz.	Presentar un mapa de un alto y ancho definido.	Obligatorio.
R4.11	Solicitar información GetFeatureInfo para un proyecto.	Evidente.	Interfaz.		Obligatorio.
R4.12	Consultar al geocoder de un proyecto.	Evidente.	Interfaz.		Obligatorio.
R4.13	Establecer las opciones de configuración del panel de control.	Evidente.	Interfaz.	Apariencia, Idioma y plugins.	Obligatorio.
R5	Permitir el desarrollo de aplicaciones compatibles con el servidor Atlas.	Evidente.	Interfaz.	En ambientes de escritorio, navegadores Web y dispositivos móviles.	Obligatorio.
R5.1	Establecer conexión con un servidor Atlas.	Evidente.	Interfaz.		Obligatorio.
R5.2	Recuperar información sobre el contenido del servidor.	Evidente.	Interfaz.	Procesando el documento GetCapabilities.	Obligatorio.
R5.2.1	Recuperar información sobre los proyectos.	Evidente.	Interfaz.		Obligatorio.
R5.2.2	Recuperar información sobre las capas.	Evidente.	Interfaz.		Obligatorio.
R5.3	Visualizar un mapa con las capas de un proyecto.	Evidente.	Interfaz.		Obligatorio.
R5.4	Interactuar con el mapa de un proyecto.	Evidente.	Interfaz.	Incluye acciones con el mouse.	Obligatorio.
R5.4.1	Cambiar el nivel de detalle de la visualización.	Evidente.	Interfaz.	Respetar denominadores de escala WMS.	Obligatorio.

R5.4.2	Desplazarse en el mapa.	Evidente.	Interfaz.		Obligatorio.
R5.4.3	Seleccionar marcadores.	Evidente.	Interfaz.		Obligatorio.
R5.5	Administrar marcadores en un mapa.	Evidente.	Interfaz.	Personalizando visualización.	Obligatorio.
R5.6	Solicitar información GetFeatureInfo para un proyecto.	Evidente.	Interfaz.		Obligatorio.
R5.7	Consultar al geocoder de un proyecto.	Evidente.	Interfaz.		Obligatorio.
R5.8	Mejorar el rendimiento dividiendo un mapa en varias imágenes.	Oculto.	Información.	Debe favorecerse la reutilización de las imágenes.	Obligatorio.
R5.9	Mejorar el rendimiento formando un caché de imágenes.	Oculto.	Información.		Obligatorio.
R5.10	Consultar las convenciones de un mapa.	Evidente.	Interfaz.	En formato PNG.	Obligatorio.
R5.11	Mejorar el rendimiento realizando peticiones simultáneas.	Oculto	Información.		Obligatorio.

6.3.2. Casos de uso del panel de control. El panel de control permite la administración del contenido del servidor. Su actor principal es el administrador del sistema y sus casos de uso se muestran en las siguientes tablas.

Caso de uso establecer conexión con un servidor Atlas.

Caso de uso	Establecer conexión con un servidor Atlas.
Actor principal	Administrador.
Otros actores	Servidor Atlas. Sistema gestor de base de datos.
Pos condiciones	El panel de control ha establecido conexión un servidor y el administrador puede proceder a cambiar el contenido.
Flujo básico	
1. El administrador ingresa los datos de la conexión que son, URL, nombre de usuario	

y contraseña.
2. El panel de control verifica la forma de la URL y se cerciora de que los campos de nombre de usuario y contraseña contengan valores.
3. El panel de control verifica la conexión TCP/IP con la dirección indicada en la URL.
4. El panel de control verifica la conexión HTTP con la dirección indicada en la URL.
5. El panel de control verifica la presencia de un servidor Atlas en la URL indicada.
6. El panel de control verifica la conexión entre el servidor y la base de datos.
7. El panel de control verifica la conexión entre él y la base de datos.
8. El panel de control verifica la estructura de tablas en la base de datos.
9. El panel de control verifica la instalación correcta de postgis en la base de datos.
10. El panel de control verifica la presencia de los metadatos del servicio.
11. El panel de control verifica las condiciones de acceso al servicio.
12. El panel de control ha establecido conexión con un servidor Atlas.

Flujo alternativo

1a. El administrador abandona el formulario.

2a. Si los datos no están completos, el panel de control notifica al administrador.

3a. En caso de fallar la conexión TCP/IP, el panel de control informa al administrador y regresa al paso 1.

4a. En caso de fallar la conexión HTTP, el panel de control informa al administrador y regresa al paso 1.

5a. En caso de no encontrar un servidor Atlas, el panel de control informa al administrador y regresa al paso 1.

6a. En caso de no lograr una conexión entre el servidor Atlas y la base de datos, el panel de control informa al administrador, presenta el error generado desde postgres y regresa al paso 1.

7a. En caso de no lograr una conexión entre el panel de control y la base de datos, el panel de control informa al administrador, presenta el error generado desde postgres y regresa al paso 1.

8a. En caso de que las tablas del sistema Atlas no estén instaladas en el gestor, el panel de control preguntará al administrador si desea instalarlas.

Si el administrador acepta crear las tablas, el panel de control las instala y se dirige al paso 9 del flujo básico.

Si el administrador decide no crear las tablas, el panel de control notifica el error y regresa a 1 del flujo básico.

9a. Si postgis no está presente en la base de datos, el panel de control lanza una advertencia y pasa a 10.

10a. Si los metadatos del servicio no están presentes, el panel de control, presenta el formulario que permite diligenciarlos llamando al caso de uso, establecer los metadatos del servicio Atlas WMS.

1. Si el administrador diligencia los datos correctamente, el proceso pasa a 11.
2. Si el administrador no diligencia los datos, el proceso de conexión informa el error y regresa a 1.

11a. Si el servidor está configurado solo para uso privado, el Panel de control lanza una advertencia y continúa a 12.

Caso de uso establecer los metadatos del servicio Atlas WMS.

Caso de uso	Establecer los metadatos del servicio Atlas WMS.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con el servidor. El administrador intenta establecer conexión con el servidor y ha pasado la prueba de estructura de base de datos.
Pos condiciones	Los metadatos del servidor Atlas exigidos por el estándar WMS se han ingresado correctamente.
Flujo básico	
1. Se presenta al administrador los campos, titulo, resumen, palabras clave, límite de capas, ancho máximo, alto máximo, URL del proveedor, URL del servicio y correo electrónico, que son obligatorios y cuyo significado corresponde al estándar WMS. 2. El administrador diligencia los campos y continúa. 3. El panel de control verifica la presencia de los campos requeridos y la forma correcta en direcciones URL y de correo electrónico. 4. El panel de control procede a guardar los datos y actualizar el servidor.	
Flujo alternativo	
3a. Si los campos requeridos no están presentes o su contenido no es válido, el panel de control informa al usuario para que este corrija el error. 2a. El administrador abandona el formulario.	

Caso de uso agregar iconos al servidor.

Caso de uso	Agregar iconos al servidor.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor.
Flujo básico	
1. El panel de control presenta vistas miniatura de los iconos presentes en el servidor. 2. El administrador selecciona la opción de agregar icono. 3. El panel de control presenta un nuevo listado de iconos, inicialmente vacío. 4. El administrador elije la opción adicionar y el panel de control presenta el cuadro de diálogo de abrir archivos, mostrando solo archivos SVG y directorios. 5. El usuario selecciona un archivo o un directorio.	

6. Los iconos encontrados se agregan al listado. 7. Si el administrador está satisfecho con la selección puede continuar a 8, de lo contrario puede regresar a 4 y agregar más iconos. 8. Los iconos se agregan y se actualiza el listado principal.
Flujo alternativo
2a. El administrador abandona el formulario. 4a. El administrador cancela la acción y regresa a 1. 5a. El administrador abandona el cuadro de diálogo y regresa a 3. 6a. Si el contenido del archivo no es un SVG valido, el panel de control regresa a 3. 6b. Si el administrador selecciona un directorio, el administrador busca todos los archivos, de ser validos los agrega al listado. 7a. El administrador cancela la operación y regresa a 1 sin realizar ningún cambio.

Caso de uso remover iconos del servidor.

Caso de uso	Remover iconos del servidor.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor.
Pos condiciones	Se ha removido un icono del sistema.
Flujo básico	
1. El panel de control presenta vistas miniatura de los iconos presentes en el servidor. 2. El administrador selecciona un icono de la lista. 3. El administrador selecciona la opción eliminar. 4. El icono se elimina y se actualiza el listado.	
Flujo alternativo	
1a. El administrador abandona el formulario. 2a. Si el administrador no selecciona un ningún icono y pasa a 3, el panel de control informa que debe existir un icono seleccionado. 4b. Si el icono seleccionado es usando por alguna simbología, el panel de control informa al administrador y no realiza ningún cambio.	

Caso de uso agregar un proyecto al servidor Atlas.

Caso de uso	Agregar un proyecto al servidor Atlas.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Un proyecto nuevo se ha agregado al servidor.

Flujo básico
1. El administrador selecciona la opción agregar proyecto. 2. El panel de control presenta un formulario con los campos, nombre del proyecto y panel de control espacial de referencia. 3. El administrador completa el campo nombre del proyecto. 4. El administrador selecciona la opción seleccionar sistema espacial de referencia y pasa al caso de uso seleccionar sistema espacial de referencia. 5. El panel de control verifica la presencia de datos del formulario y crea el nuevo proyecto.
Flujo alternativo
1a. El administrador cancela la acción. 5a. Si los datos no están presentes, el panel de control informa al administrador que debe diligenciar el formulario completo y regresa a 2.

Caso de uso eliminar un proyecto del servidor Atlas.

Caso de uso	Eliminar un proyecto del servidor Atlas.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.
Pos condiciones	Se ha eliminado un proyecto del servidor Atlas.
Flujo básico	
1. El panel de control presenta una lista de los proyectos existentes. 2. El administrador selecciona el proyecto que desea eliminar y selecciona la opción eliminar. 3. El panel de control presenta un diálogo de confirmación para la acción. 4. El administrador decide continuar. 5. El panel de control remueve el proyecto del servidor de forma permanente.	
Flujo alternativo	
2a. El administrador cancela la acción. 4a. El administrador decide no proseguir y el panel de control regresa a 1.	

Caso de uso editar un proyecto del servidor Atlas.

Caso de uso	Editar un proyecto del servidor Atlas.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.
Pos condiciones	Los datos sobre el proyecto han sido modificados.

Flujo básico
1. El panel de control presenta una lista de los proyectos existentes. 2. El administrador selecciona el proyecto que desea editar y selecciona la opción editar. 3. El panel de control presenta los datos, nombre y sistema espacial de referencia. 4. El administrador modifica el nombre del proyecto. 5. El administrador selecciona la opción seleccionar sistema espacial de referencia y pasa al caso de uso seleccionar sistema espacial de referencia. 6. El panel de control verifica que los dos campos estén presentes y guarda los cambios.
Flujo alternativo
2a. El administrador cancela la operación. 5a. Si el proyecto ya tiene orígenes de datos el sistema espacial de referencia se presenta solo para lectura. 6a. Si los datos del proyecto no son correctos, el sistema informa al administrador y regresa a paso 3.

Caso de uso seleccionar sistema espacial de referencia.

Caso de uso	Seleccionar sistema espacial de referencia.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha iniciado el caso de uso, agregar un proyecto al servidor Atlas.
Pos condiciones	Se informa el sistema seleccionado por el usuario.
Flujo básico	
1. El panel de control presenta un árbol con los sistemas de referencia espacial disponibles, clasificados en, geográficos y proyectados, y clasificando los proyectados según el tipo del proyección. 2. El administrador selecciona el sistema de su preferencia y selecciona la opción continuar.	
Flujo alternativo	
2a. El administrador cancela la acción sin seleccionar ningún sistema.	

Caso de uso agregar un origen de datos a un proyecto.

Caso de uso	Agregar un origen de datos a un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.

Pos condiciones	Un origen de datos se ha agregado al proyecto.
Flujo básico	
1. El panel de control presenta un listado con los proyectos existentes. 2. El administrador selecciona el proyecto al que desea adicionar el origen. 3. El administrador selecciona el tipo de origen de que desea agregar, si el origen es ShapeFile se pasa al caso de uso agregar ShapeFile, si el origen es Postgis se pasa al caso de uso agregar desde Postigis.	
Flujo alternativo	
2a. El administrador cancela la acción.	

Caso de uso agregar un origen de datos ShapeFile.

Caso de uso	Agregar un origen de datos ShapeFile.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha iniciado el caso de uso agregar un origen de datos a un proyecto.
Pos condiciones	Se ha agregado un origen de datos al proyecto.
Flujo básico	
1. El panel de control presenta el cuadro de diálogo de abrir archivos, mostrando solo archivos shp. 2. El administrador selecciona el archivo que desea agregar. 3. El panel de control presenta un listado de los campos que contiene el archivo. 4. El administrador selecciona los campos que desea agregar y continúa. 5. El origen de datos se convierte al formato del sistema y se agrega.	
Flujo alternativo	
2a. El administrador cancela la acción y termina el caso. 3a. Si el ShapeFile no es válido o está corrupto el panel de control informa el error y termina el caso de uso. 4a. El administrador cancela la acción y termina el caso. 4b. Si el administrador no selecciona ningún campo, el panel de control informa que se debe seleccionar al menos un campo para importar y regresa a 3.	

Caso de uso agregar un origen de datos desde postgis.

Caso de uso	Agregar un origen de datos desde postgis.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha iniciado el caso de uso agregar un origen de datos a un proyecto.
Pos condiciones	Se ha agregado un origen de datos al proyecto.

Flujo básico
1. El panel de control presenta un listado de las tablas postgis válidas que encuentre en la base de datos del panel de control. 2. El administrador selecciona la tabla que desea agregar. 3. El panel de control verifica que la tabla tenga una llave primaria de tipo numérico y que no haya sido agregada aun. 4. El origen de datos se agrega al proyecto.
Flujo alternativo
1a. Si postgis no está instalado en la base de datos, el panel de control informa al administrador y termina el caso. 2b. El administrador cancela la acción. 3a. Si la tabla seleccionada no cumple con las condiciones requeridas o ya se encuentra asignada al proyecto, el panel de control informa al administrador y regresa a 1.

Caso de uso remover orígenes de datos a un proyecto.

Caso de uso	Remover orígenes de datos a un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.
Pos condiciones	El origen de datos ha sido removido de un proyecto.
Flujo básico	
1. El panel de control presenta un listado de los orígenes de datos del proyecto. 2. El administrador selecciona el origen que desea eliminar. 3. El panel de control verifica que el origen no se encuentre en uso por alguna capa. 4. El origen de datos se elimina del proyecto.	
Flujo alternativo	
1a. Si el proyecto no tiene orígenes instalados el listado se presenta vacío. 2a. El administrador abandona sin seleccionar ningún origen. 3a. Si el origen se encuentra en uso, el panel de control informa al administrador y regresa a 1.	

Caso de uso agregar un geocoder a proyecto.

Caso de uso	Agregar un geocoder a proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.

Pos condiciones	El geocoder se ha adicionado al proyecto.
Flujo básico	
1. El panel de control presenta un listado con los proyectos existentes. 2. El panel de control presenta un listado con los geocoders disponibles. 3. El administrador selecciona un proyecto y el geocoder que desea agregar. 4. El administrador selecciona la opción agregar. 5. El geocoder se ha agregado al proyecto.	
Flujo alternativo	
1a. Si el servidor no tiene algún proyecto el caso termina. 2a. Si el panel de control no tiene algún geocoder instalado, el listado se presenta vacío. 3a. El administrador abandona el proceso. 4a. Si el panel de control encuentra que el geocoder ya está presente en el proyecto, informa al usuario y regresa a 2.	

Caso de uso remover un geocoder de un proyecto.

Caso de uso	Remover un geocoder de un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.
Pos condiciones	El geocoder se ha removido del proyecto.
Flujo básico	
1. El panel de control presenta un listado con los geocoders existentes en el proyecto. 2. El administrador selecciona la opción eliminar. 3. El panel de control presenta un diálogo de confirmación para la acción. 4. El usuario confirma la acción. 5. El geocoder se ha eliminado del proyecto.	
Flujo alternativo	
1a. Si el sistema no tiene algún proyecto, el caso termina. 1b. Si el proyecto no tiene algún geocoder, el caso termina. 2a. El administrador abandona el proceso. 4a. Si el administrador decide no proseguir, el panel de control no realiza cambios y regresa a 1.	

Caso de uso configurar un geocoder para un proyecto.

Caso de uso	Configurar un geocoder para un proyecto.
Actor principal	Administrador.

Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor Atlas.
Pos condiciones	Se ha cambiado la configuración del un geocoder para un proyecto.
Flujo básico	
1. El panel de control presenta un listado con los geocoders existentes en el proyecto. 2. El administrador selecciona la opción configurar. 3. El panel de control presenta el diálogo de configuración del geocoder. 4. El panel de control toma la nueva configuración.	
Flujo alternativo	
1a. Si el sistema no tiene algún proyecto, el caso termina. 1b. Si el proyecto no tiene algún geocoder, el caso termina. 2a. El administrador abandona el proceso.	

Caso de uso agregar una capa a un proyecto.

Caso de uso	Agregar una capa a un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Una capa nueva se agrega al proyecto.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El administrador selecciona la opción agregar capa. 3. El panel de control presenta un formulario para agregar una capa con los campos, titulo, resumen, denominadores de escala, palabras clave, atribución y consultable, cuyos significados y valores son establecidos por el estándar WMS. 4. El administrador ingresa los campos del formulario y continúa. 5. El panel de control valida los campos requeridos y agrega la nueva capa.	
Flujo alternativo	
4a. El administrador abandona el formulario. 5a. Si los campos requeridos no están presentes o su contenido no es válido, el panel de control informa al administrador para que este corrija el error, regresando a 3.	

Caso de uso editar una capa de un proyecto.

Caso de uso	Editar una capa de un proyecto.
Actor principal	Administrador.

Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Se realiza la edición de una capa.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción editar capa. 4. El panel de control presenta el formulario de la capa con los datos de titulo, resumen, denominadores de escala, palabras clave, atribución y consultable ya cargados. 5. El administrador edita los campos del formulario y continúa. 6. El panel de control valida los campos requeridos y actualiza los datos de la capa.	
Flujo alternativo	
5a. El administrador abandona el formulario. 6a. Si los campos requeridos no están presentes o su contenido no es válido, el panel de control informa al usuario para que este corrija el error, regresando a 4.	

Caso de uso eliminar una capa de un proyecto.

Caso de uso	Eliminar una capa de un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Una capa es eliminada del proyecto.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción eliminar capa. 4. El panel de control pide una confirmación de la acción. 5. El administrador confirma la acción. 6. El panel de control elimina la capa de manera permanente.	
Flujo alternativo	
3a. El administrador abandona el formulario. 5a. El administrador no confirma la acción de eliminar, cancelando la acción.	

Caso de uso cambiar el orden de las capas de un proyecto.

Caso de uso	Cambiar el orden de las capas de un proyecto.
Actor principal	Administrador.

Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Cambia el orden de visualización de las capas del proyecto.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción cambiar orden. 4. El panel de control verifica si la capa puede cambiar a la posición indicada. 5. Cambia la posición de visualización de la capa, ordenando las capas restantes. 6. Si el administrador está satisfecho con el orden de la capas puede terminar la operación, si lo desea puede regresar a 3.	
Flujo alternativo	
4a. Si la capa no puede cambiar a la posición indicada en control panel informa al administrador que la operación no puede realizarse.	

Caso de uso asignar un origen de datos a una capa.

Caso de uso	Asignar un origen de datos a una capa.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Un origen de datos se asigna a una capa.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción asignar origen de datos. 4. El panel de control muestra un formulario con un listado de los orígenes de datos disponibles en el proyecto. 5. El administrador selecciona el origen de datos para la capa y continúa. 6. El panel de control asigna el origen de datos a la capa.	
Flujo alternativo	
5a. El administrador no selecciona ningún origen de datos para la capa y decide continuar, caso en que la capa permanece sin origen. 5b. El administrador abandona el formulario sin realizar cambios.	

Caso de uso asignar etiquetado a una capa.

Caso de uso	Asignar etiquetado a una capa.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Se asigna el etiquetado a una capa.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción etiquetar capa. 4. El panel de control muestra un formulario para asignar etiquetas a la capa, con los campos de fuente, estilo, tamaño, prioridad y campo de la capa a etiquetar. 5. El administrador completa los campos del formulario y continúa. 6. El panel de control configura la capa para que se visualice con el etiquetado indicado.	
Flujo alternativo	
5a. Si los campos requeridos no están presentes o su contenido no es válido, el panel de control informa al administrador para que este corrija el error, regresando a 4. 5b. El administrador abandona el formulario.	

Caso de uso editar la simbología de una capa.

Caso de uso	Editar la simbología de una capa.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un servidor Atlas.
Pos condiciones	Se edita la simbología de una capa.
Flujo básico	
1. El administrador selecciona un proyecto. 2. El panel de control lista las capas del proyecto seleccionado. 3. El administrador selecciona la capa y selecciona la opción editar simbología. 4. El panel de control muestra un formulario permitiendo el cambio de simbologías fijas, de clase o por rangos. 5. El administrador selecciona el tipo de simbología que desea editar. 6. Se llama al caso de uso editar simbología fija, editar simbología por clases o editar simbología por rangos, dependiendo de la selección del administrador.	
Flujo alternativo	
4a. El administrador abandona el formulario.	

Caso de uso editar simbología fija de una capa.

Caso de uso	Editar simbología fija de una capa.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asigna una simbología fija a una capa.
Flujo básico	
1. El panel de control muestra el formulario para asignar simbología fija. 2. El administrador selecciona la opción generar simbología. 3. El panel de control asigna una regla para la capa. 4. El panel de control, pasa al caso de uso editar regla para una capa tipo punto, editar regla para una capa tipo línea, editar regla para una capa tipo polígono dependiendo del tipo de geometría de la capa.	
Flujo alternativo	
2a. El administrador abandona el formulario.	

Caso de uso editar simbología por clase de una capa.

Caso de uso	Editar simbología por clases de una capa.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asigna una simbología por clase a una capa.
Flujo básico	
1. El panel de control muestra un formulario para configurar la simbología por clase con los atributos de la capa. 2. El administrador selecciona el atributo de la capa que desee para formar las clases de la simbología. 3. El administrador selecciona la opción generar simbología. 4. El panel de control genera tantas reglas como número de clases contiene la capa y las presenta. 5. El administrador selecciona una regla para su edición. 6. Se pasa al caso de uso editar regla para una capa tipo punto, editar regla para una capa tipo línea, editar regla para una capa tipo polígono dependiendo del tipo de geometría de la capa. 7. Si el administrador está satisfecho con la configuración de la reglas termina el caso, si no es así, puede regresar a 5.	
Flujo alternativo	
3a. El administrador abandona el formulario sin hacer cambios.	

Caso de uso editar simbología por rangos de una capa.

Caso de uso	Editar simbología por rangos de una capa.
Actor principal	Administrador.
Otros actores	Atlas Server.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asigna una simbología por rangos a una capa.
Flujo básico	
1. El panel de control muestra un formulario para configurar la simbología por rangos con los campos, color inicial, color final, los atributos numéricos de la capa y el número de rangos a generar. 2. El administrador completa los datos para generar la simbología por rangos. 3. El administrador selecciona la opción generar simbología. 4. El panel de control genera tantas reglas como número de rangos y las presenta. 5. El administrador selecciona una regla para su edición. 6. Se pasa al caso de uso editar regla para una capa tipo punto, editar regla para una capa tipo línea, editar regla para una capa tipo polígono dependiendo del tipo de geometría de la capa. 7. Si el administrador está satisfecho con la configuración de la reglas termina el caso, si no es así, puede regresar a 5.	
Flujo alternativo	
3a. El administrador abandona el formulario.	

Caso de uso editar regla para una capa tipo punto.

Caso de uso	Editar una regla para una capa tipo punto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asignan la regla de dibujo a una capa tipo punto.
Flujo básico	
1. El panel de control muestra un formulario con las opciones disponibles para una regla de una capa tipo punto, color de línea, color de relleno, tamaño y un listado de los iconos del sistema. 2. El administrador edita la regla de la capa, modificando cada uno de los campos y continúa. 3. El panel de control configura la capa y simbología seleccionada por el administrador.	
Flujo alternativo	
2a. El administrador abandona el formulario.	

Caso de uso editar regla para una capa tipo línea.

Caso de uso	Editar una regla para una capa tipo línea.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asignan la regla de dibujo a una capa tipo línea.
Flujo básico	
1. El panel de control muestra un formulario con las opciones disponibles para una regla de una capa tipo línea, estilo, color, opacidad y ancho. 2. El administrador edita la regla de la capa, modificando cada uno de los campos y continúa. 3. El panel de control configura la capa y simbología seleccionada por el administrador.	
Flujo alternativo	
2a. El administrador abandona el formulario.	

Caso de uso editar regla para una capa tipo polígono.

Caso de uso	Editar una regla para una capa tipo polígono.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	Caso de uso editar la simbología de una capa.
Pos condiciones	Se asignan la regla de dibujo a una capa polígono.
Flujo básico	
1. El panel de control muestra un formulario con las opciones disponibles para una regla de una capa tipo polígono, estilo de línea, color de línea, opacidad, ancho de la línea y color de fondo. 2. El administrador edita la regla de la capa, modificando cada uno de los campos y continúa. 3. El panel de control configura la capa y simbología seleccionada por el administrador.	
Flujo alternativo	
2a. El administrador abandona el formulario	

Caso de uso visualizar la cartografía del proyecto.

Caso de uso	Visualizar la cartografía del proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha establecido conexión con un Servidor.

Flujo básico
1. El panel de control presenta una lista de los proyectos disponibles en el servidor. 2. El administrador selecciona el proyecto que desea visualizar. 3. El panel de control presenta el mapa compuesto por la visualización de las capas del proyecto, que son solicitadas al servidor Atlas vía HTTP. 4. El administrador puede realizar operaciones sobre el mapa como, acercarse o alejarse y desplazarse, así como consultar al geocoder del proyecto llamando al caso de uso Consultar el geocoder de un proyecto u obtener información sobre algún punto del mapa llamando al caso de uso consultar información GetFeatureInfo.
Flujo alternativo
1a. Si el sistema no tiene algún proyecto, el caso termina.3a. Si el proyecto no tiene alguna capa con origen de datos, el caso termina.

Caso de uso consultar información GetFeatureInfo.

Caso de uso	Consultar información GetFeatureInfo.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha llamado al caso de uso visualizar la cartografía del proyecto.
Pos condiciones	El administrador ha recibido información sobre los rasgos existentes en un punto del mapa.
Flujo básico	
1. El administrador elije el punto en píxeles que desea consultar. 2. El administrador elije el formato en que desea recibir la información HTML o GML. 3. El panel de control compone y envía la petición al Servidor Atlas vía HTTP. 4. El panel de control muestra el resultado al administrador.	
Flujo alternativo	
4a. Si se produce algún error en el envío, el panel de control informa al administrador y termina el caso.	

Caso de uso consultar el geocoder de un proyecto.

Caso de uso	Consultar el geocoder de un proyecto.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Precondiciones	El administrador ha llamado al caso de uso visualizar la cartografía del proyecto.

Pos condiciones	El administrador ha recibido información sobre la ubicación geográfica asociada a la descripción textual de un lugar.
Flujo básico	
El administrador ingresa la descripción textual que desea consultar. El panel de control compone y envía la petición al servidor Atlas vía HTTP. El panel de control muestra el resultado al administrador.	
Flujo alternativo	
4a. Si se produce algún error en el envío, el panel de control informa al administrador y termina el caso.	

Caso de uso establecer las opciones de configuración.

Caso de uso	Establecer las opciones de configuración.
Actor principal	Administrador.
Pos condiciones	Las opciones del panel de control han sido modificadas.
Flujo básico	
1. El administrador puede seleccionar el idioma que desea para la interfaz de usuario, los idiomas disponibles son inglés y español. 2. El administrador puede seleccionar la apariencia que desea para la aplicación, las opciones disponibles son, apariencia del sistema operativo y apariencia por defecto de Java. 3. El administrador puede agregar y remover plugins de geocodificación llamando a los casos de uso, agregar un plugin de geocoder al panel de control y eliminar un plugin de geocoder del panel de control. 4. Si el administrador está satisfecho con los cambios, selecciona continuar. 5. El panel de control informa al usuario que debe reiniciar la aplicación para que los cambios tengan efecto y los guarda.	
Flujo alternativo	
4a. El administrador cancela la acción sin aplicar los cambios.	

Caso de uso agregar un plugin de geocoder al panel de control.

Caso de uso	Agregar un plugin de geocoder al panel de control.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Pos condiciones	Un plugin de geocoder se ha agregado al panel de control.
Flujo básico	
1. El panel de control presenta el cuadro de diálogo de abrir archivo, mostrando únicamente archivos JAR. 2. El administrador selecciona el archivo que contiene el plugin. 3. El panel de control encuentra que el JAR es un plugin de geocoder Atlas.	

4. El panel de control agrega el plugin.
Flujo alternativo
1a. El administrador cancela la acción. 3a. Si el panel de control encuentra que el archivo JAR seleccionado no es un plugin valido, informa al usuario y termina el caso.

Caso de uso eliminar un plugin de geocoder del panel de control.

Caso de uso	Eliminar un plugin de geocoder del panel de control.
Actor principal	Administrador.
Otros actores	Servidor Atlas.
Pos condiciones	Un plugin de geocoder se ha removido al panel de control.
Flujo básico	
1. El panel de control un listado con los plugins ya instalados. 2. El administrador selecciona el plugin que desea remover. 3. El panel de control presenta un diálogo de confirmación de la acción. 4. El administrador decide continuar. 5. El panel de control remueve el plugin.	
Flujo alternativo	
2a. El administrador cancela la acción. 4a. Si administrador decide no continuar el caso termina sin realizar cambios.	

6.3.3. Casos de uso del Web Archive. El Web Archive es una herramienta que permite la preparación del archivo WAR que contiene los componentes de servidor. Este paso es indispensable para la instalación del sistema.

Caso de uso generar un archivo WAR de servidor Atlas.

Caso de uso	Generar un archivo WAR de servidor Atlas.
Actor principal	Administrador.
Otros actores	Web Archive.
Pos condiciones	Se ha generado un archivo WAR listo para ser instalado en el contendor de servlets.
Flujo básico	
1. El Web Archive llama al caso de uso establecer propiedades de seguridad y despliegue de la aplicación. 2. El Web Archive llama al caso de uso establecer propiedades de conexión con la base de datos. 3. El Web Archive llama al caso de uso establecer los plugins de geocodificación. 4. El Web Archive presenta el diálogo de guardar archivo.	

5. El usuario selecciona la ubicación donde desea crear el archivo. 6. El Web Archive crea el archivo y lo copia en la ruta indicada.
Flujo alternativo
2a. El administrador abandona el proceso.

Caso de uso establecer propiedades de seguridad y despliegue de la aplicación.

Caso de uso	Establecer propiedades de seguridad y despliegue de la aplicación.
Actor principal	Administrador.
Precondiciones	El caso de uso generar un archivo WAR de servidor Atlas.
Pos condiciones	Se han establecido las propiedades de seguridad y despliegue de la aplicación.
Flujo básico	
1. El Web Archive presenta los campos de, ruta de despliegue, nombre de usuario, contraseña y tamaño en megabytes del caché de imágenes. 2. El administrador llena los datos y continúa. 3. El Web Archive se asegura de que los campos contengan valores correctos, los campos de nombre de usuario, contraseña y ruta de despliegue deben ser combinaciones de letras mayúsculas y minúsculas y números, el tamaño del caché debe ser un número entero.	
Flujo alternativo	
2a. El administrador abandona el proceso. 3a. Si alguno de los campos está incorrecto el Web Archive informa al administrador para que este corrija los errores y regresa a 1.	

Caso de uso establecer propiedades de conexión con la base de datos.

Caso de uso	Establecer propiedades de conexión con la base de datos.
Actor principal	Administrador.
Otros actores	Web Archive.
Precondiciones	El caso de uso generar un archivo WAR de servidor Atlas.
Pos condiciones	Se han establecido las propiedades de conexión con la BD.
Flujo básico	
1. El Web Archive presenta los campos de dirección IP del servidor de base de datos, nombre de la base de datos, nombre de usuario y contraseña en el gestor. 2. El administrador llena los campos y continúa. 3. El Web Archive se asegura de que los campos contengan valores correctos, los campos de nombre de usuario, contraseña deben ser combinaciones de letras mayúsculas y minúsculas y números, la dirección del servidor debe ser una IP válida.	

Flujo alternativo
2a. El administrador abandona el proceso. 2b. El administrador pasa al caso de uso propiedades de seguridad y despliegue de la aplicación. 3a. Si alguno de los campos está incorrecto el Web Archive informa al administrador para que este corrija los errores y regresa a 1.

Caso de uso establecer los plugins de geocodificación.

Caso de uso	Establecer los plugins de geocodificación.
Actor principal	Administrador.
Otros actores	Web Archive.
Precondiciones	El caso de uso generar un archivo WAR de servidor Atlas.
Pos condiciones	Se han establecido los plugins de geocodificación.
Flujo básico	
1. El Web Archive presenta una lista, inicialmente vacía con los plugins que se van a incluir en el archivo WAR. 2. El administrador puede seleccionar la opción agregar plugin, y el Web Archive presenta el cuadro de diálogo de abrir archivo, mostrando únicamente archivos JAR. 3. El administrador selecciona el archivo que contiene el plugin. 4. El Web Archive encuentra que el JAR es un plugin de geocoder Atlas, lo agrega a las lista y regresa a 1. 5. El administrador puede seleccionar un plugin de la lista y removerlo con la opción eliminar. 6. Si el administrador está conforme con la lista de plugins puede seleccionar la opción continuar y pasar al caso de uso generar el Archivo WAR según lo establecido, de lo contrario puede regresar a 2 o a 5.	
Flujo alternativo	
2a. El administrador puede continuar sin agregar plugins y pasar a 6. 5a. El administrador puede continuar sin remover plugins. 6a. El administrador puede abandonar el proceso.	

6.3.4. Casos de uso para los componentes de desarrollo. Los componentes de desarrollo permiten la construcción de aplicaciones que trabajan con el servidor Atlas, para obtener información de él y combinarla con datos propios del domino de tal aplicación, a continuación se presentan sus casos de uso.

Caso de uso establecer conexión con un servidor Atlas.

Caso de uso	Establecer conexión con un servidor Atlas.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Pos condiciones	Se ha establecido conexión con un servidor Atlas.
Flujo básico	
1. La aplicación indica la URL de servidor Atlas con el que desea trabajar. 2. Los componentes de desarrollo se conectan a la dirección indicada en la URL vía HTTP y establecen que se trata de un servidor Atlas.	
Flujo alternativo	
2a. Si es imposible establecer una conexión HTTP con la URL indicada, se informa a la aplicación. 2b. Si es imposible hallar un servidor Atlas en la URL indicada, se informa a la aplicación.	

Caso de uso retornar listado de proyectos en el servidor.

Caso de uso	Retornar listado de proyectos en el servidor.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer conexión con un servidor Atlas.
Pos condiciones	Se ha recuperado un listado de los proyectos disponibles.
Flujo básico	
1. La aplicación indica que desea obtener un listado de proyectos disponibles en el servidor. 2. Los componentes de desarrollo establecen conexión con el servidor Atlas y solicitan información sobre los proyectos disponibles. 3. Los componentes de desarrollo procesan la respuesta del servidor y entregan el listado.	
Flujo alternativo	
2a. Si no es posible establecer una conexión con el servidor Atlas, se informa a la aplicación sobre el error.	

Caso de uso retornar listado de capas por proyecto.

Caso de uso	Retornar listado de capas por proyecto.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer conexión con un servidor Atlas. Retornar listado de proyectos en el servidor.

Pos condiciones	Se ha retornado un listado con las capas que contiene un proyecto en particular.
Flujo básico	
1. La aplicación indica el proyecto del que desea obtener información, este proyecto debe pertenecer al listado de proyectos del servidor. 2. Los componentes de desarrollo consultan el listado al servidor Atlas. 3. Los componentes de desarrollo procesan la respuesta del servidor y entregan el listado.	
Flujo alternativo	
2a. Si no es posible recuperar el listado, los componentes informan del error.	

Caso de uso establecer el proyecto y las capas para visualización.

Caso de uso	Establecer el proyecto y las capas para visualización.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer conexión con un servidor Atlas. Retornar listado de proyectos en el servidor. Retornar listado de capas por proyecto.
Pos condiciones	Se ha establecido el proyecto y las capas que se desea visualizar.
Flujo básico	
1. La aplicación indica el proyecto y las capas que desea visualizar. 2. Los componentes establecen estos datos.	

Caso de uso establecer la región visible.

Caso de uso	Establecer la región visible.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización.
Pos condiciones	Se cambiado la región visible de mapa.
Flujo básico	
1. La aplicación indica la región que desea visualizar en coordenadas del mapa. 2. Los componentes de desarrollo fijan estos datos. 3. Se llama al caso de uso visualizar mapa.	

Caso de uso visualizar mapa.

Caso de uso	Visualizar mapa.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización. Establecer la región visible.
Pos condiciones	Se visualiza un mapa de acuerdo a las opciones establecidas para los componentes.
Flujo básico	
1. Los componentes toman el proyecto actual, las capas y la región visible, y determinan las imágenes necesarias para formar un mapa con estas características. 2. Los componentes solicitan estas imágenes al servidor Atlas mediante peticiones GetMap. 3. Los componentes unen las imágenes y presentan el mapa.	
Flujo alternativo	
2a. Si no es posible establecer una conexión con el servidor Atlas, se informa a la aplicación sobre el error.	

Caso de uso agregar un marcador.

Caso de uso	Agregar un marcador.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización.
Pos condiciones	Se ha agregado un marcador al mapa actual.
Flujo básico	
1. La aplicación indica las coordenadas de mapa donde debe ubicarse en marcador. 2. Los componentes de desarrollo fijan estos datos. 3. Se llama al caso de uso visualizar mapa.	

Caso de uso remover un marcador.

Caso de uso	Remover un marcador.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización.
Pos condiciones	Se ha agregado un marcador al mapa actual.
Flujo básico	
1. La aplicación indica el marcador que desea remover.	

2. Los componentes de desarrollo remueven el marcador. 3. Se llama al caso de uso visualizar mapa.

Caso de uso retornar la ubicación de una dirección textual.

Caso de uso	Retornar la ubicación de una dirección textual.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización.
Pos condiciones	Se ha entregado la ubicación en el mapa de la dirección textual indicada.
Flujo básico	
1. La aplicación indica la descripción textual del lugar que desea conocer. 2. Los componentes de desarrollo envían la solicitud al servidor Atlas. 3. Los componentes de desarrollo procesan la respuesta del servidor Atlas. 4. Los componentes de desarrollo retornan las posibles localizaciones asociadas a la ubicación indicada.	
Flujo alternativo	
3a. Si el servidor Atlas no logra establecer ubicaciones para la descripción indicada, los componentes de desarrollo informan de la situación.	

Caso de uso retornar información mediante GetFeatureInfo.

Caso de uso	Retornar información mediante GetFeatureInfo.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización.
Pos condiciones	Se ha entregado la información GetFeatureInfo correspondiente al punto indicado.
Flujo básico	
1. La aplicación indica las coordenadas en píxeles del lugar del que desea recibir información. 2. La aplicación indica el formato en que desea recibir la información, pudiendo escoger entre HTML, GML y texto plano. 3. Los componentes de desarrollo usan la información de la región visible y las coordenadas para crear una petición que envían al servidor Atlas. 4. Los componentes de desarrollo retornan la información entregada por el servidor.	
Flujo alternativo	
3a. Si los componentes no logran establecer comunicación con el servidor informan de la situación.	

Caso de uso interactuar con el mapa.

Caso de uso	Interactuar con el mapa.
Actor principal	Aplicación.
Otros actores	Componentes de desarrollo Atlas.
Precondiciones	Establecer el proyecto y las capas para visualización. Establecer la región visible.
Flujo básico	
1. La aplicación cambia el área visible, e interactúa con los marcadores que se hayan agregado mediante acciones de mouse, como hacer clic y arrastrar el puntero. 2. Los componentes informan cuando se ha cambiado la visualización por este medio e informa si se ha seleccionado algún marcador. 3. Se llama a visualizar mapa cada vez que el área visible es modificada.	

6.3.5. Casos de uso el servidor Atlas. El servidor Atlas es una aplicación J2EE basada en servlets que se encarga de contestar las peticiones HTTP realizadas por los clientes. Sus casos de uso se presentan a continuación.

Caso de uso validar el acceso.

Caso de uso	Validar el acceso.
Actor principal	Panel de control.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se informa la coincidencia de los datos.
Flujo básico	
1. El panel de control envía el nombre y la contraseña del sistema. 2. El servidor verifica si el nombre y contraseña coinciden con los de su configuración. 3. El servidor informa si los datos coinciden o no.	
Flujo alternativo	
2a. Si los datos están incompletos o no coinciden el servidor informa el error.	

Caso de uso reportar conexión con la base de datos.

Caso de uso	Reportar conexión con la base de datos.
Actor principal	Panel de control.
Otros actores	Servidor Atlas. Gestor de base de datos.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se informa si la conexión se puede realizar.

Flujo básico
1. El panel de control solicita al servidor verificar la conexión con la base de datos. 2. El servidor verifica su conexión de acuerdo a su configuración. 3. El servidor informa que la conexión se realizo con éxito.
Flujo alternativo
3a. El servidor informa que la conexión no se pudo realizar.

Caso de uso enviar datos de conexión con la base de datos.

Caso de uso	Enviar datos de conexión con la base de datos.
Actor principal	Panel de control.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Enviar los datos de la conexión.
Flujo básico	
1. El panel de control solicita los datos de conexión de la base de datos y envía los datos de acceso. 2. Pasa al caso de uso validar el acceso. 3. Pasa al caso de uso reportar conexión con la base de datos. 4. Envía los datos de la conexión de la base de datos.	
Flujo alternativo	
2a. Informa un error al validar los datos de acceso. 4a. Informa un error al reportar la conexión con la base de datos.	

Caso de uso reiniciar el servidor.

Caso de uso	Reiniciar el servidor.
Actor principal	Panel de control.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	El servidor se reinicia.
Flujo básico	
1. El panel de control solicita reiniciar el servidor y envía los datos de acceso. 2. Pasa al caso de uso validar el acceso. 3. Pasa al caso de uso reportar conexión con la base de datos. 4. El servidor se reinicia.	
Flujo alternativo	
2a. Informa un error al validar los datos de acceso. 3a. Informa un error al reportar la conexión con la base de datos.	

Caso de uso enviar información sobre el mapa según GetFeatureInfo.

Caso de uso	Enviar información sobre el mapa según GetFeatureInfo.
Actor principal	Cliente.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se informa los resultados de la operación GetFeatureInfo.
Flujo básico	
1. El cliente solicita información de un mapa según GetFeatureInfo, con los datos establecidos en el estándar WMS. 2. El servidor valida los datos de solicitud. 3. El servidor genera los resultados de la solicitud. 4. El servidor da formato de presentación a los resultados. 5. El servidor informa los resultados.	
Flujo alternativo	
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error. 3a. La solicitud no genera resultados.	

Caso de uso enviar información sobre un CRS.

Caso de uso	Enviar información sobre un CRS.
Actor principal	Cliente.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se informa los resultados sobre el CRS.
Flujo básico	
1. El cliente solicita información sobre el CRS con el código EPSG del sistema de referencia. 2. El servidor verifica el código del sistema de referencia. 3. El servidor genera los resultados de la solicitud. 4. El servidor informa los datos del CRS en XML.	
Flujo alternativo	
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error.	

Caso de uso enviar un mapa según GetMap.

Caso de uso	Enviar un mapa según GetMap.
Actor principal	Cliente.
Otros actores	Servidor Atlas.

Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se envía una imagen del mapa de conformidad con el estándar WMS petición GetMap.
Flujo básico	
1. El cliente solicita una imagen del mapa de conformidad con la petición GetMap, con los datos establecidos en el estándar WMS. 2. El servidor verifica los datos de la solicitud. 3. El servidor genera la imagen del mapa en el formato especificado. 4. El servidor envía la imagen generada.	
Flujo alternativo	
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error.	

Caso de uso enviar la leyenda de una capa.

Caso de uso	Enviar la leyenda de una capa.
Actor principal	Cliente.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se envía una imagen con la leyenda de la capa.
Flujo básico	
1. El cliente solicita la leyenda de una capa, con los datos código de proyecto, código de capa. 2. El servidor verifica si los datos de la solicitud son correctos. 3. El servidor genera una imagen con la leyenda de la capa. 4. El servidor envía la imagen generada.	
Flujo alternativo	
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error.	

Caso de uso enviar declaración de recursos GetCapabilities.

Caso de uso	Enviar declaración de recursos GetCapabilities.
Actor principal	Cliente.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se envía un documento de conformidad con el estándar WMS petición GetCapabilities.
Flujo básico	
1. El cliente solicita la declaración de los recursos del servidor, de conformidad con	

la petición GetCapabilities, con los datos establecidos en el estándar WMS. 2. El servidor verifica los datos de la solicitud. 3. Pasa al caso de uso generar documento XML. 4. El servidor envía el documento.
Flujo alternativo
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error.

Caso de uso generar documento XML.

Caso de uso	Generar documento XML.
Actor principal	Servidor Atlas
Precondiciones	Caso de uso enviar declaración de recursos GetCapabilities.
Pos condiciones	Se genera un documento XML de conformidad con el estándar WMS
Flujo básico	
1. Se genera un XML con las características del servidor de conformidad con el estándar WMS como por ejemplo, formatos de disponibles de imágenes, URL para realizar las peticiones. 2. Se obtiene un listado de todos los proyectos disponibles. 3. Se obtiene las capas de cada proyecto listado. 4. Se agrega al XML todos los campos requeridos de conformidad con el estándar WMS, por cada una de las capas.	
Flujo alternativo	
2a. Se genera un documento especificando que el servidor no contiene información.	

Caso de uso enviar documento JSON.

Caso de uso	Enviar documento JSON.
Actor principal	Cliente.
Otros actores	Servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se envía un documento formato JSON con los recursos disponibles del servidor.
Flujo básico	
1. El cliente solicita los recursos del servidor en formato JSON. 2. El servidor genera el documento en formato JSON. 3. El servidor envía el documento generado.	

Caso de uso enviar dirección geocodificada.

Caso de uso	Enviar dirección geocodificada.
Actor principal	Cliente.
Otros actores	Geocoder, servidor Atlas.
Precondiciones	El servidor Atlas en funcionamiento.
Pos condiciones	Se envía el resultado de una dirección geocodificada.
Flujo básico	
1. El cliente solicita geocodificar una dirección textual, con los campos dirección, formato y código del proyecto. 2. El servidor verifica los datos de la solicitud. 3. El servidor inicia el proceso para asignar la geocodificación de la dirección a los geocoders disponibles en el proyecto solicitado. 4. Cada geocoder intenta el proceso de geocodificación para la dirección. 5. Cada geocoder entrega los resultados obtenidos. 6. El servidor crea un documento con los resultados y lo envía.	
Flujo alternativo	
2a. Si los datos de la solicitud están incompletos o equivocados el servidor informa el error. 3a. El servidor informa que no existen geocoders disponibles. 5a. El geocoder no genera resultados para la dirección especificada.	

6.3.6. Diagramas de casos de uso del panel de control.

En las figuras 56 y 57 se muestran los diagramas de casos de uso correspondientes al panel de control, para más detalles acerca de los casos de uso vaya al numeral 6.3.2 casos de uso del panel de control.

Figura 56. Diagrama A de casos de uso del panel de control.

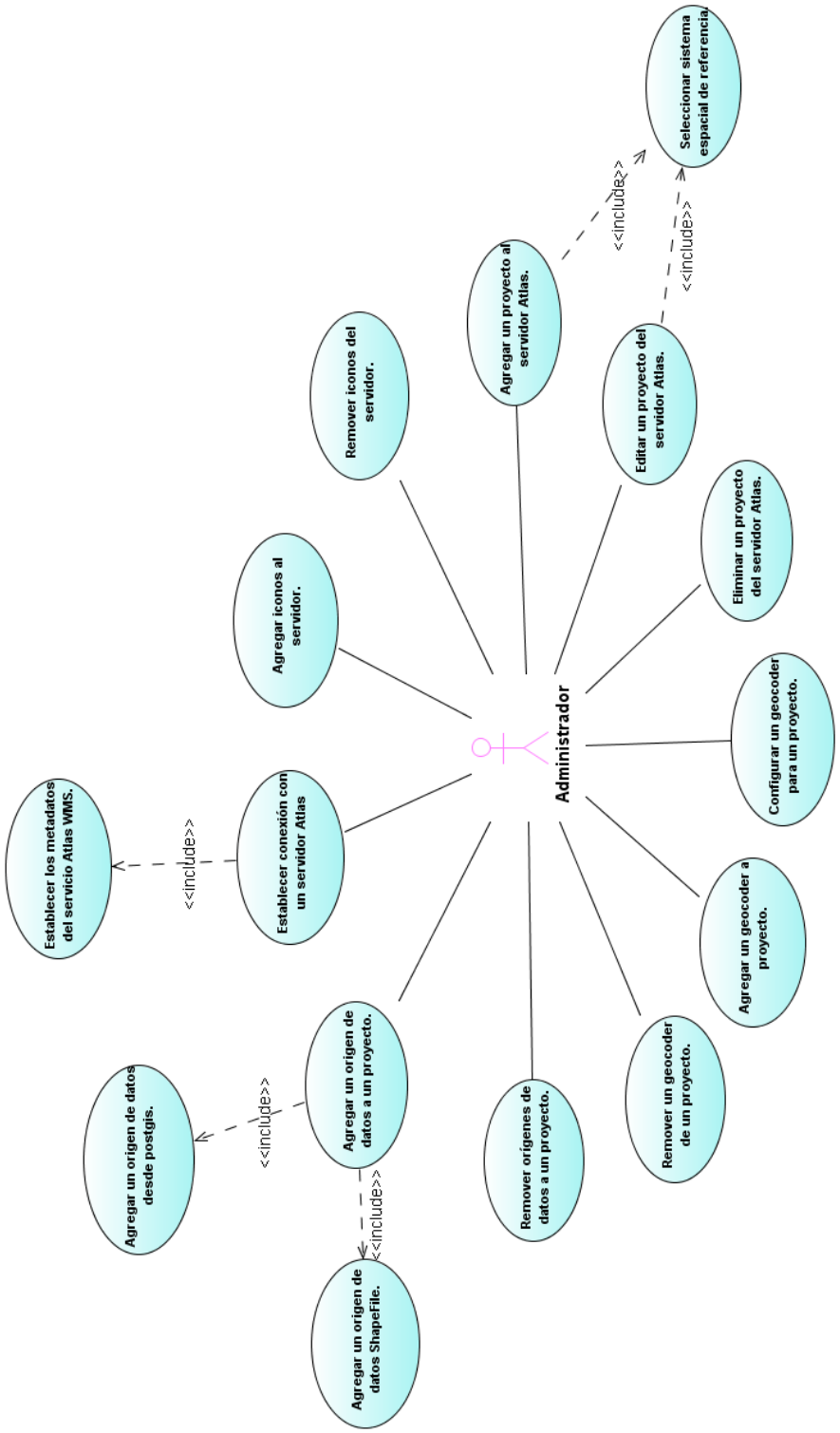
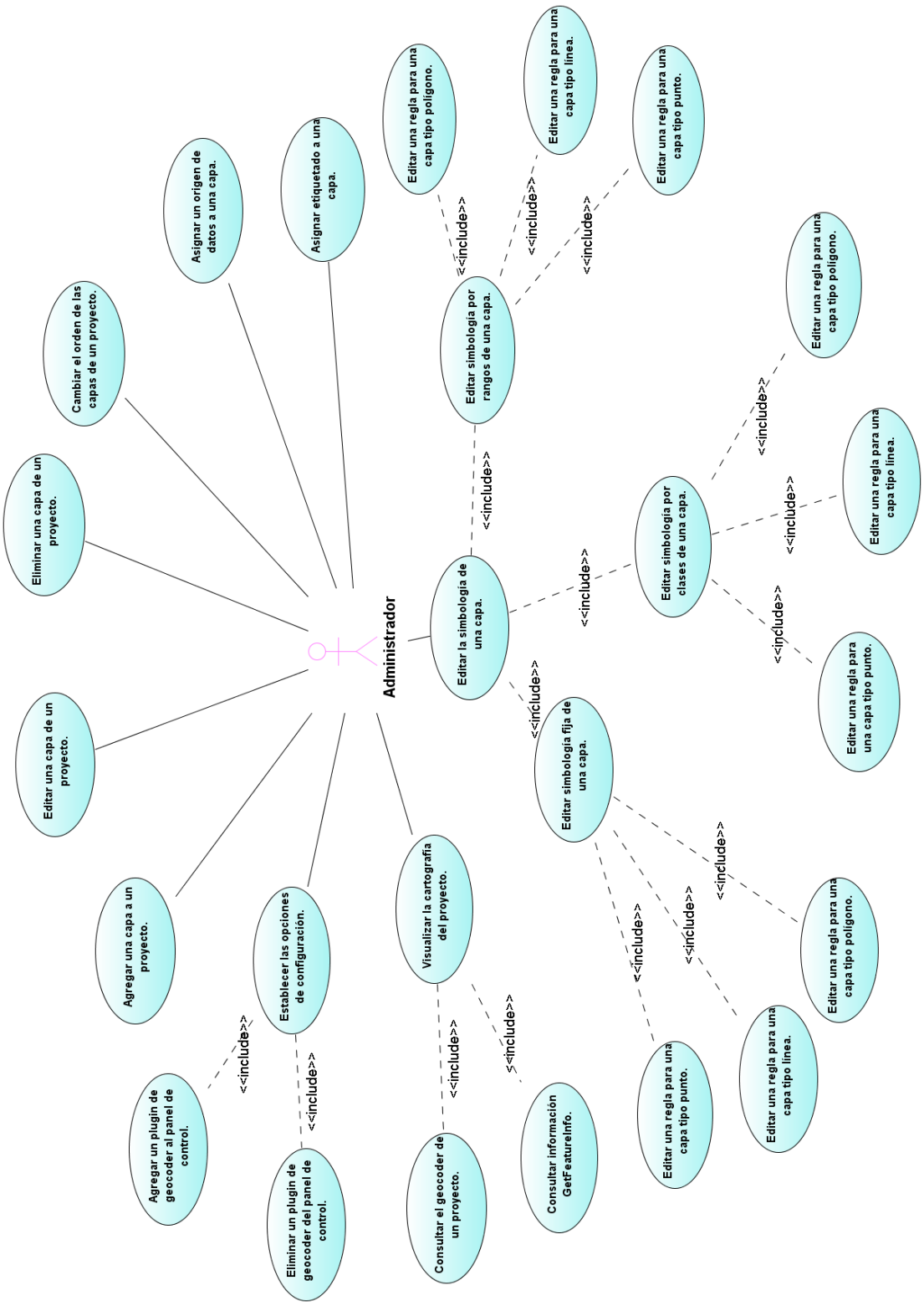


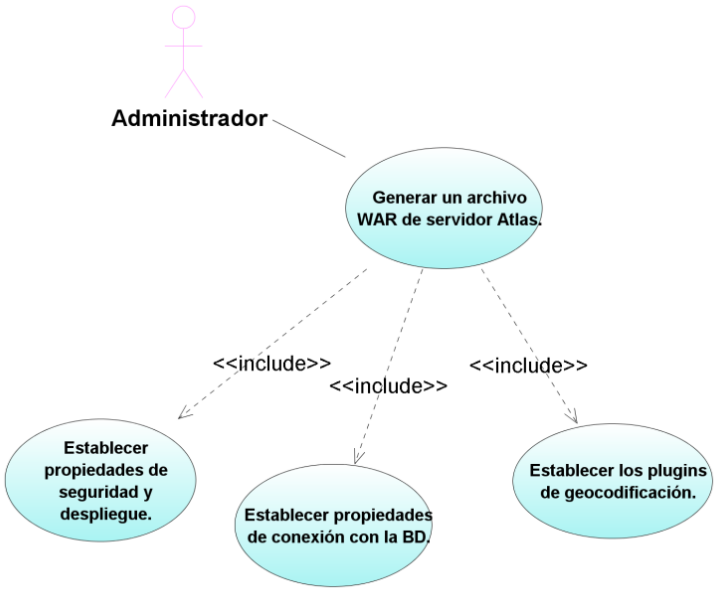
Figura 57. Diagrama B de casos de uso del panel de control.



6.3.7. Diagramas de casos de uso del Web Archive.

La figura 58 muestra el diagrama de casos de uso correspondientes al Web Archive, para más detalles acerca de los casos de uso vaya al numeral 6.3.3 casos de uso del Web Archive.

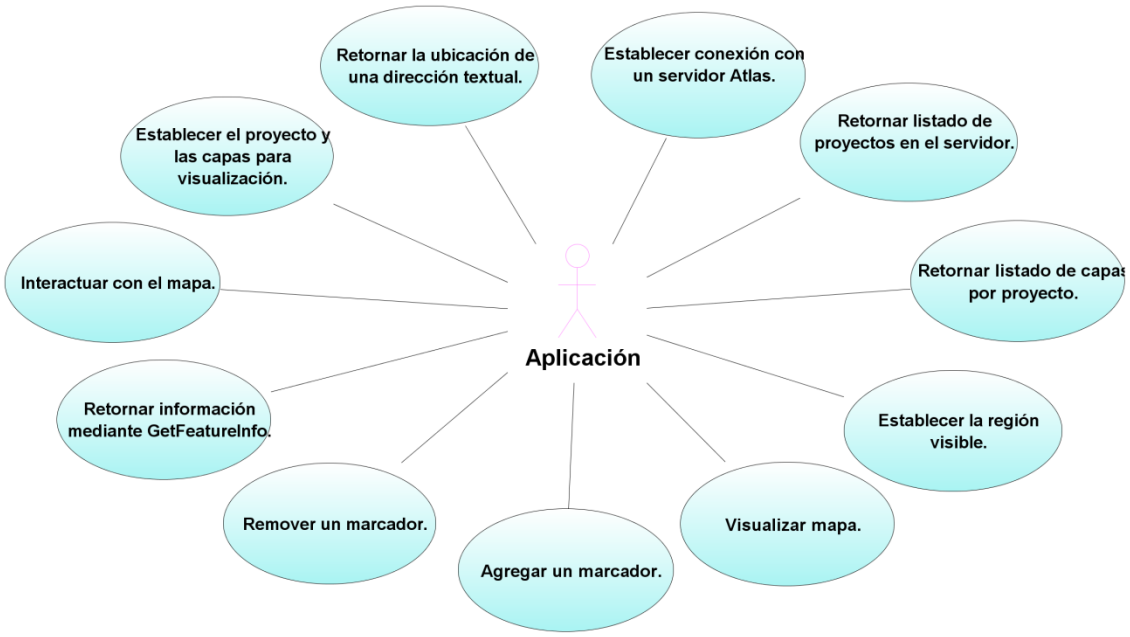
Figura 58. Diagrama general de casos de uso del Web archive.



6.3.8. Diagramas de casos de los componentes de desarrollo Atlas.

La figura 59 muestra el diagrama de casos de uso correspondientes a los componente de desarrollo, para más detalles acerca de los casos de uso vaya al numeral 6.3.4 casos de uso de los componentes de desarrollo.

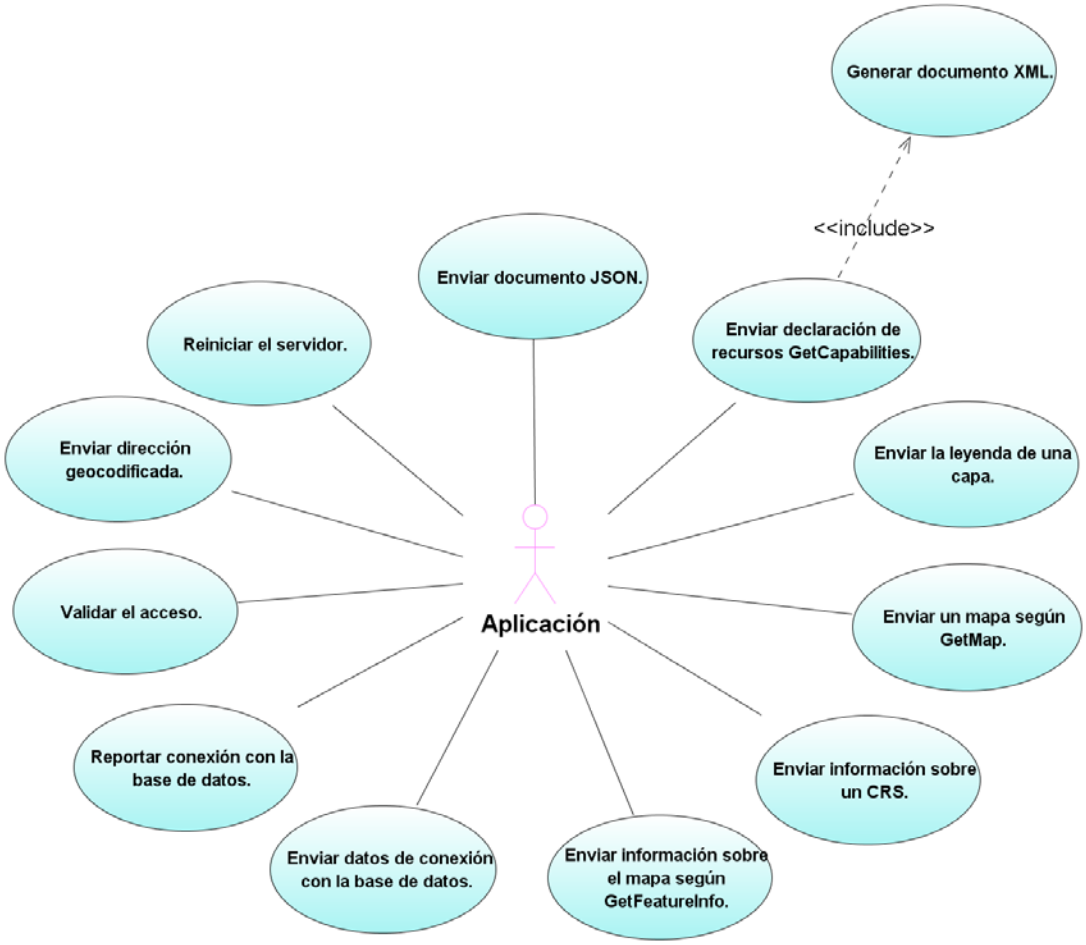
Figura 59. Diagrama general de casos de uso de los componentes de desarrollo.



6.3.9. Diagramas de casos del servidor Atlas.

La figura 60 muestra el diagrama de casos de uso correspondientes al servidor Atlas, para más detalles acerca de los casos de uso vaya al numeral 6.3.5 casos de uso del servidor Atlas.

Figura 60. Diagrama general de casos de uso del servidor Atlas.



6.3.10. Diagramas de secuencia de sistema del panel de control.

Las figuras 61, 62 y 63 muestran los diagramas de secuencia del sistema del panel de control.

Figura 61. Diagrama de secuencia del sistema, conexión con un servidor.

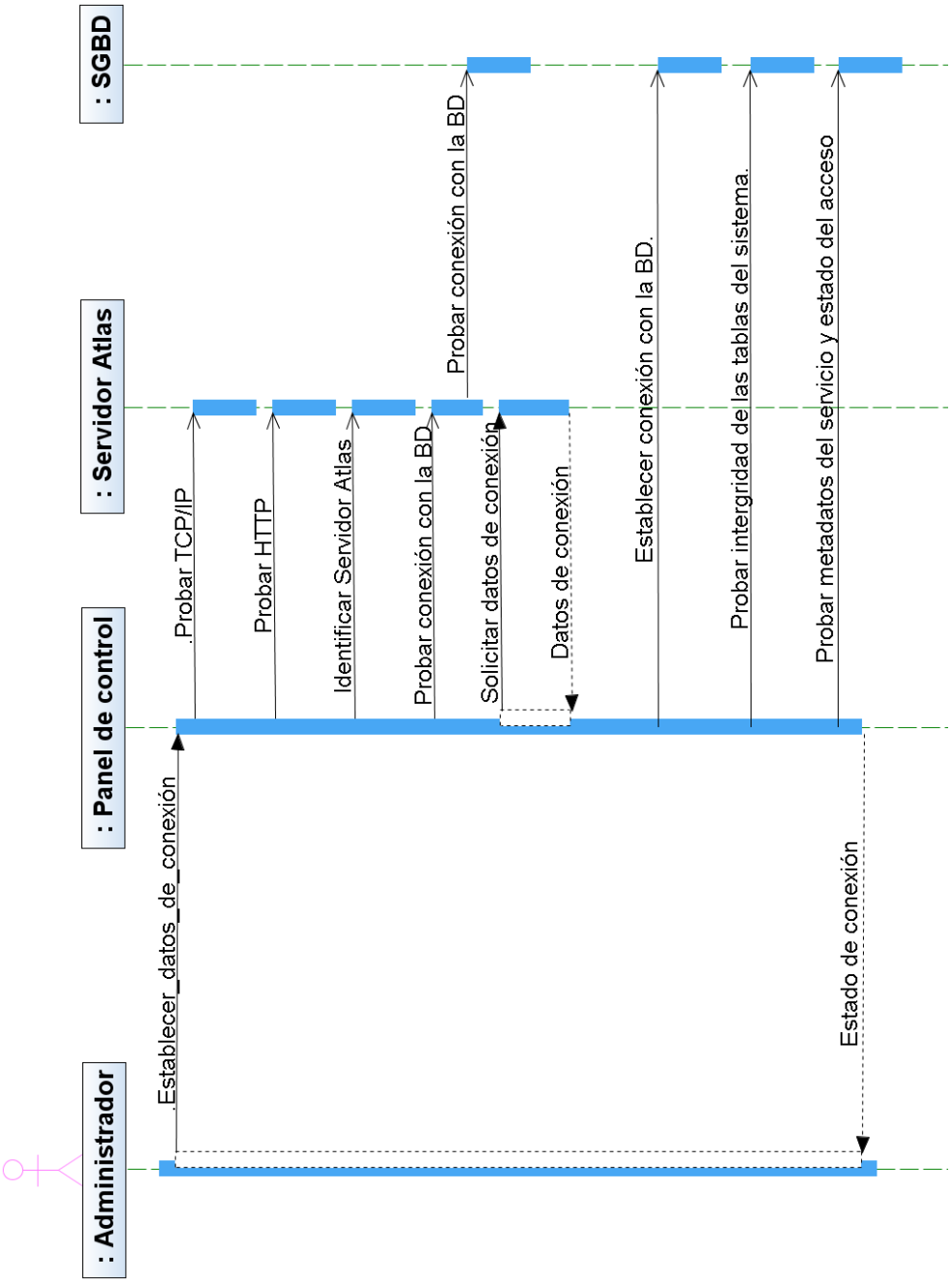


Figura 62. Diagrama de secuencia del sistema, editar simbología por clases.

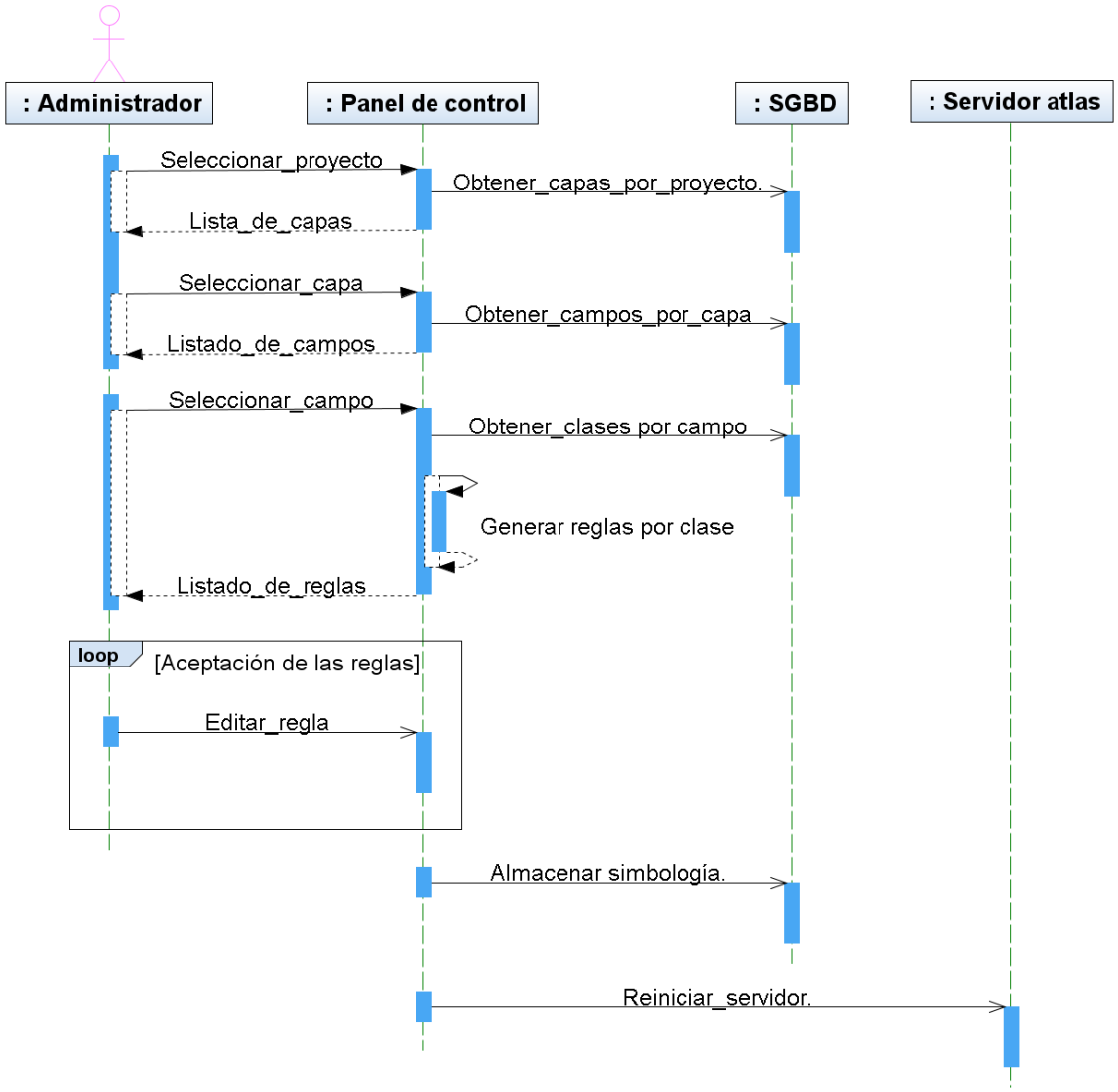
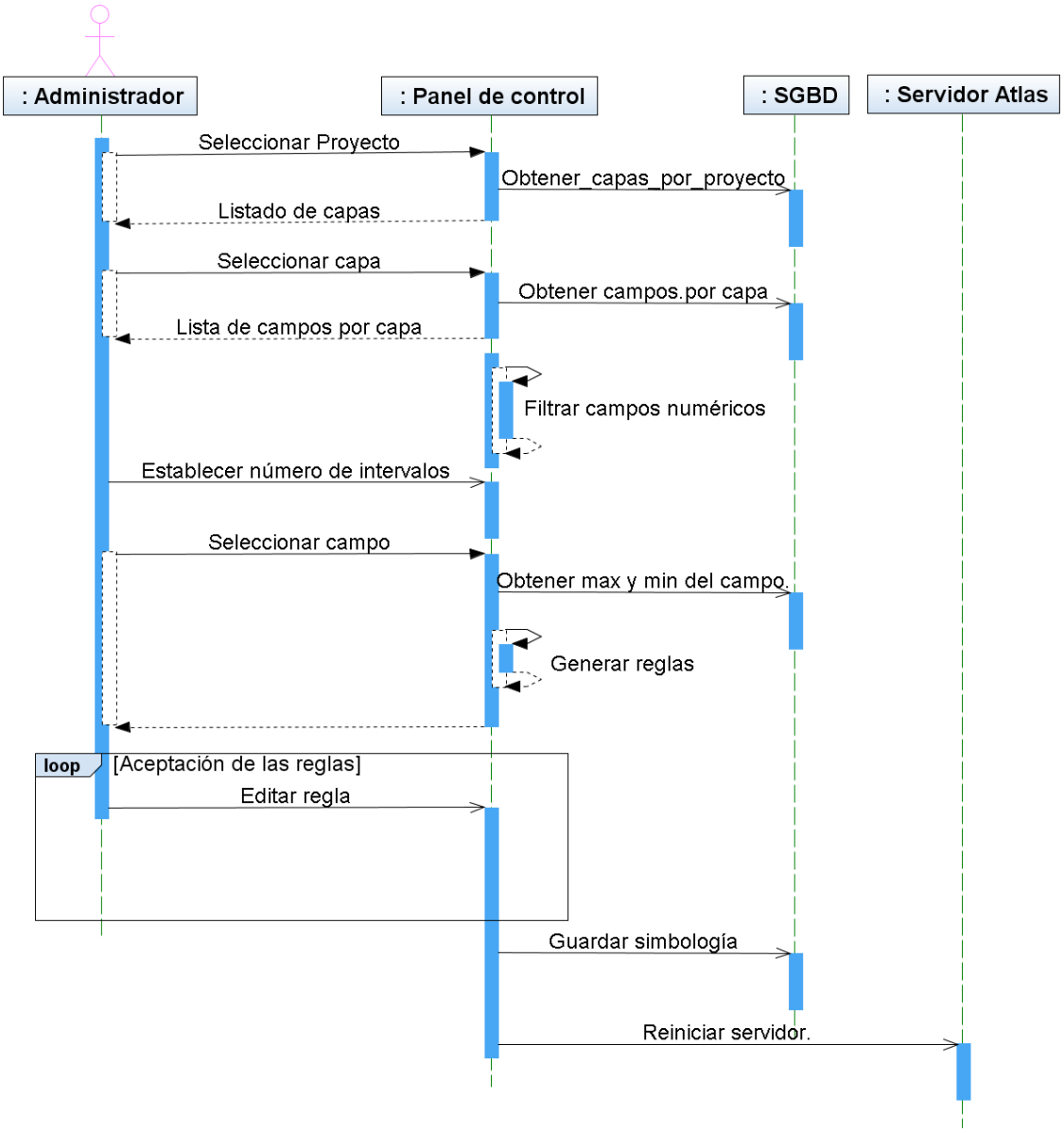


Figura 63. Diagrama de secuencia del sistema, editar simbología por rangos.



6.3.11. Diagramas de secuencia de sistema del servidor Atlas.

Las figuras 64, 65, 66 y 67 muestran los diagramas de secuencia del sistema del servidor Atlas.

Figura 64. Diagrama de secuencia del sistema, petición GetCapabilities.

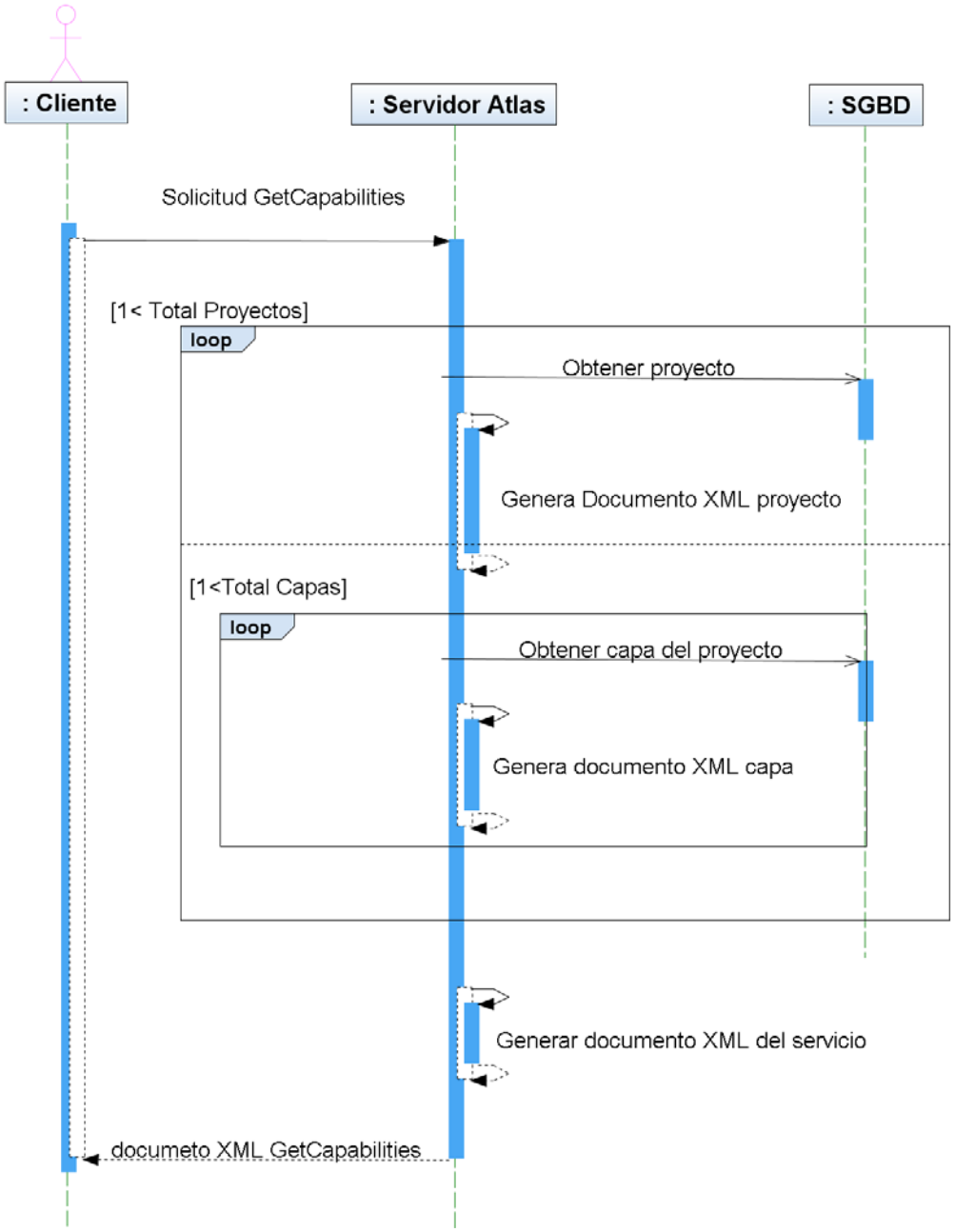


Figura 65. Diagrama de secuencia del sistema, petición GetMap.

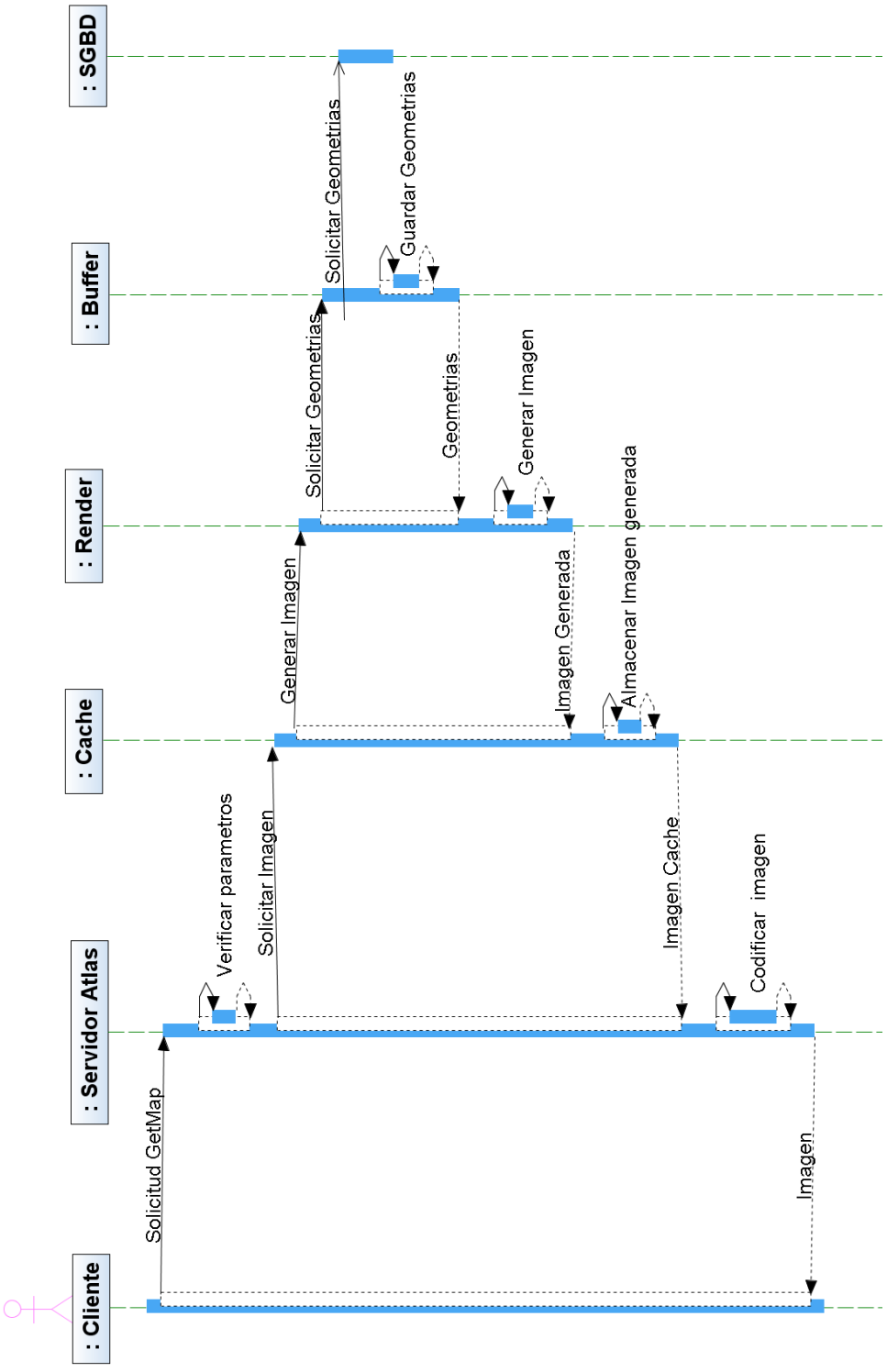


Figura 66. Diagrama de secuencia del sistema, petición GetFeatureInfo.

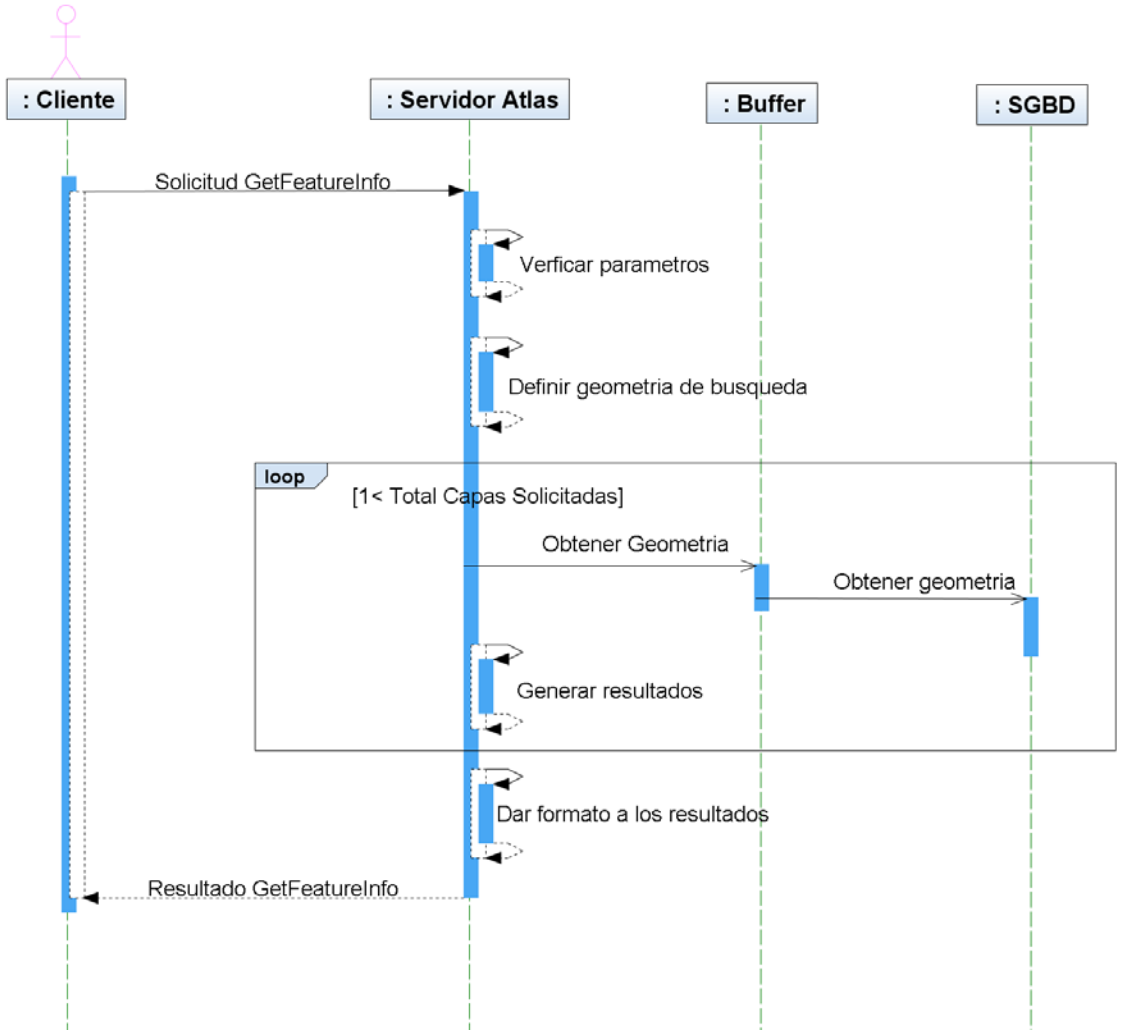
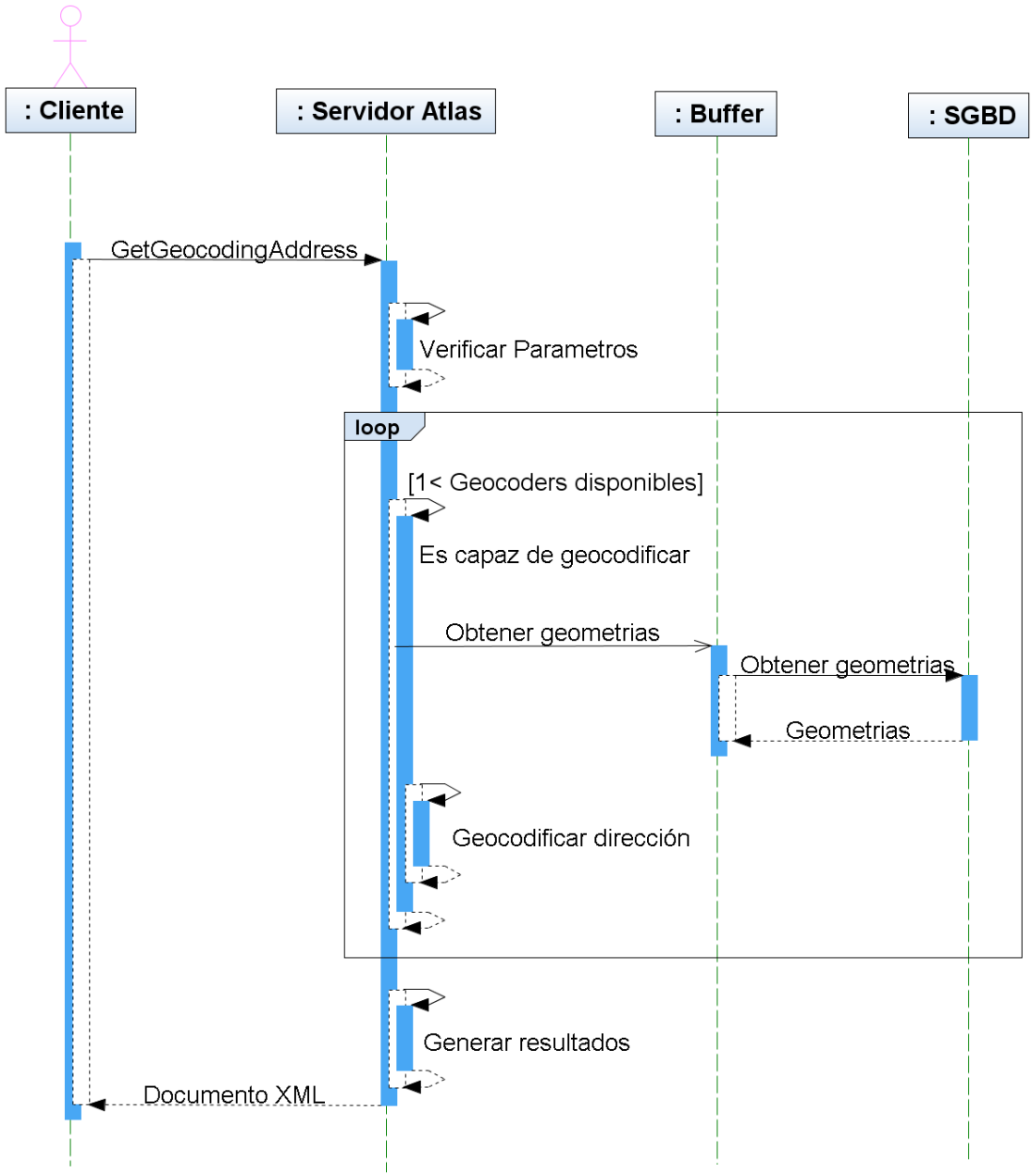


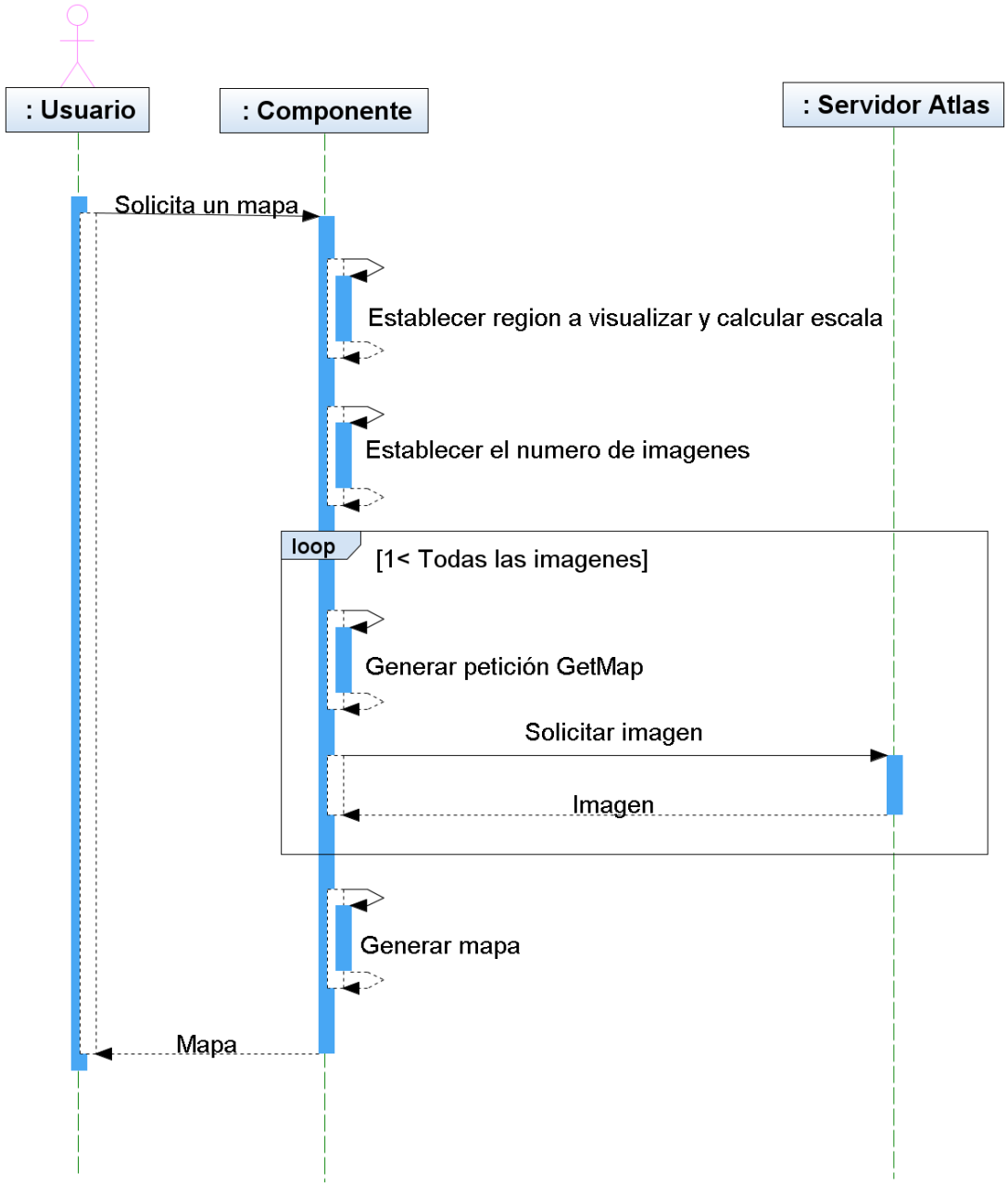
Figura 67. Diagrama de secuencia del sistema, petición geocoder.



6.3.12. Diagramas de secuencia de sistema de los componentes de desarrollo.

Las figura 68 muestran el diagrama de secuencia de sistema de los componentes de desarrollo para la visualización de un mapa.

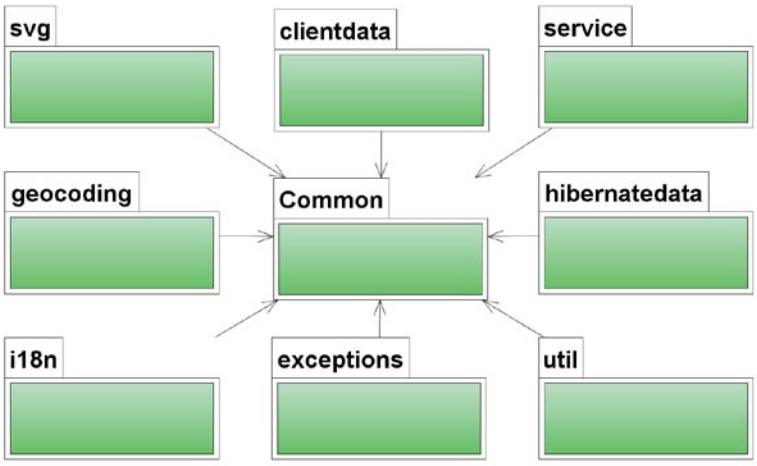
Figura 68. Diagrama de secuencia del sistema, visualizar mapa.



6.3.13. Diagramas de paquetes common.

La figura 69 muestra el diagrama de paquetes common.

Figura 69. Paquetes del módulo common.



6.3.14. Diagramas de paquetes del panel de control.

La figura 70 muestra el diagrama general del panel de control, la figura 71 muestran el diagrama de paquetes GUI del panel de control.

Figura 70. Paquetes del panel de control.

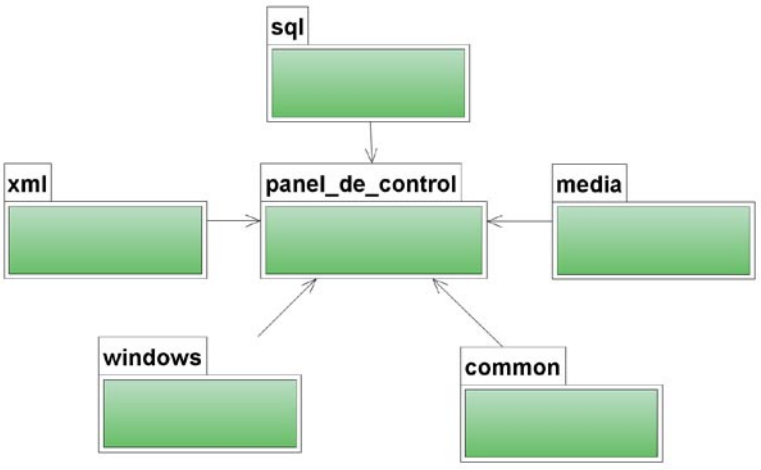
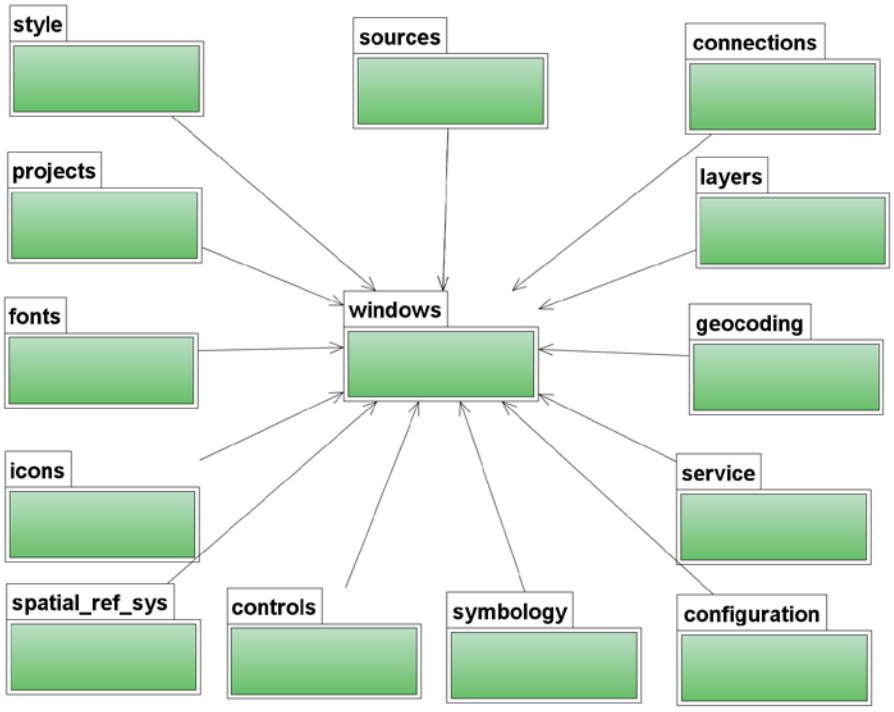


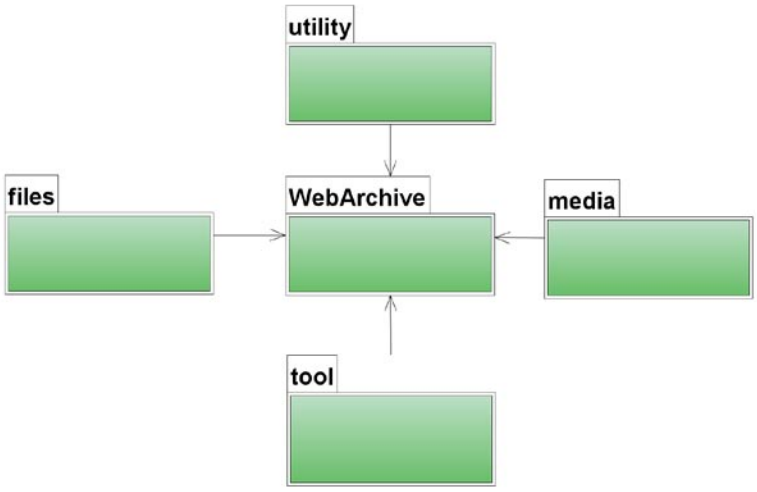
Figura 71. Paquete GUI en el panel de control.



6.3.15. Diagramas de paquetes del Web Archive.

La figura 72 muestra el diagrama de paquetes del Web Archive

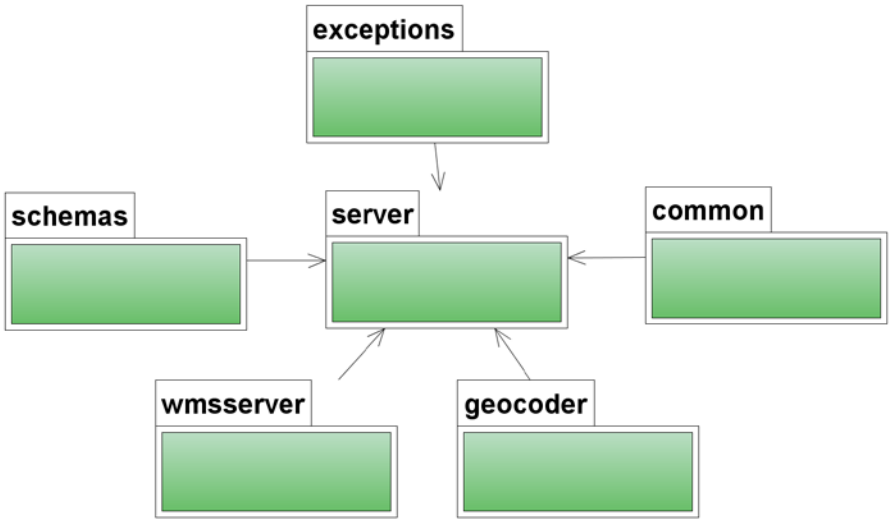
Figura 72. Paquetes del Web Archive.



6.3.16. Diagramas de paquetes del servidor.

La figura 73 muestra el diagrama de paquetes del servidor Atlas.

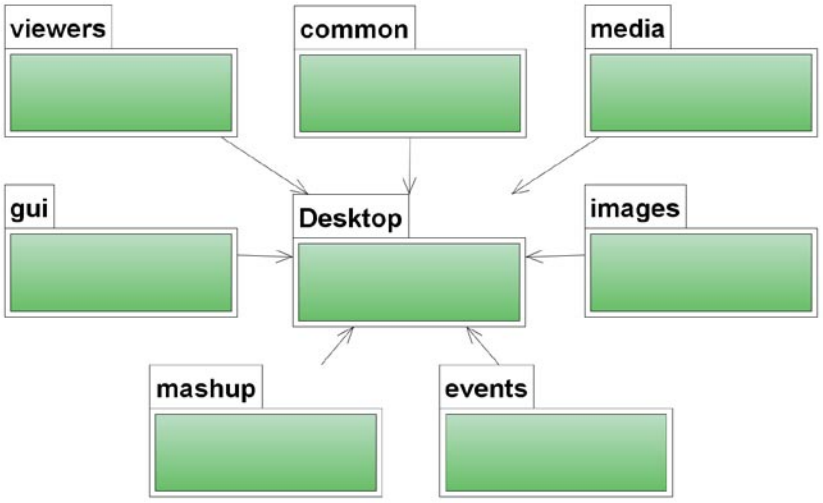
Figura 73. Paquetes del servidor.



6.3.17. Diagramas de paquetes, biblioteca para desarrollo Java SE.

Los figura 74 muestra el diagrama de paquetes de la biblioteca para desarrollo de aplicaciones JAVA SE.

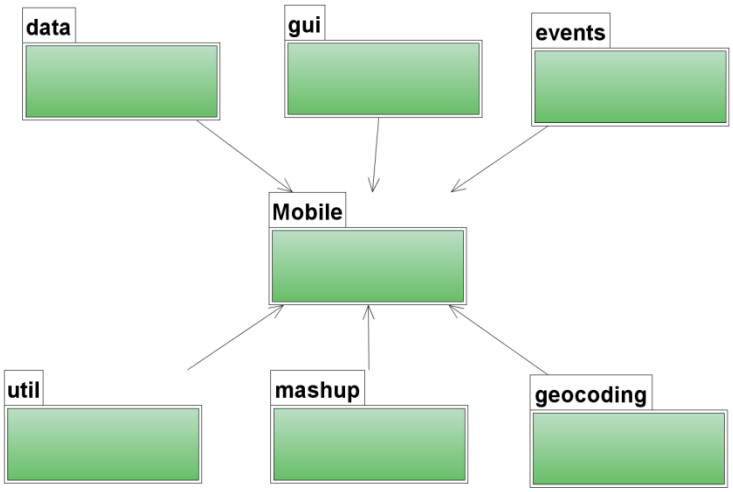
Figura 74. Paquetes de la biblioteca para desarrollo Java SE.



6.3.18. Diagramas de paquetes, biblioteca para desarrollo Java ME.

Los figura 75 muestra el diagrama de paquetes de la biblioteca para desarrollo de aplicaciones JAVA ME.

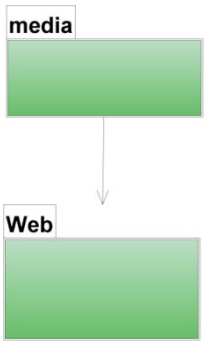
Figura 75. Paquetes de la biblioteca para desarrollo Java ME.



6.3.19. Diagramas de paquetes, biblioteca para desarrollo Web 2.0.

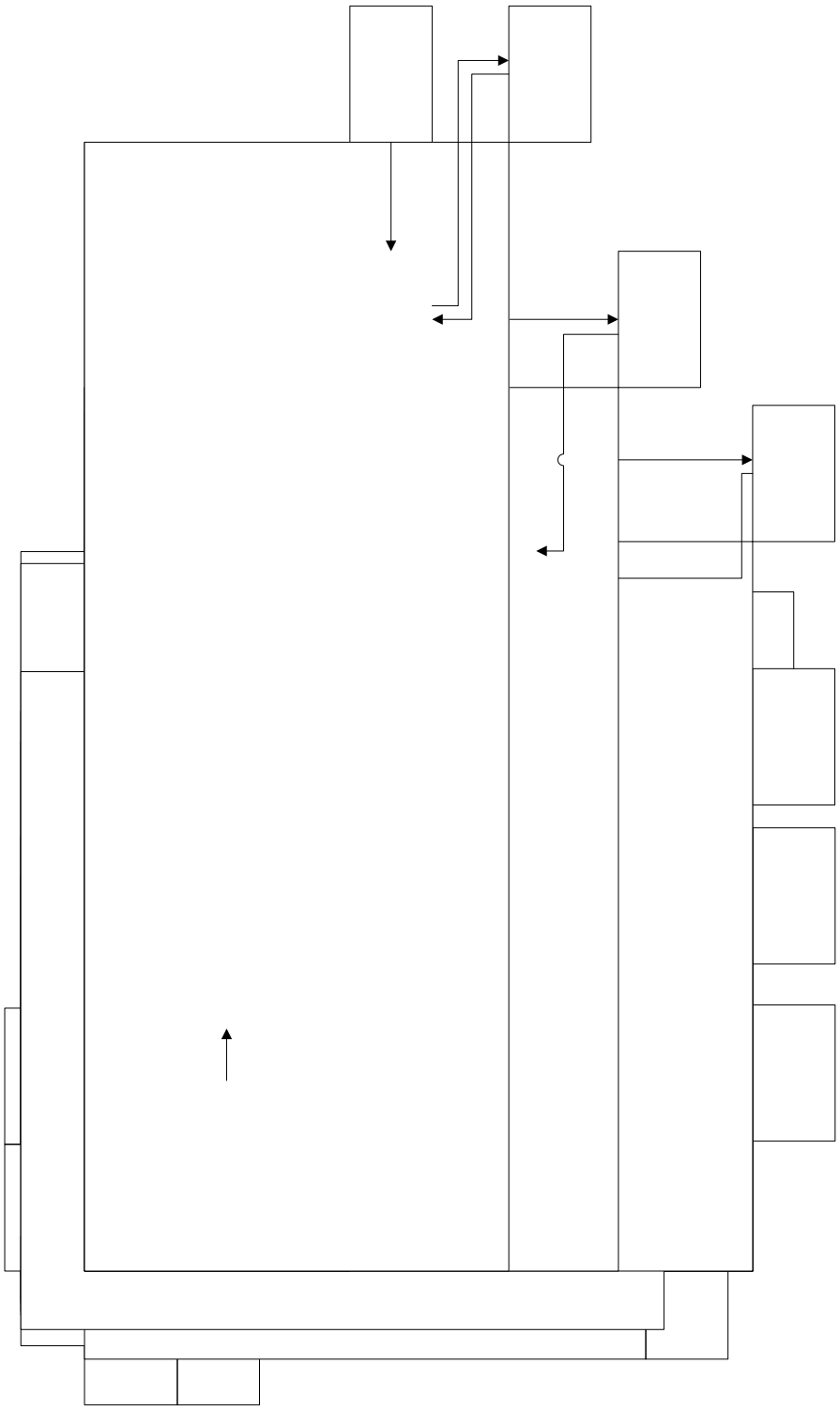
La figura 76 muestra el diagrama de paquetes de la biblioteca para desarrollo de aplicaciones Web 2.0 con javascript.

Figura 76. Paquetes de la biblioteca para desarrollo de aplicaciones Web.



La figura 77 muestra el diagrama de secuencia de interfaz del panel de control.

Figura 77. Diagrama de secuencia de interfaz del panel de control de ATLAS.



6.3.20. Diagramas de clases del dominio.

Las figuras 78, 79, 80 y 81 representan los diagramas de clases de dominio.

Figura 78. Diagrama A de clases del dominio.

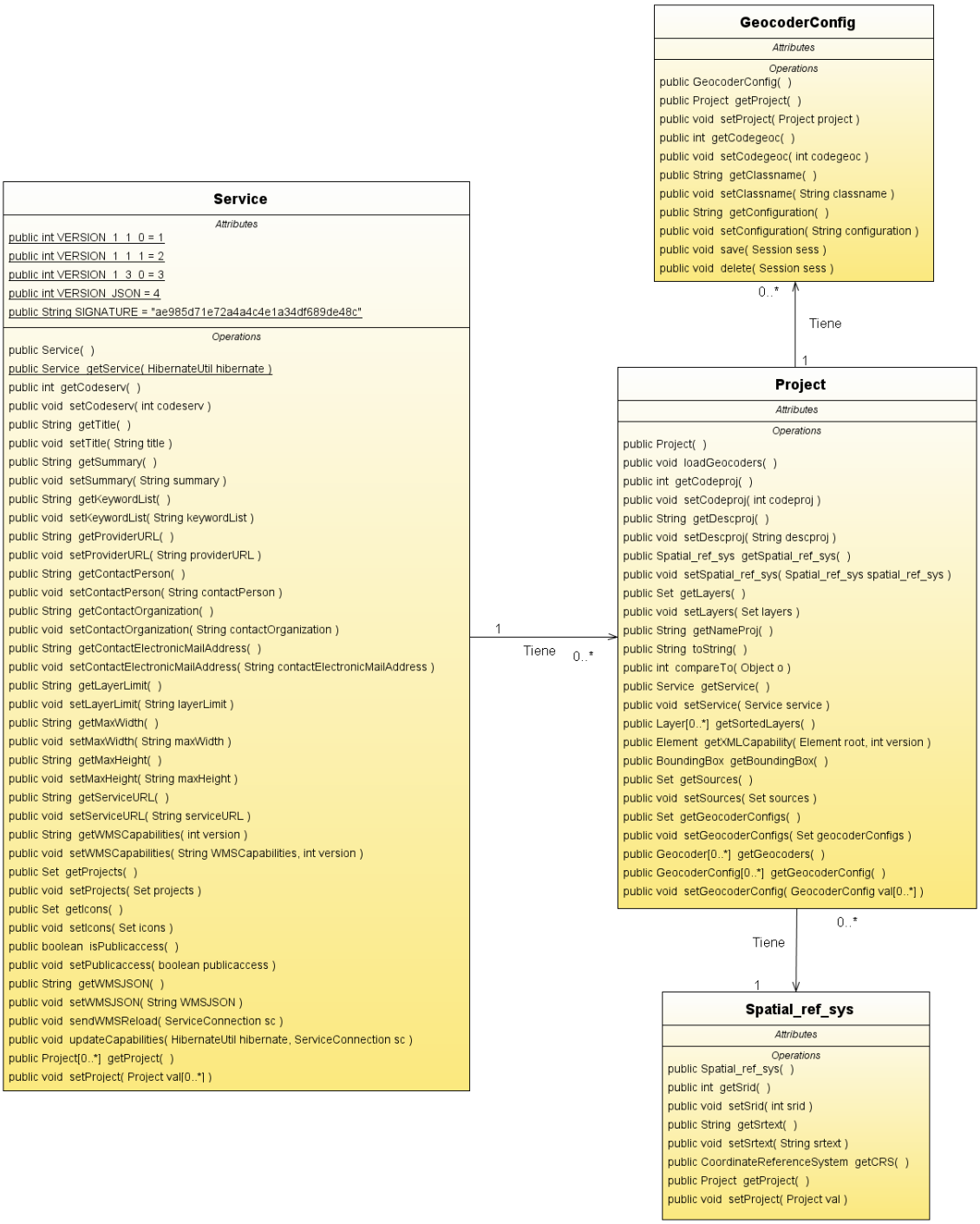


Figura 79. Diagrama B de clases del dominio.

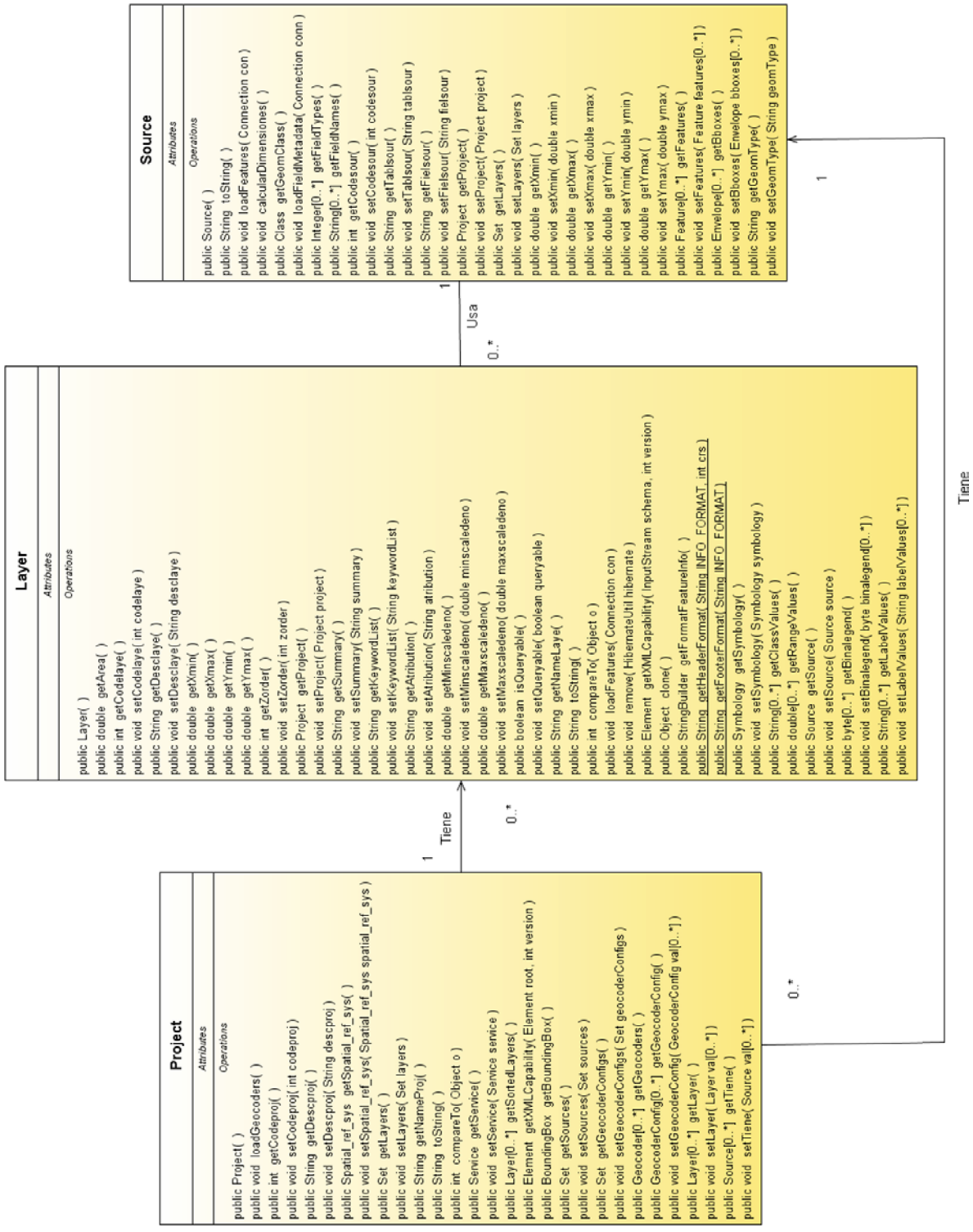


Figura 80. Diagrama C de clases del dominio.

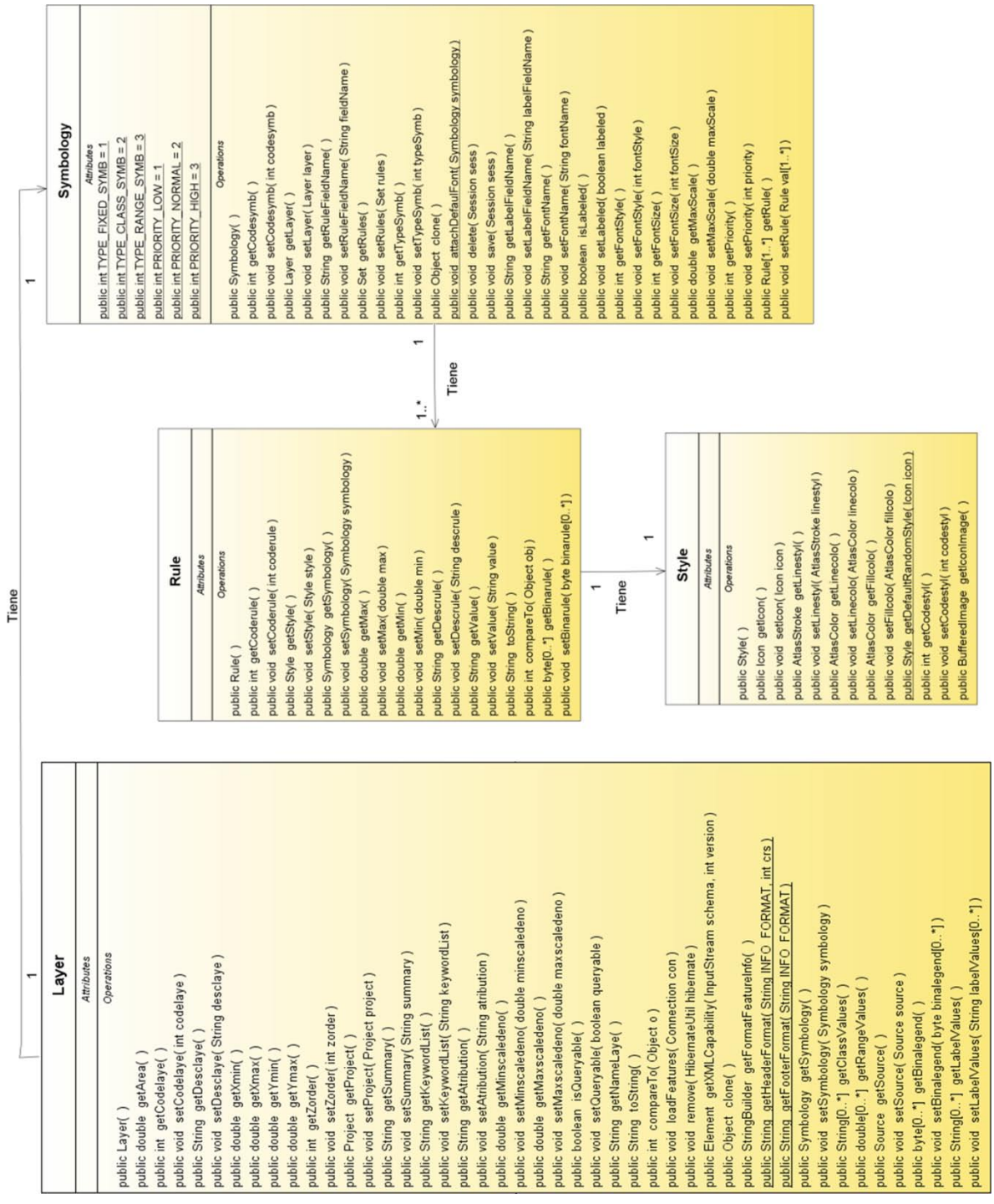
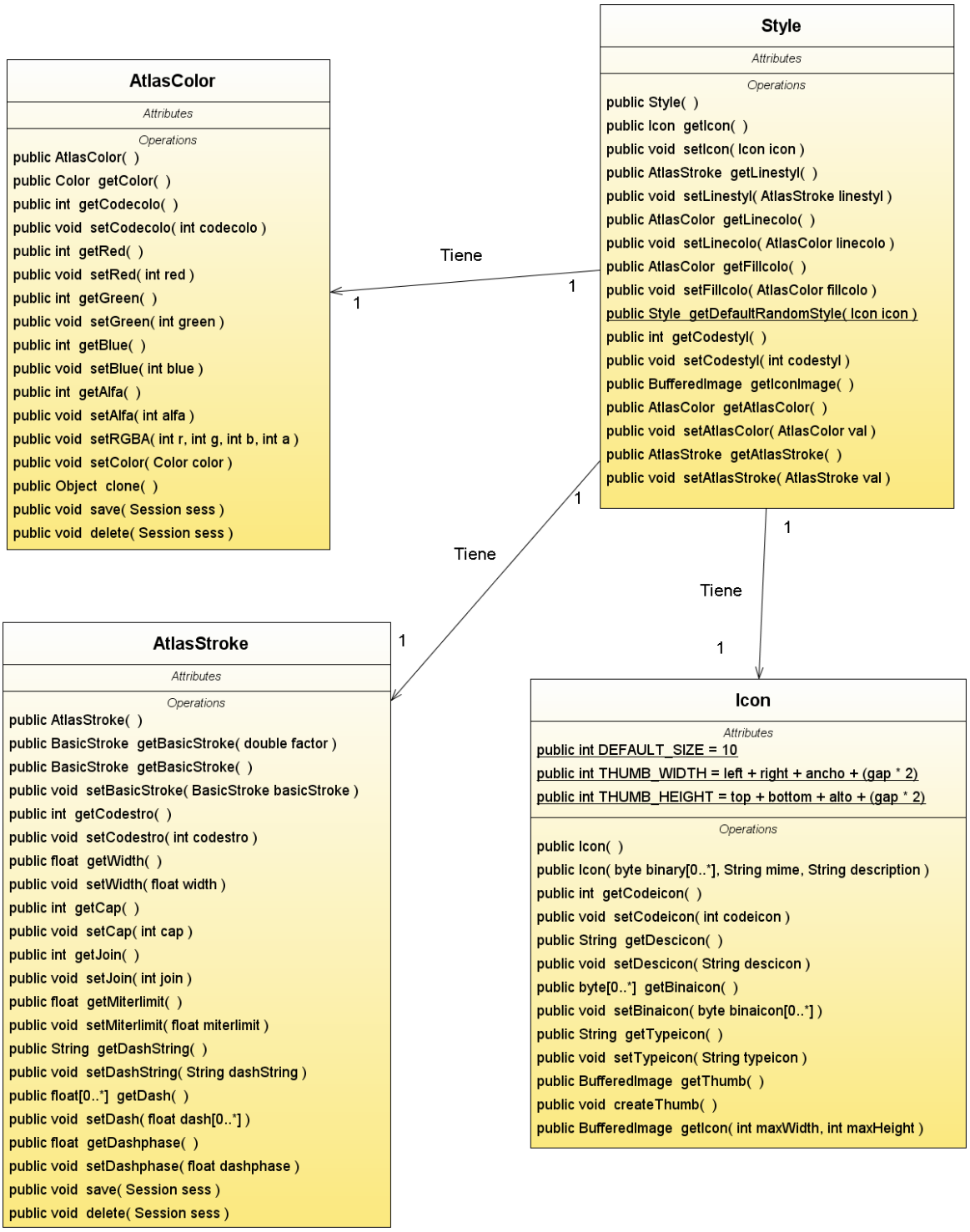


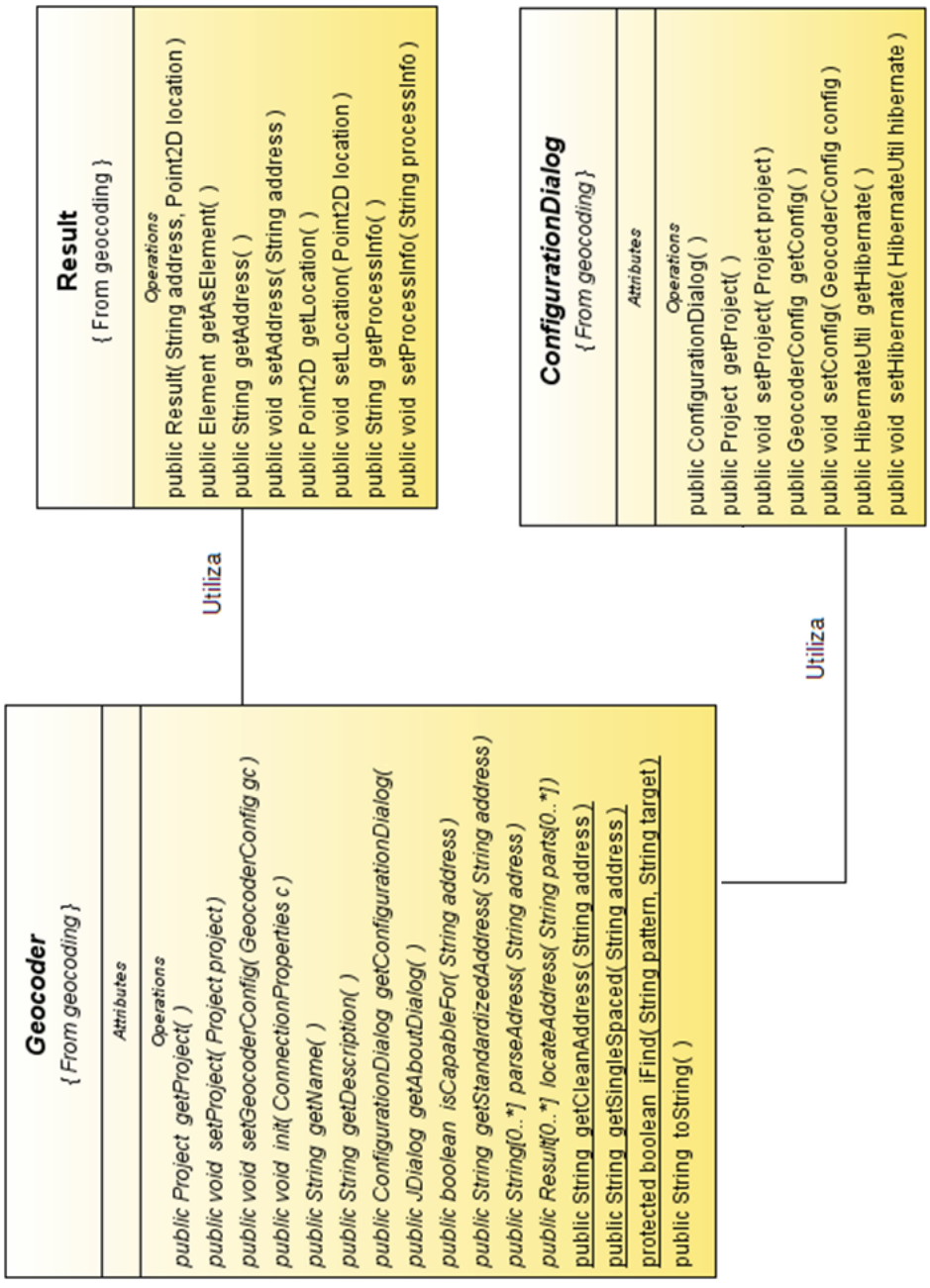
Figura 81. Diagrama D de clases del dominio.



6.3.21. Diagramas de clases para geocodificación.

La figura 82 representa el diagrama de clases abstractas e interfaces para el geocoder.

Figura 82. Diagrama de clases abstractas e interfaces de un geocoder.



6.3.22. Diagramas de clases para la biblioteca de componentes de desarrollo.

Las figuras 83 y 84 representan los diagramas generales de clases para los componentes de desarrollo.

Figura 83. Diagrama de clases del componente MapPanel.

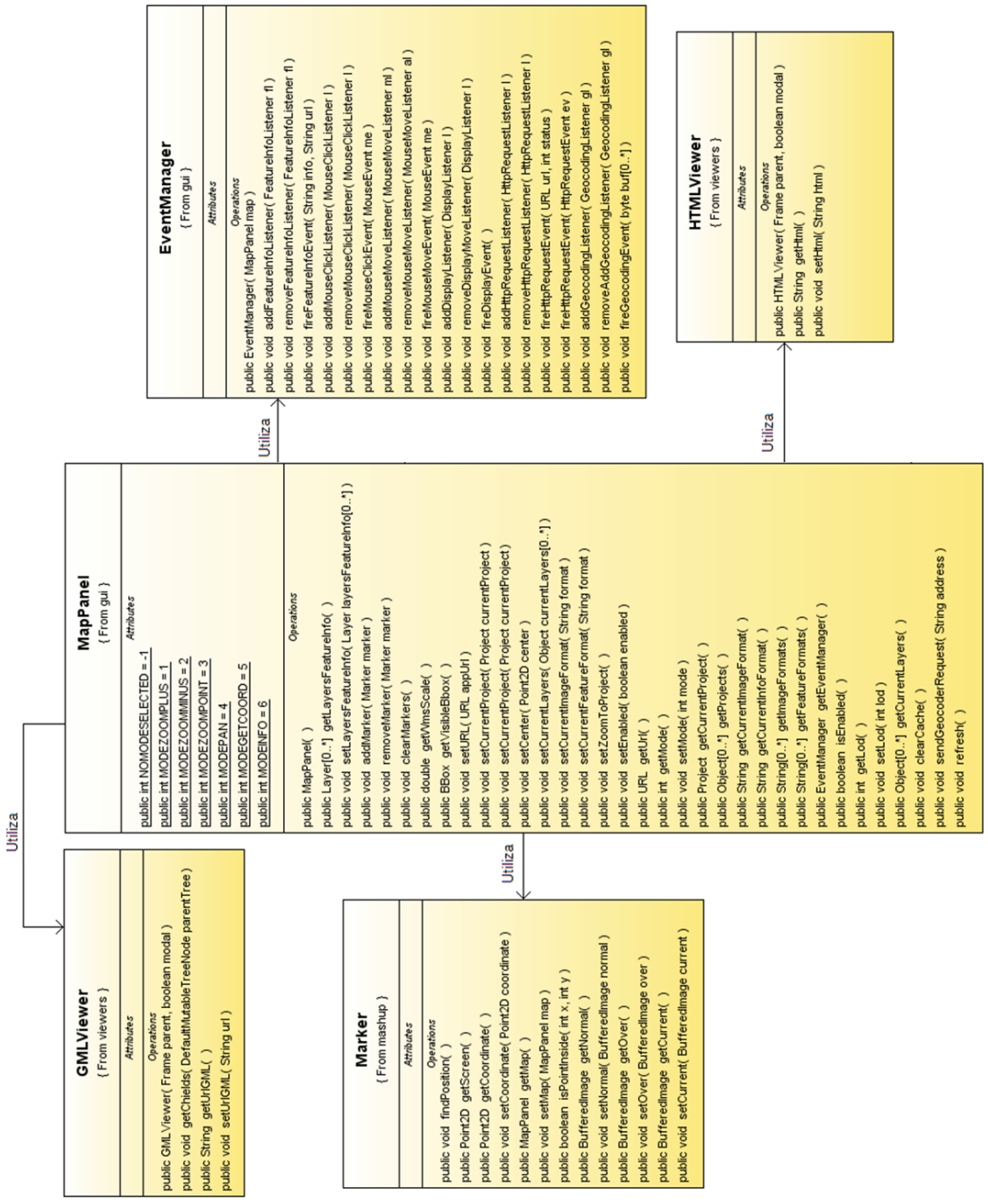
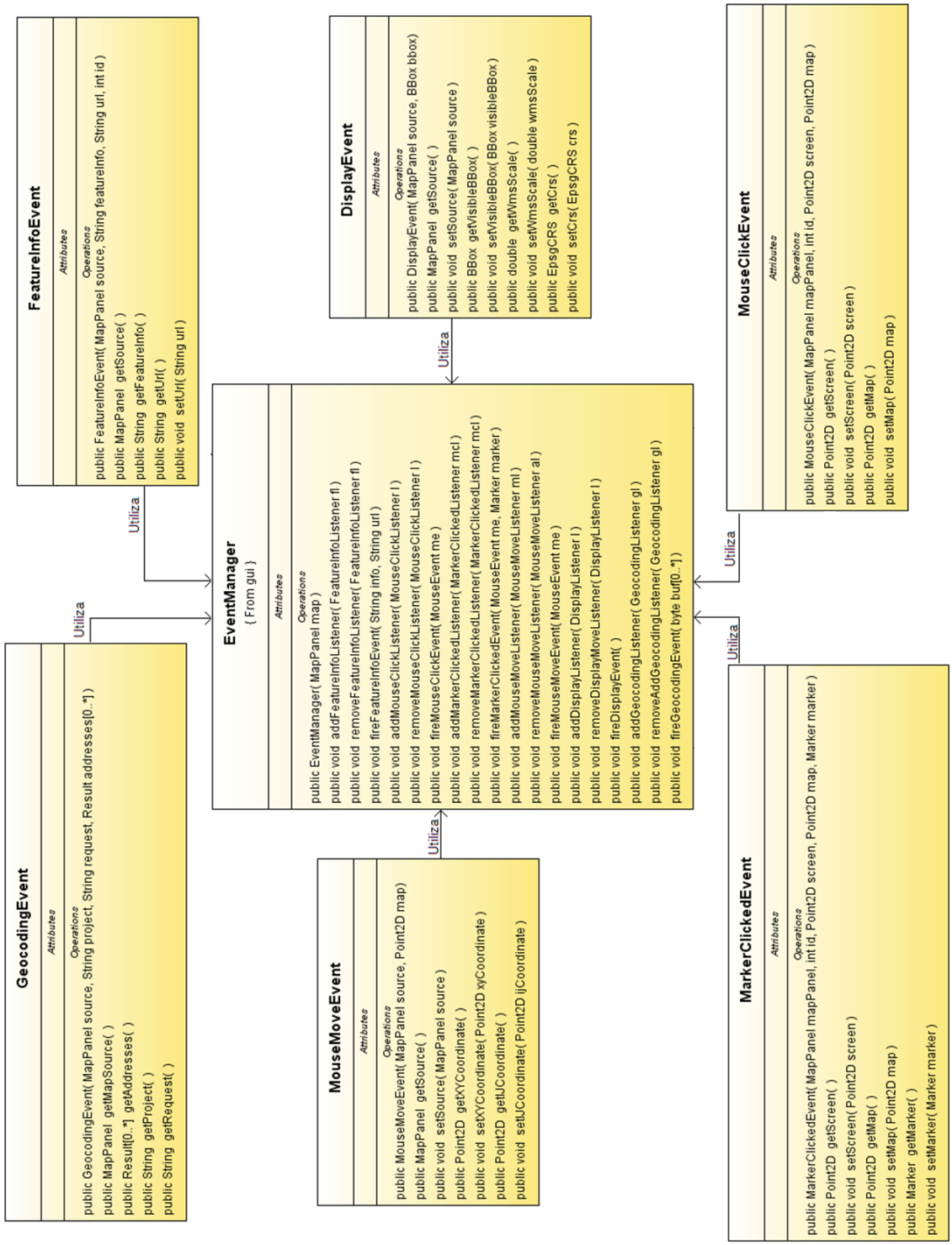


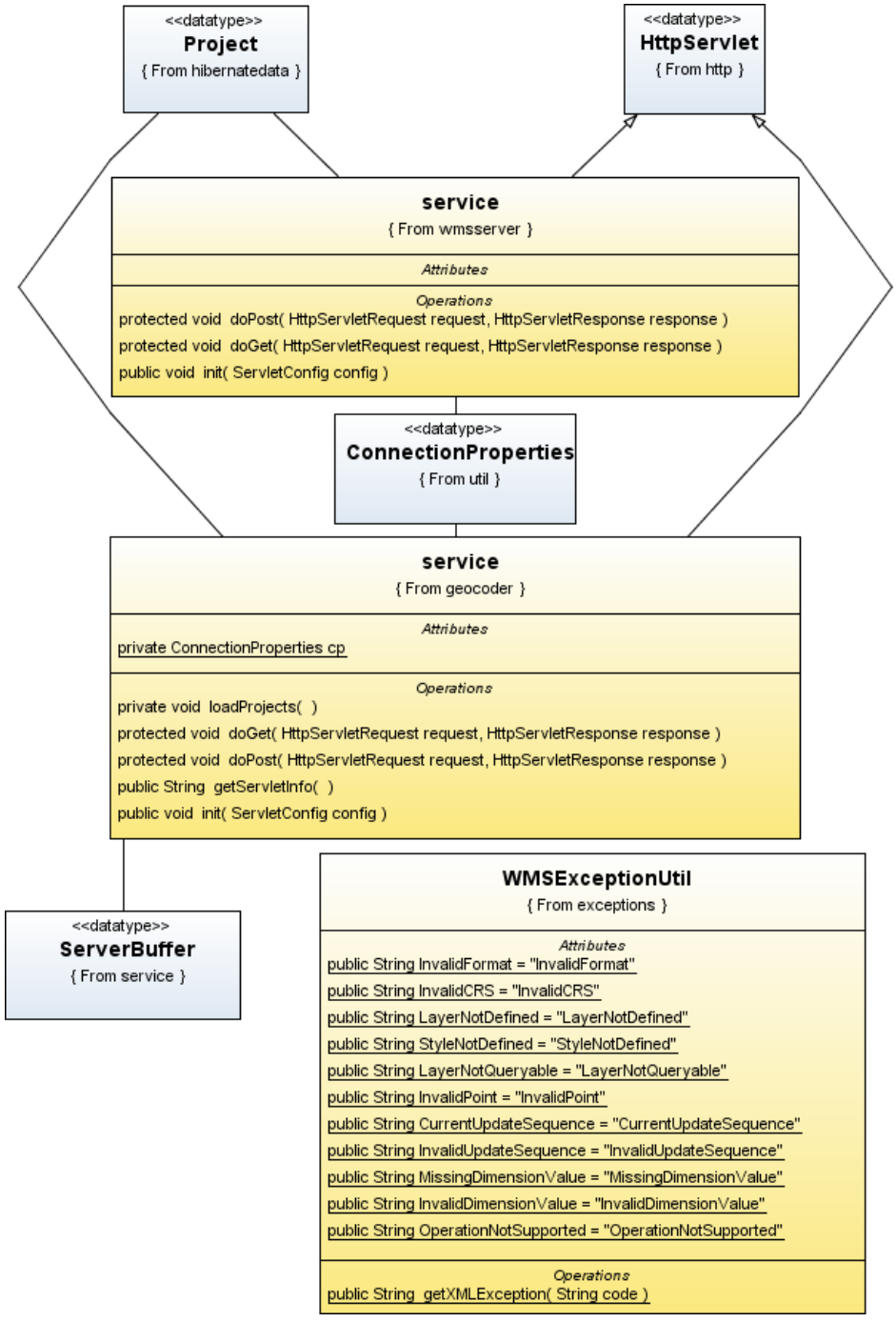
Figura 84. Diagrama de clases de eventos.



6.3.23. Diagramas de clases del lado del servidor.

La figura 85 muestra el diagrama de clases para el servidor Atlas.

Figura 85. Diagrama de clases de servlets.



7. IMPLEMENTACIÓN DE LA HERRAMIENTA ATLAS

Los sistemas operativos sobre los cuales se trabajó durante el desarrollo e implementación de la herramienta Atlas son Linux y Windows. Los lenguajes de programación en los cuales está elaborado Atlas son Java y JavaScript. EL sistema gestor de bases de datos en el cual se trabajó es PostgreSQL.

7.1. ARQUITECTURA DE ATLAS

Los siguientes son los módulos conceptuales que componen esta herramienta y su diagrama se muestra en la figura 86.

Módulo WMS. Este módulo es el encargado de administrar el envío y recepción de las peticiones de conformidad con el estándar WMS, en sus versiones 1.1.0, 1.1.1, 1.3.0.

Modulo de geocodificación. Este módulo es el encargado de administrar la configuración, el envío y recepción de las peticiones al servicio de geocodificación.

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.

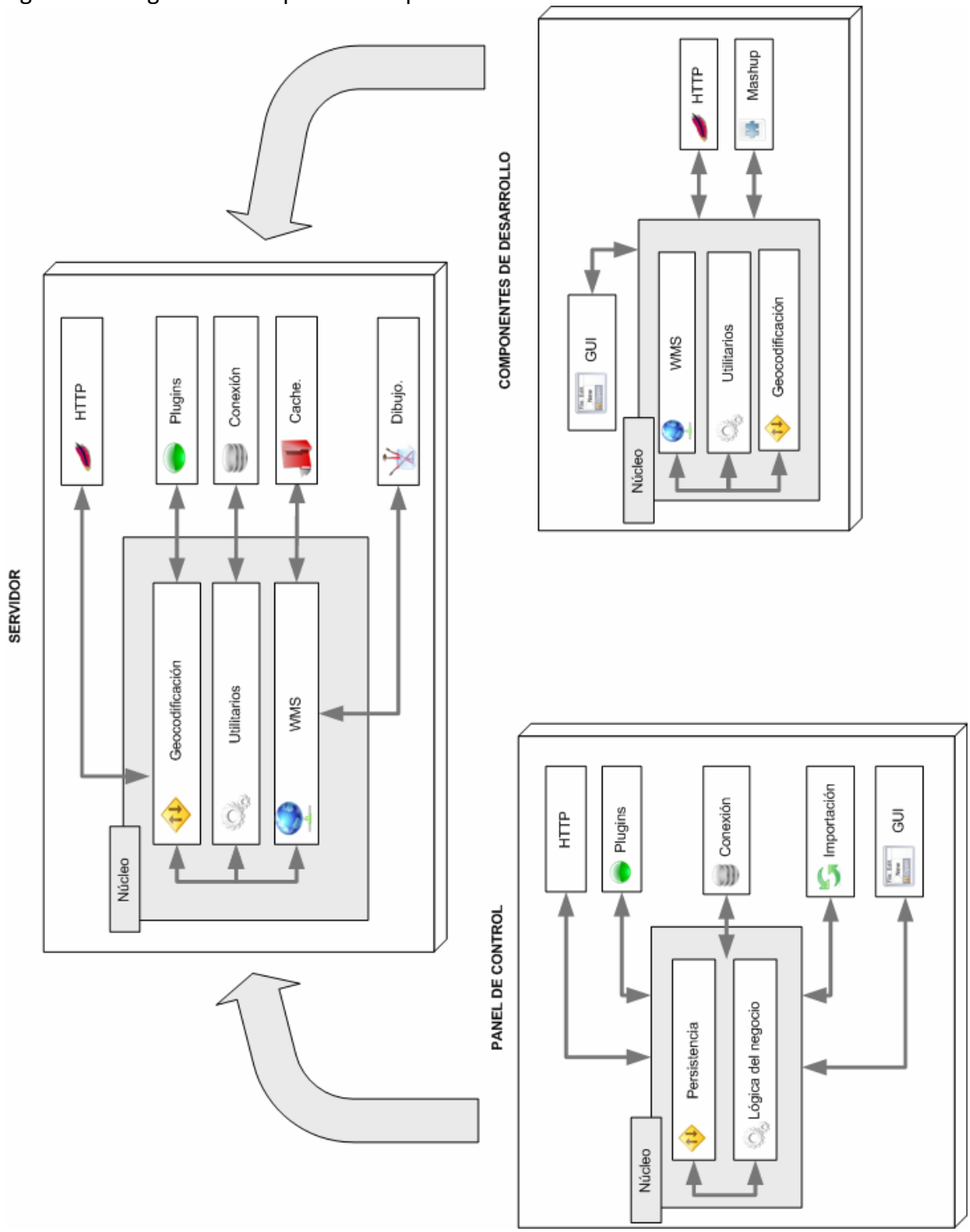
Módulo de Conexión. Este módulo es el encargado de mantener una comunicación constante entre la herramienta y la base de datos.

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

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.

Figura 86. Diagrama conceptual de arquitectura de la herramienta Atlas.



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.

Modulo de Utilitarios. Este módulo 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.

Módulo Mashup. Este módulo se encarga de administrar los componentes asociados a la producción de aplicaciones hibridas usando la biblioteca
A continuación se describe de manera específica la estructura de paquetes de Atlas y las clases que pertenecen a cada paquete. Posteriormente, se amplia y se detalla la forma como fueron desarrolladas dichas clases.

7.2. MODULO DE UTILIDAD

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.

7.2.1. 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 siguiente tabla.

Tabla 13. 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 siguiente tabla.

Tabla 14. 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 siguiente tabla.

Tabla 15. 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 siguiente tabla.

Tabla 16. 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.

7.2.2. Paquete geocoding.

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 siguiente tabla.

Tabla 17. 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 siguiente tabla.

Tabla 18. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 19. 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.

7.2.3. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 20. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 21. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 22. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 23. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 24. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 25. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 26. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 27. 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 contactElectronicMail Address) 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 28. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 29. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 30. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 31. 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.

7.2.4. Paquete util. El paquete contiene clases de utilidades empleadas en diferentes lugares del sistema. En la tala 33 se listan y describen las clases de paquete útil.

Tabla 32. 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.

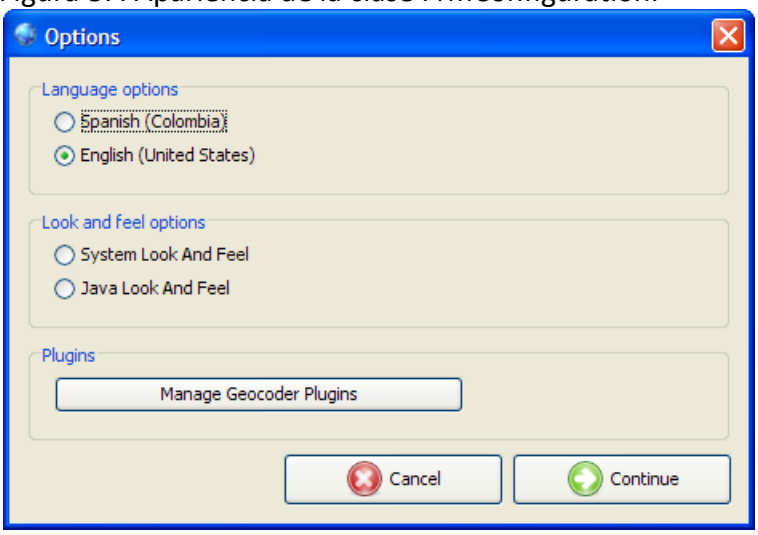
DataBaseUtils	Utilidades para conexión directa con la base de datos.
EWKB	Conversión al formato EWKB.
FrmWait	Formulario modal de espera.
HibernateUtil	Utilidades para creación y mantenimiento de sesiones hibernate.
IOUtils	Utilidades para trabajo con flujos.
PostGis	Utilidades para detectar la instalación 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.

7.3. MODULO PANEL DE CONTROL

7.3.1. Paquete windows.configuration. Contiene diálogos para la configuración de la herramienta.

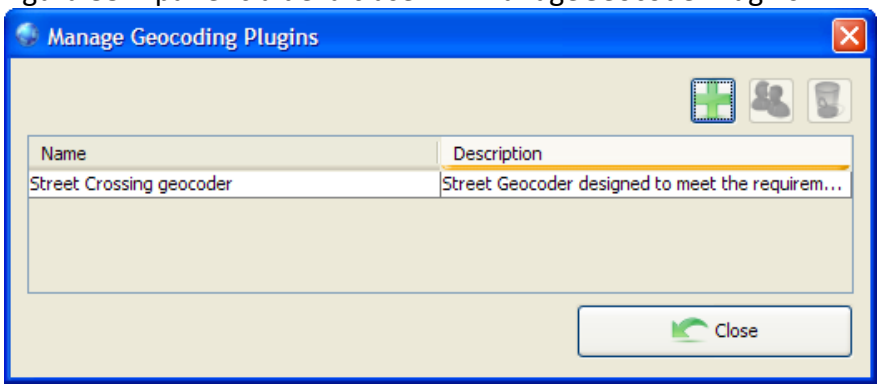
- **Clase FrmConfiguration**. Permite establecer opciones de la herramienta como el idioma de la interfaz, el look and feel e iniciar el administrador de plugins. La apariencia de esta clase se muestra en la siguiente figura.

Figura 87. 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 siguiente figura.

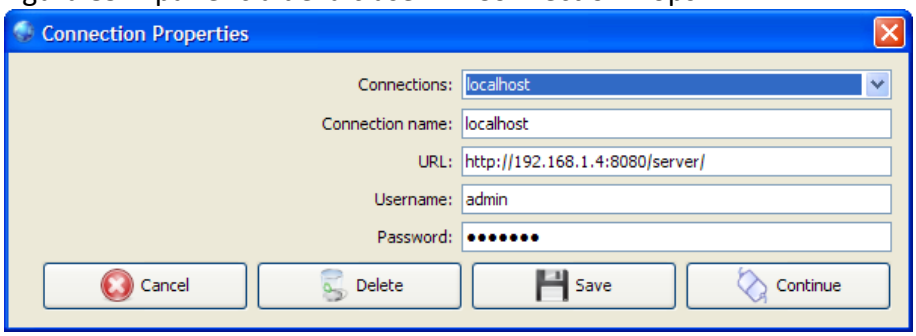
Figura 88. Apariencia de la clase FrmManageGeocoderPlugins.



7.3.2. 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 siguiente figura.

Figura 89. Apariencia de la clase FrmConnectionProps.

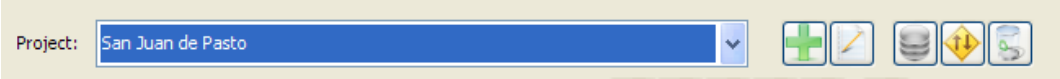


7.3.3. Paquete windows.controls. Contiene los componentes que conforman la ventana principal del la aplicación.

- **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 siguiente figura.

Figura 90. 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 91 y 92.

Figura 91. Apariencia de la clase PnlLayers en pestaña de capas.

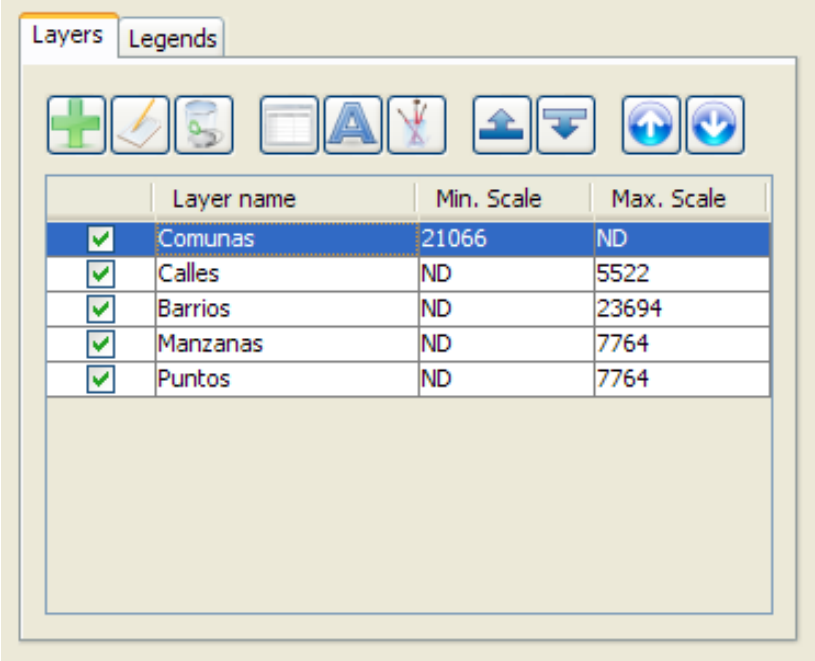
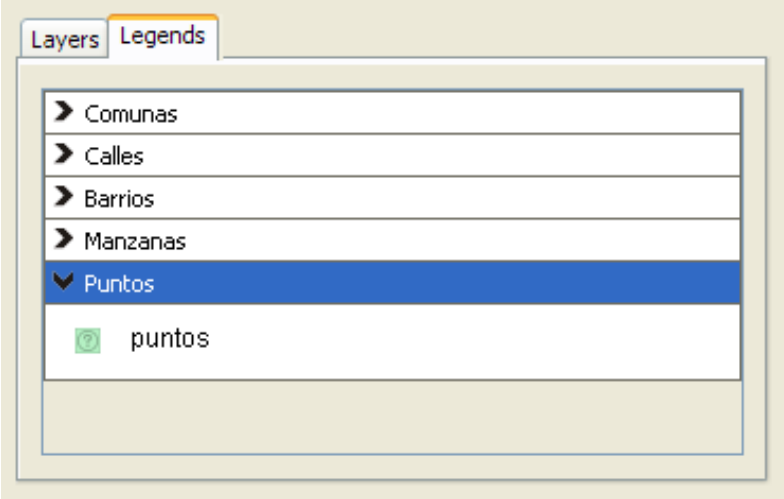
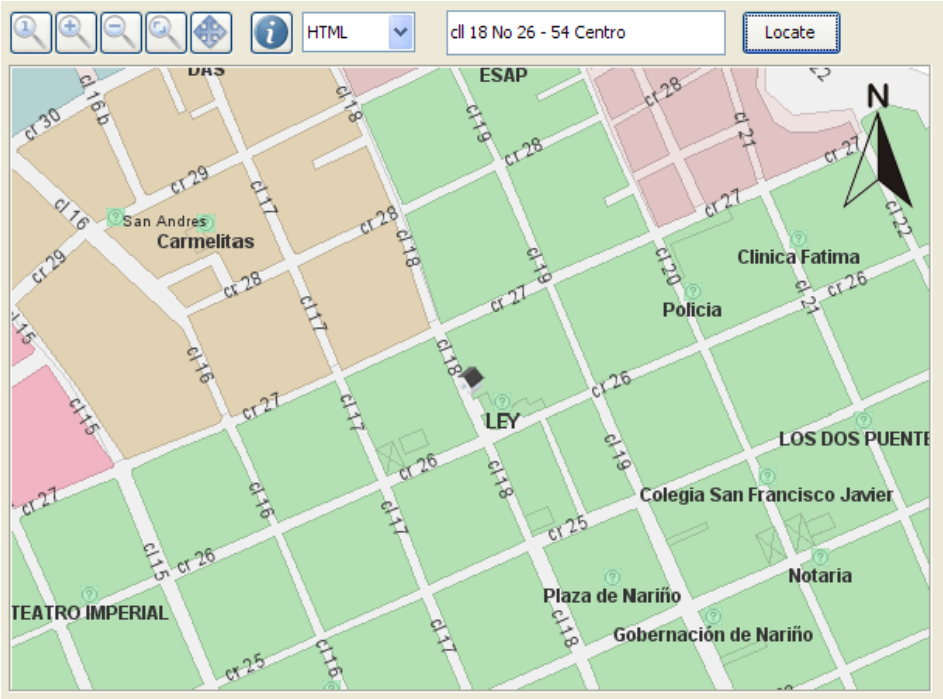


Figura 92. Apariencia de la clase PnlLayers en pestaña de leyendas.



- **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.

Figura 93. Apariencia de la clase PnlViewer.



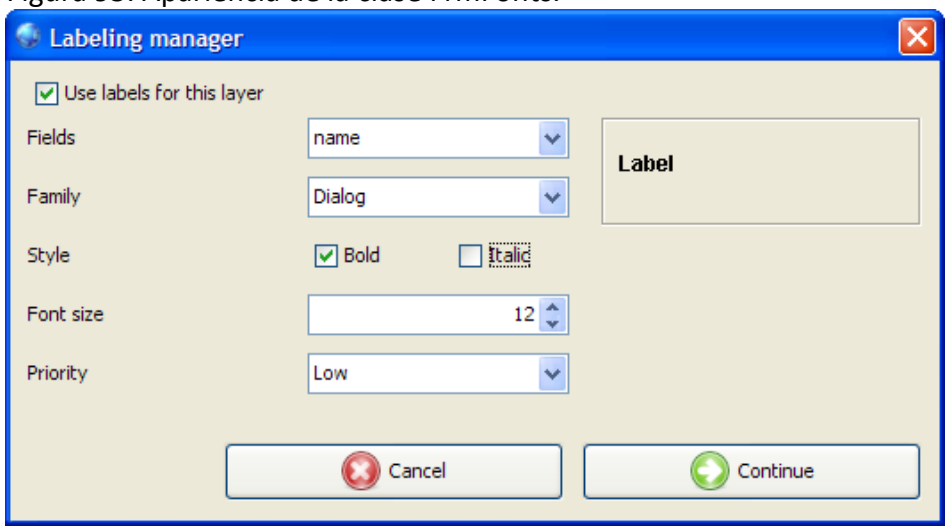
- **Clase AtlasColorChooser**. Componente que permite la sección de un color con canal alfa. La apariencia de esta clase se muestra en la siguiente figura.

Figura 94. Apariencia de la clase AtlasColorChooser.

7.3.4. 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 la siguiente figura.

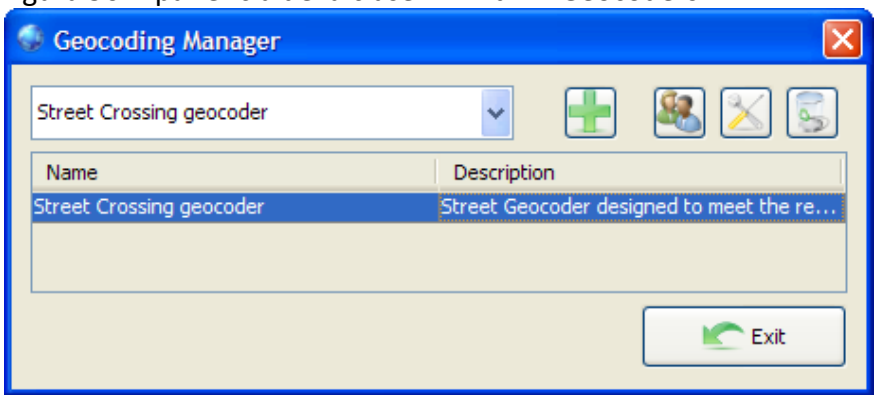
Figura 95. Apariencia de la clase FrmFonts.



7.3.5. 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 siguiente figura.

Figura 96. Apariencia de la clase FrmAdminGeocoders.



7.3.6. 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 siguiente figura.

Figura 97. 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 98.

Figura 98. Apariencia de la clase FrmAdminIcons.

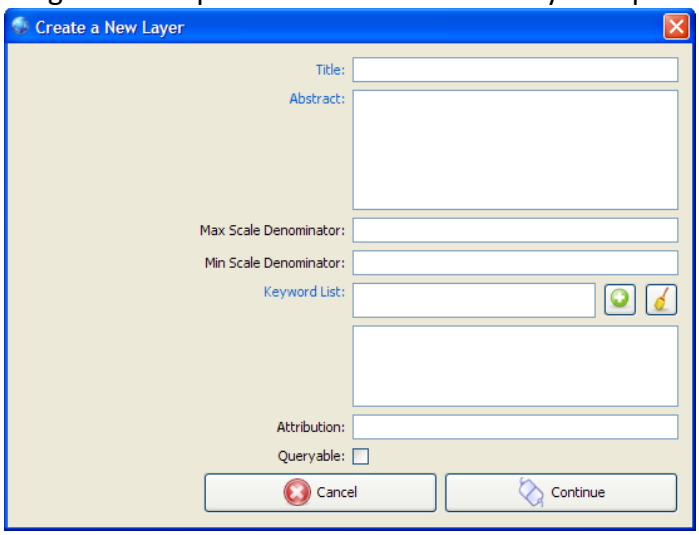
- **Clase IconsPanel**. Componente que permite mostrar un listado de vistas en miniatura de varios iconos, permite operaciones de selección, adición y eliminación. La apariencia de esta clase se muestra en la siguiente figura.

Figura 99. Apariencia de la clase IconsPanel.

7.3.7. 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 siguiente figura.

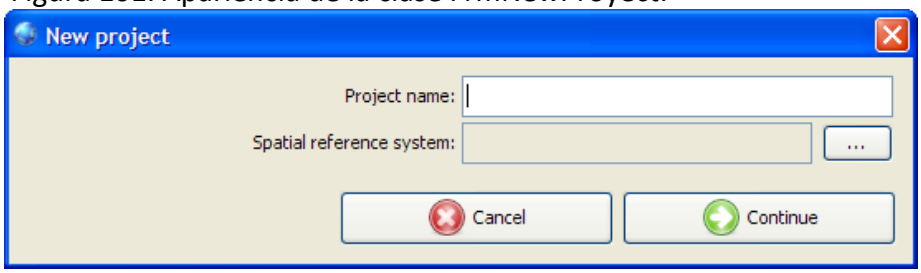
Figura 100. Apariencia de la clase FrmLayerProps.



7.3.8. 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 siguiente figura.

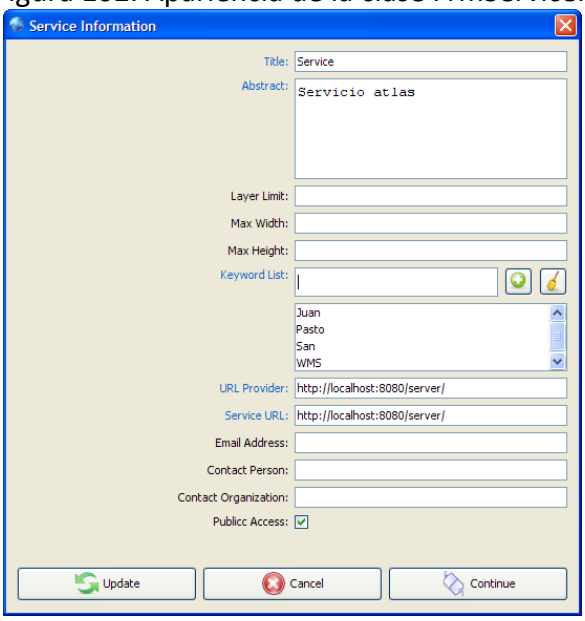
Figura 101. Apariencia de la clase FrmNewProyect.



7.3.9. 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 siguiente figura.

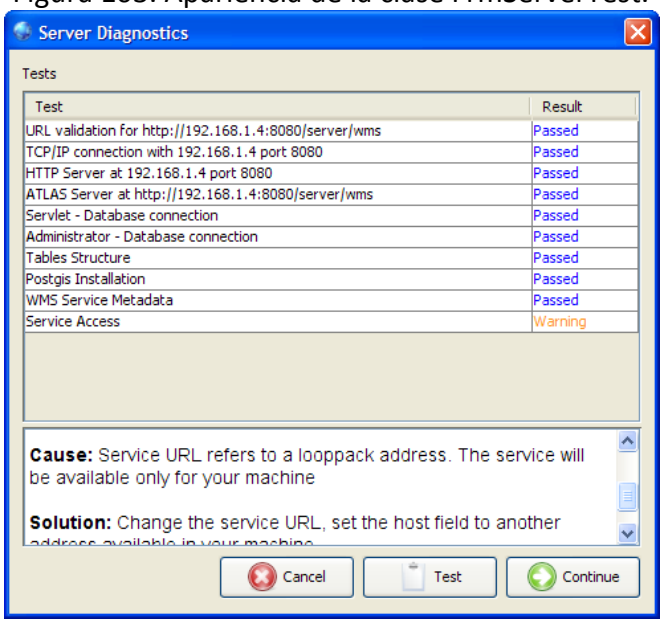
Figura 102. Apariencia de la clase FrmServiceProps.



7.3.10. 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 la siguiente figura.

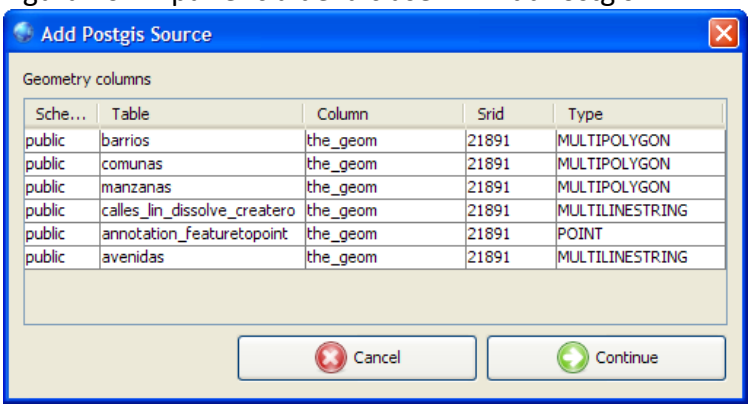
Figura 103. Apariencia de la clase FrmServerTest.



7.3.11. 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 siguiente figura.

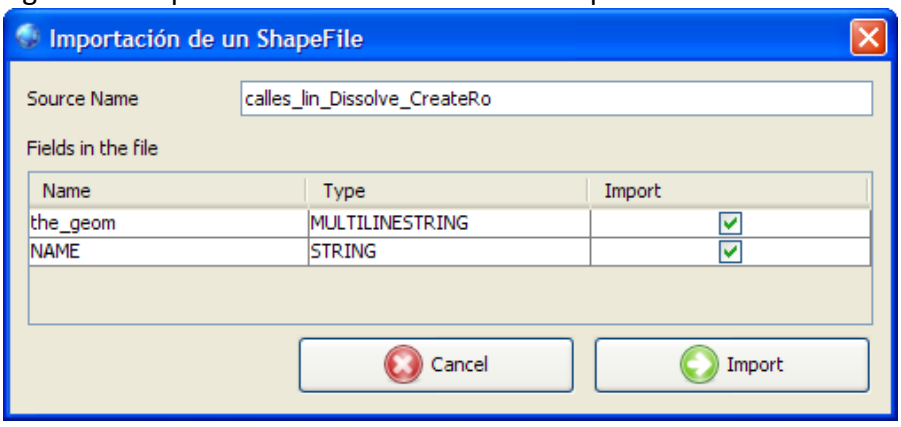
Figura 104. Apariencia de la clase FrmAddPostgis.



- **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

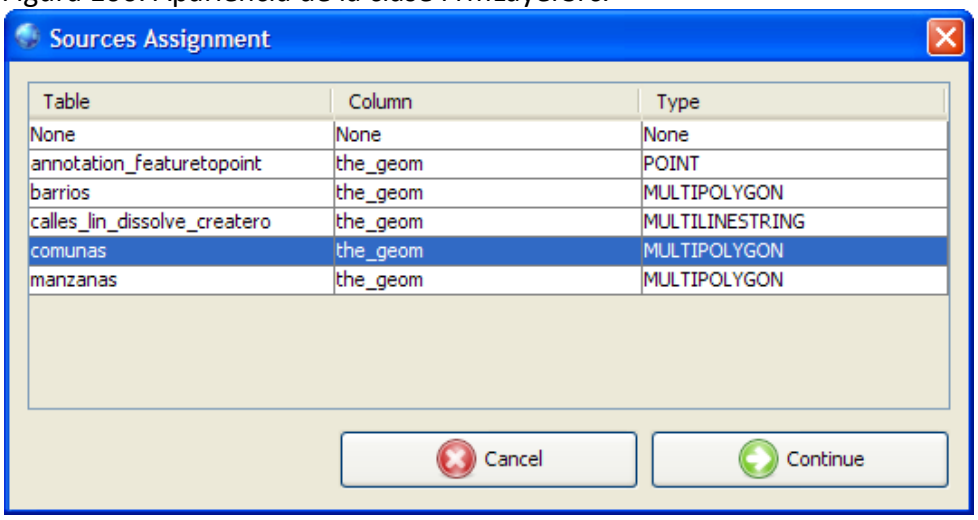
seleccione cuáles de ellos desea importar. La apariencia de esta clase se muestra en la siguiente figura.

Figura 105. Apariencia de la clase FrmAddShapeFile.



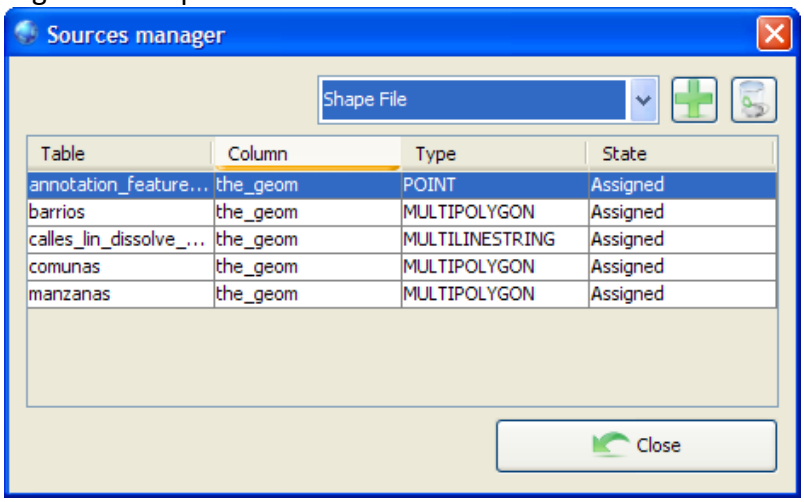
- **Clase FrmLayerSrc.** Permite seleccionar el origen de datos que será usado por la capa. La apariencia de esta clase se muestra en la siguiente figura.

Figura 106. 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 siguiente figura.

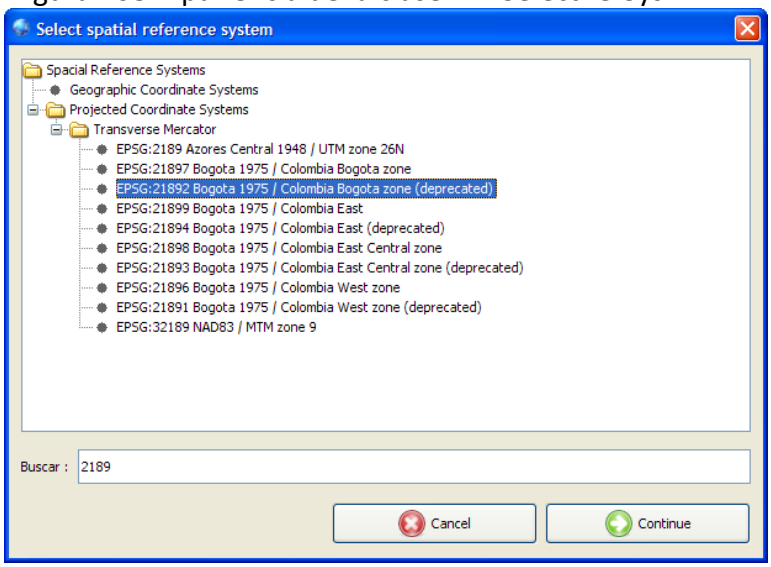
Figura 107. Apariencia de la clase FrmSources.



7.3.12. 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 108.

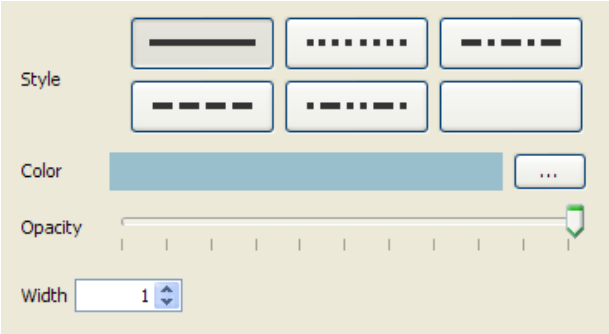
Figura 108. Apariencia de la clase FrmSelectRefSys.



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

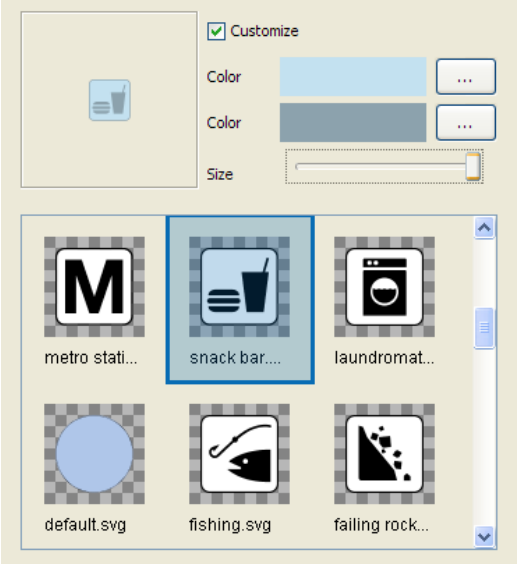
- **Clase PnlEditLineStyl**. Permite la edición de estilos para líneas, notifica de los cambios en el estilo atreves de escuchadores. La apariencia de esta clase se muestra en la siguiente figura.

Figura 109. Apariencia de la clase PnlEditLineStyl.



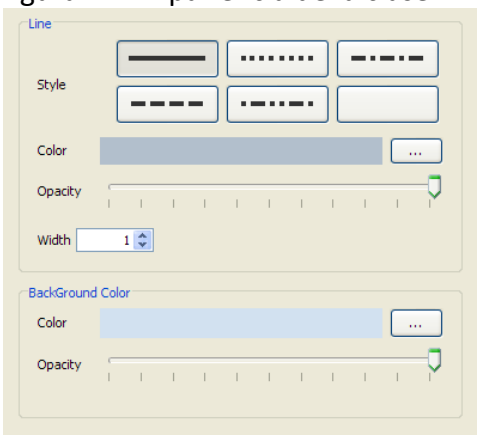
- **Clase PnlEditPointStyle**. Permite la edición de estilos para puntos, notifica de los cambios en el estilo atreves de escuchadores. Permite seleccionar un icono desde la biblioteca de iconos del sistema, personalizar su tamaño, color de borde y de relleno. La apariencia de esta clase se muestra en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

Figura 110. 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 siguiente figura.

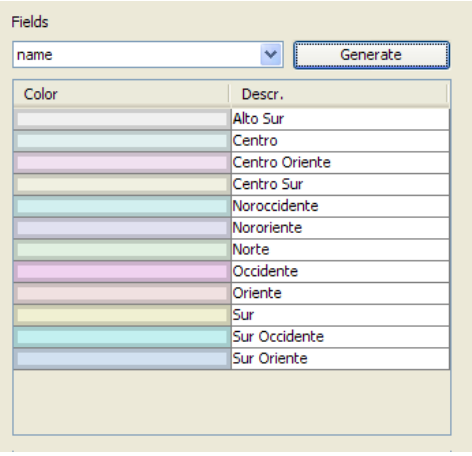
Figura 111. Apariencia de la clase PnlEditPolygonStyle.



7.3.14. 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

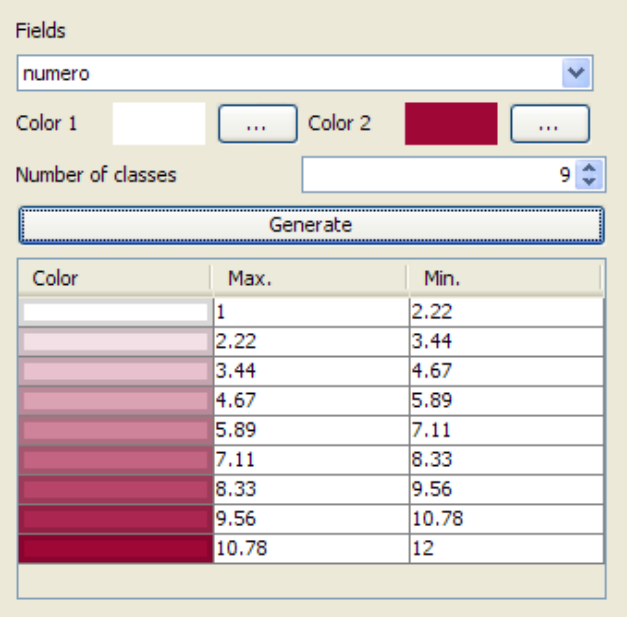
- **Clase ClassRuleGenerator**. Permite la generación de reglas de simbología, formando clases dado un campo del origen de datos de la capa. Luego, de ser necesario, entregará el estilo de cada regla generada al componente de la clase Windows.style que corresponda según el tipo de geometría. La apariencia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

Figura 112. Apariencia de la clase ClassRuleGenerator.



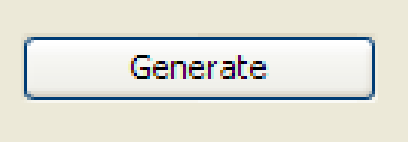
- **Clase RangeRuleGenerator**. Permite la generación de reglas de simbología formando rangos dado un campo del origen de datos de la capa y el número deseado de clases. Luego, de ser necesario, entregará el estilo de cada regla generada al componente de la clase Windows.style que corresponda. La apariencia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

Figura 113. 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 siguiente figura.

Figura 114. 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 115, 116 y 117.

Figura 115. Apariencia de la clase FrmSymbLine con simbología fija.

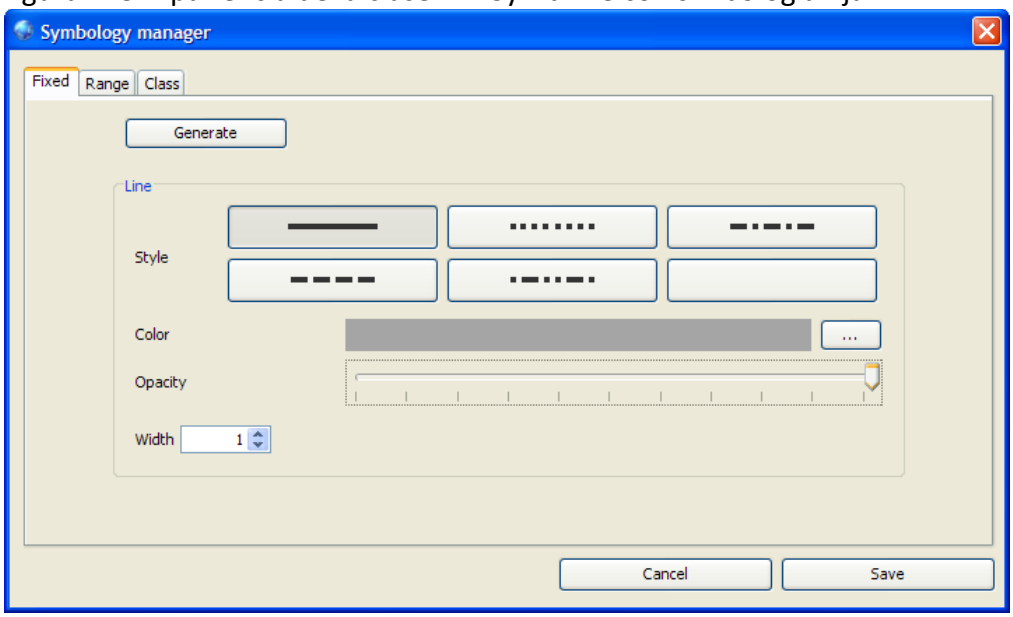


Figura 116. Apariencia de la clase FrmSymbLine con simbología por rangos.

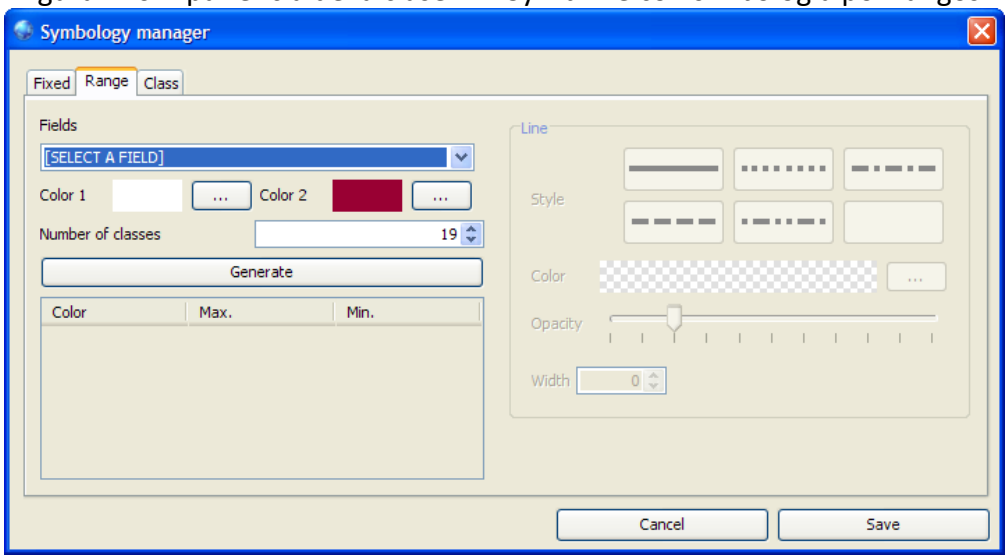
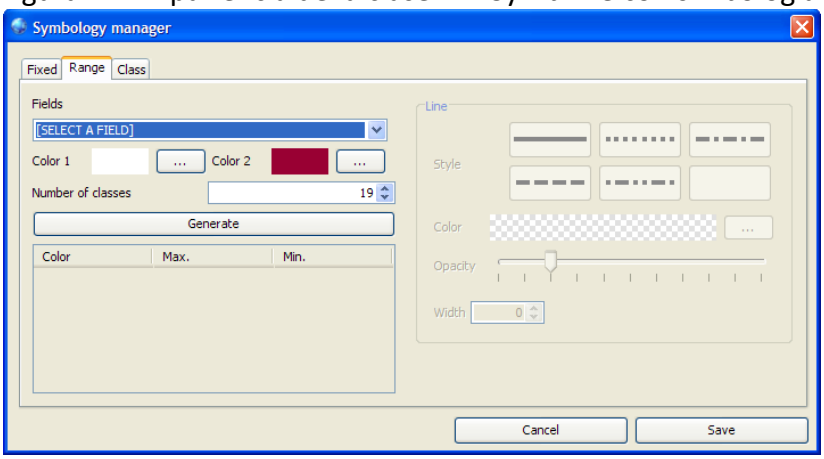


Figura 117. Apariencia de la clase FrmSymbLine con simbología por clases.



- **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 118, 119 y 120.

Figura 118. Apariencia de la clase FrmSymbPoint con simbología fija.

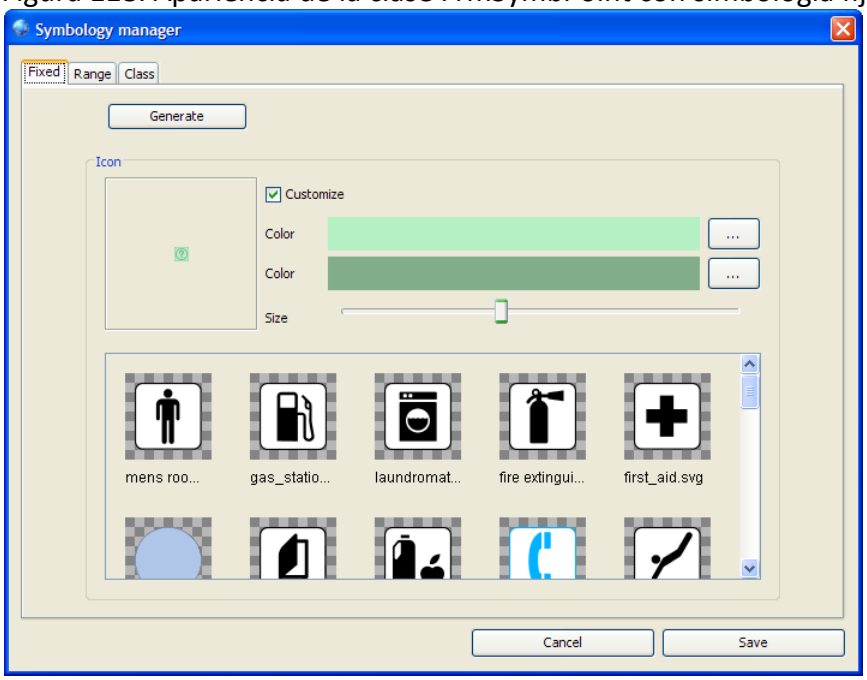


Figura 119. Apariencia de la clase FrmSymbPoint con simbología por rangos.

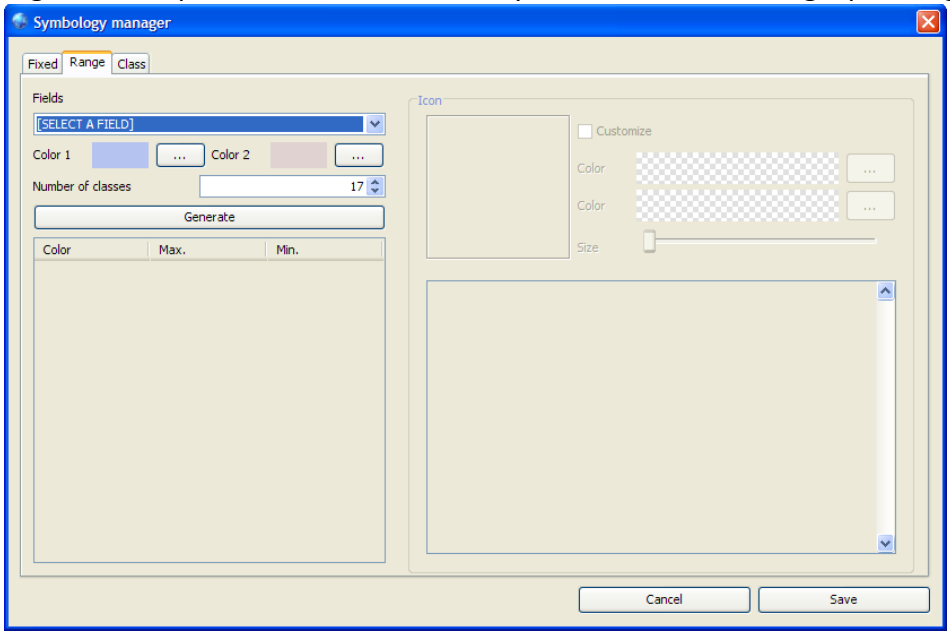
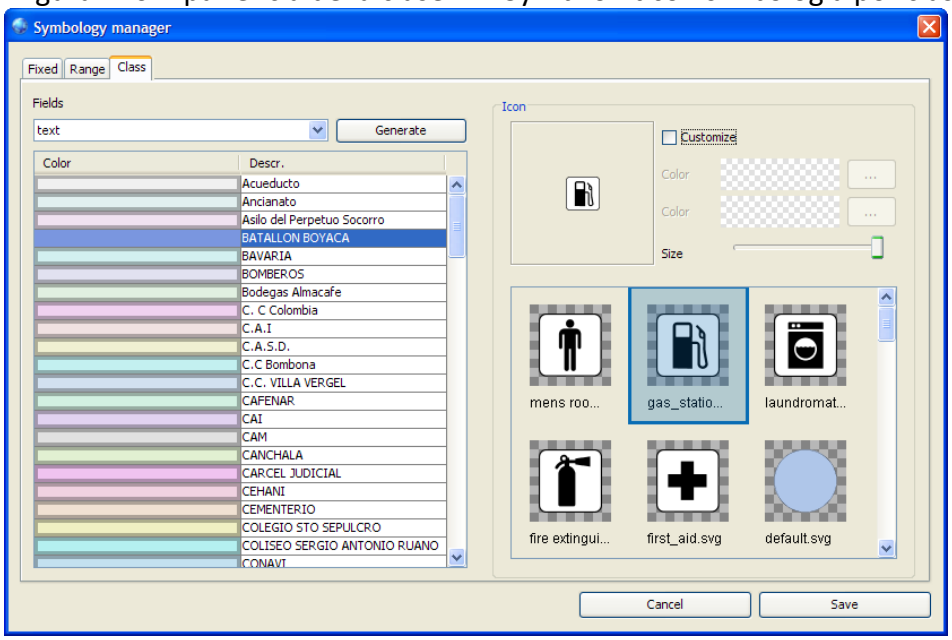


Figura 120. Apariencia de la clase FrmSymbPoint con simbología por clases.



- **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 121, 122 y 123.

Figura 121. Apariencia de la clase FrmSymbPoly con simbología fija.

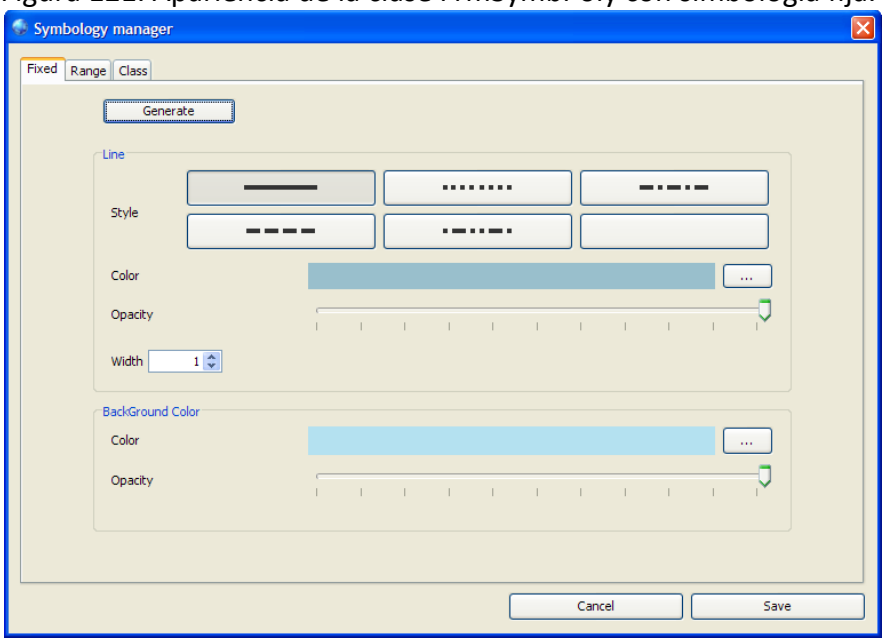


Figura 122. Apariencia de la clase FrmSymbPoly con simbología por rangos.

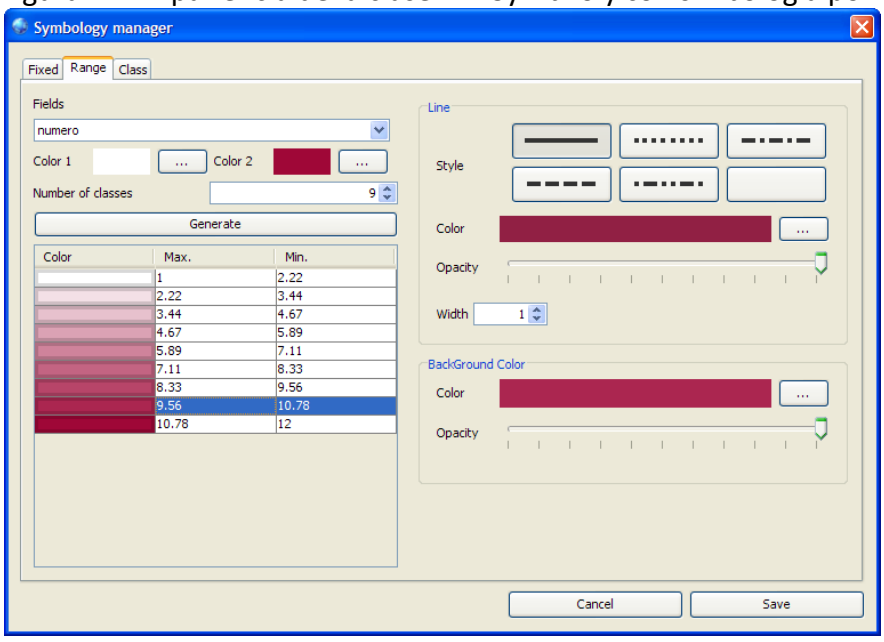
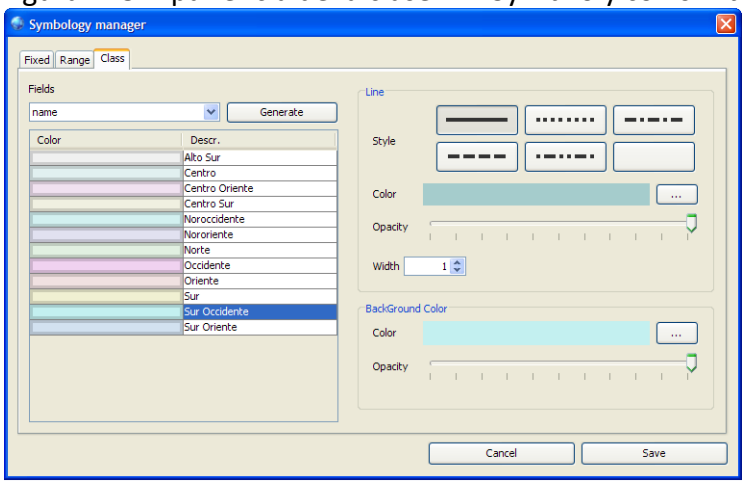


Figura 123. Apariencia de la clase FrmSymbPoly con simbología por clases.



7.4. MODULO DE SERVIDOR

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

7.4.1. 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 siguiente tabla.

Tabla 33. 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.

7.4.2. 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 siguiente tabla.

Tabla 34. 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.

7.4.3. 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 siguiente tabla.

Tabla 35. 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 sistemas 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 llamado por el contenedor de servlets.

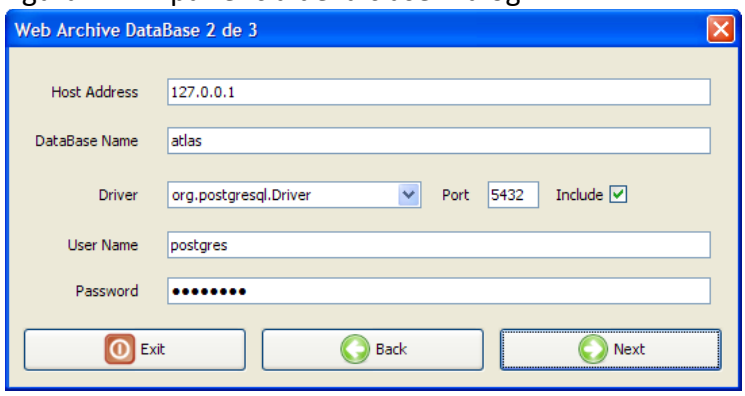
7.5. MODULO WEB ARCHIVE

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

7.5.1. Paquete tool. Contiene los diálogos de la herramienta.

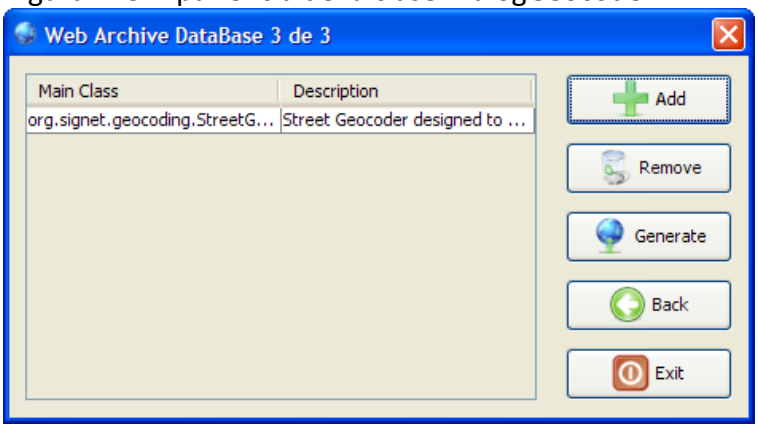
- **Clase DialogDB**. Permite establecer opciones sobre la forma en que los servlets de la aplicación se conectan con la base de datos. La apariencia de esta clase se muestra en la siguiente figura.

Figura 124. Apariencia de la clase DialogDB.



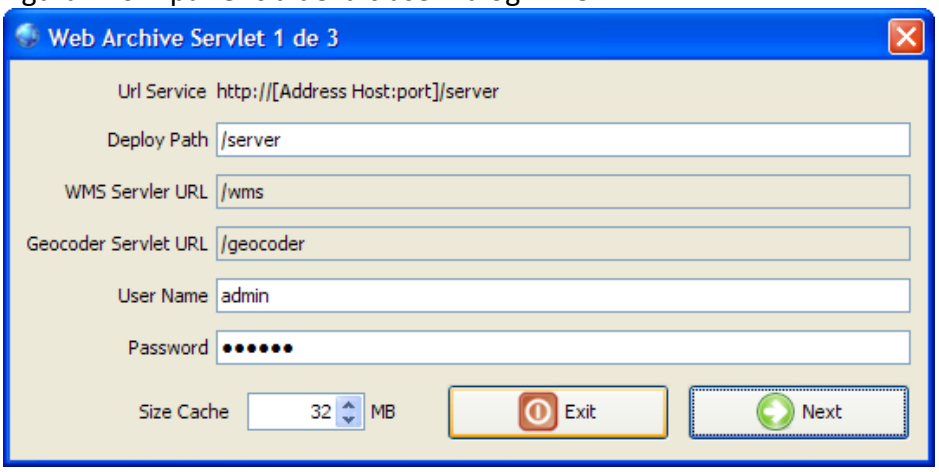
- **Clase DialogGeocoder**. Permite establecer los plugins de geocodificación para el servidor. La apariencia de esta clase se muestra en la siguiente figura.

Figura 125. 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 siguiente figura.

Figura 126. Apariencia de la clase DialogWMS.



7.6. BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA STANDARD EDITION

7.6.1. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 36. 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 siguiente tabla.

Tabla 37. 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.

7.6.2. Paquete events. Contiene las clases e interfaces asociadas a diferentes eventos manipulados por el EventManager. En la tabla 39 se listan y describen las clases e interfaces del paquete events.

Tabla 38. 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 39. 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.

7.6.3. 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 siguiente tabla.

Tabla 40. 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.

7.7. BIBLIOTECA PARA DESARROLLO DE APLICACIONES WEB 2.0 CON JAVASCRIPT

7.7.1. 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 siguiente tabla.

Tabla 41. 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 siguiente tabla.

Tabla 42. 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 siguiente tabla.

Tabla 43. 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 siguiente tabla.

Tabla 44. 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 siguiente tabla.

Tabla 45. 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 siguiente tabla.

Tabla 46. 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 siguiente tabla.

Tabla 47. 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 siguiente tabla.

Tabla 48. 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 siguiente tabla.

Tabla 49. 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.

7.8. BIBLIOTECA PARA DESARROLLO DE APLICACIONES JAVA MOBILE EDITION

7.8.1. 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 siguiente tabla.

Tabla 50. 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. Más detalles de esta clase se muestran en la siguiente tabla.

Tabla 51. 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 siguiente tabla.

Tabla 52. 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 siguiente tabla.

Tabla 53. 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.

7.8.2. 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 siguiente tabla.

Tabla 54. 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 siguiente tabla.

Tabla 55. 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 siguiente tabla.

Tabla 56. 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 la siguiente tabla.

Tabla 57. 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 58. 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.

7.8.3. 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 siguiente tabla.

Tabla 59. 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.

7.8.4. 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 siguiente tabla.

Tabla 60. 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 siguiente tabla.

Tabla 61. 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.

8. 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 62. 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 62. Características de los computadores usados en las pruebas.

Procesador	Intel Pentium D 3.4 GHZ
Memoria RAM	2 Gb DDR2
Disco Duro	SATA-II de 80 GB y 7200 RPM
Sistema Operativo	Windows XP.
Tarjeta de Red	PCI velocidad 10/100T

8.1. 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.

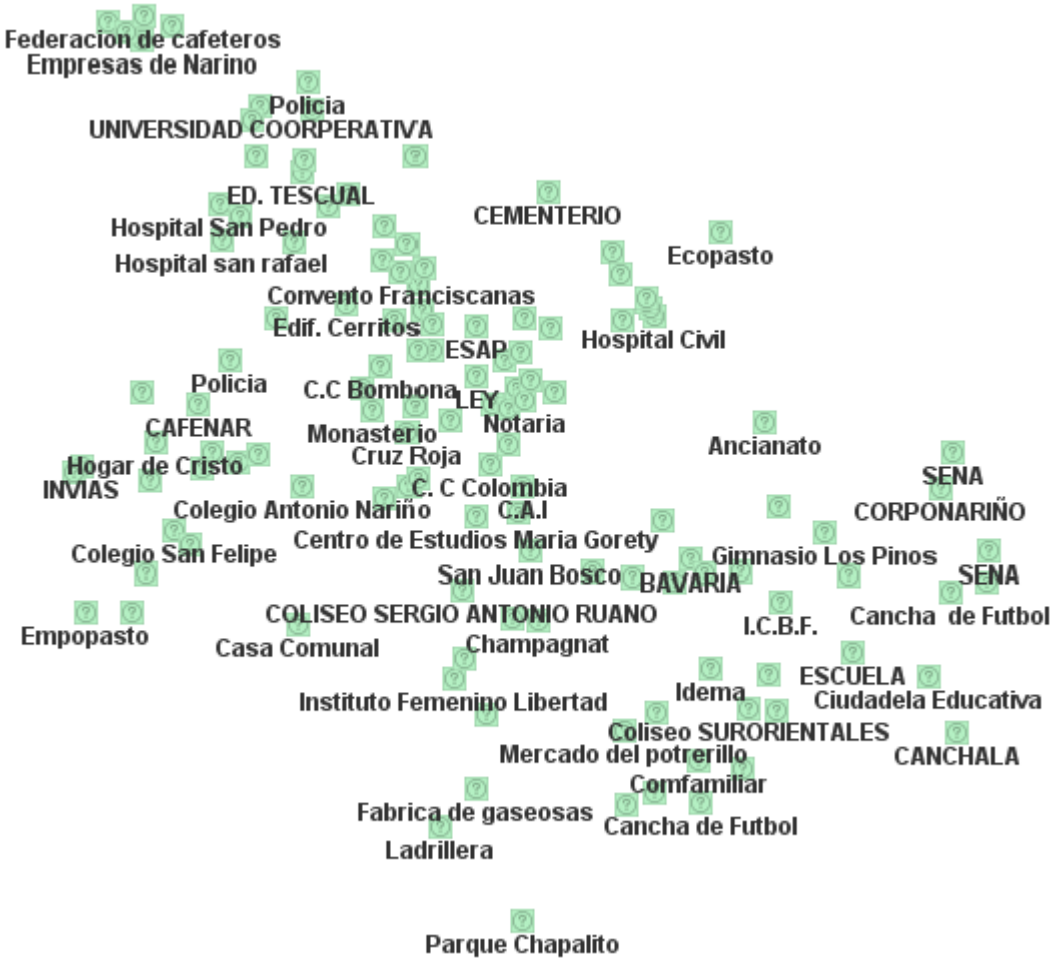
Los archivos de Autocad deben ser transformados en archivos apropiados para aplicaciones geográficas, para ello, se usa la herramienta CAD2Shape que entrega ShapeFiles de ESRI, una vez en este formato se realizaron operaciones de translación, además se agregaron atributos que no lograron interpretarse 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 127) 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 127. Apariencia de la capa Puntos.shp.



Comunas.shp

La capa comunas.shp (ver figura 128) 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 128. Apariencia de la capa Comunas.shp.

Barrios.shp

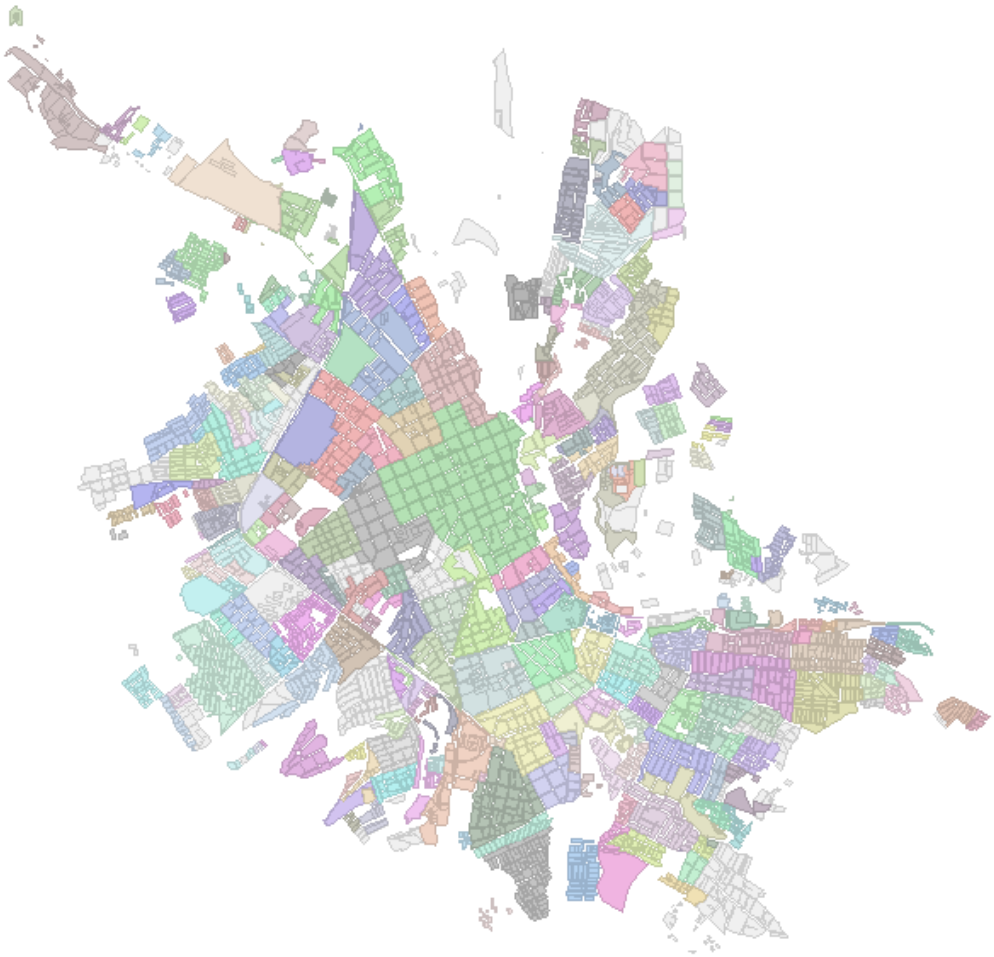
Barrios.shp (ver figura 129) es una geometría tipo polígono, representa los barrios de la ciudad de Pasto, contiene la información de nombre del barrio, y el número de comuna a la cual pertenece, cuenta con un total de 284 geometrías y un tamaño 173KB. Para la construcción de esta capa se tomó los polígonos 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 129. Apariencia de la capa Barrios.shp.

Manzanas.shp

La capa manzanas.shp (ver figura 130) 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 130. Apariencia de la capa Manzana.shp.



Malla.shp

La capa malla.shp (ver figura 131) 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 131. Apariencia de la capa Malla.shp.

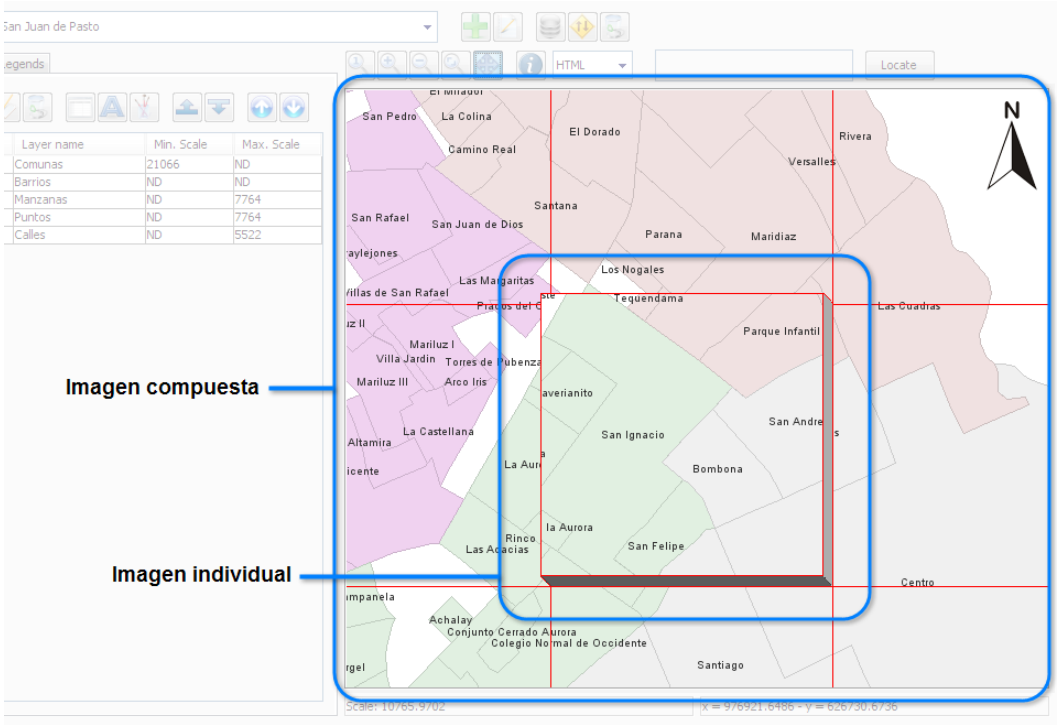
8.2. PRUEBAS SOBRE EL SERVIDOR DE CARTOGRAFÍA.

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.

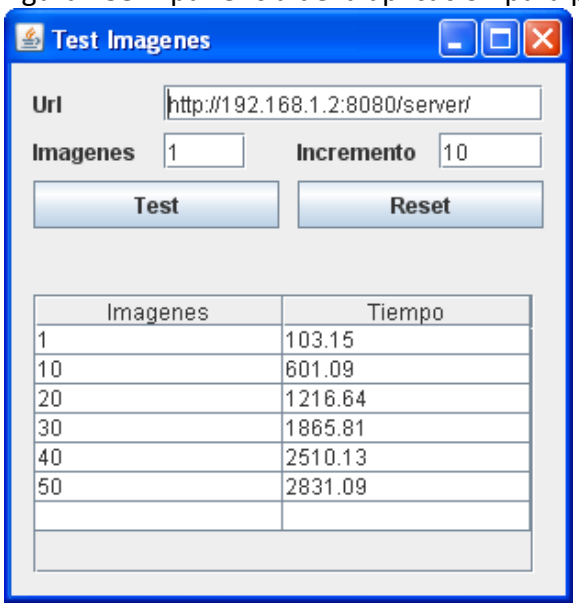
En adelante la imagen de cada región pequeña se denominará “imagen individual” y el conjunto de imágenes que representan la totalidad de la región solicita por el usuario se denominará “imagen compuesta”. Como se muestra en la siguiente figura.

Figura 132. Imagen compuesta e imagen individual.



8.2.1. 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 133.

Figura 133. Apariencia de la aplicación para pruebas individuales.

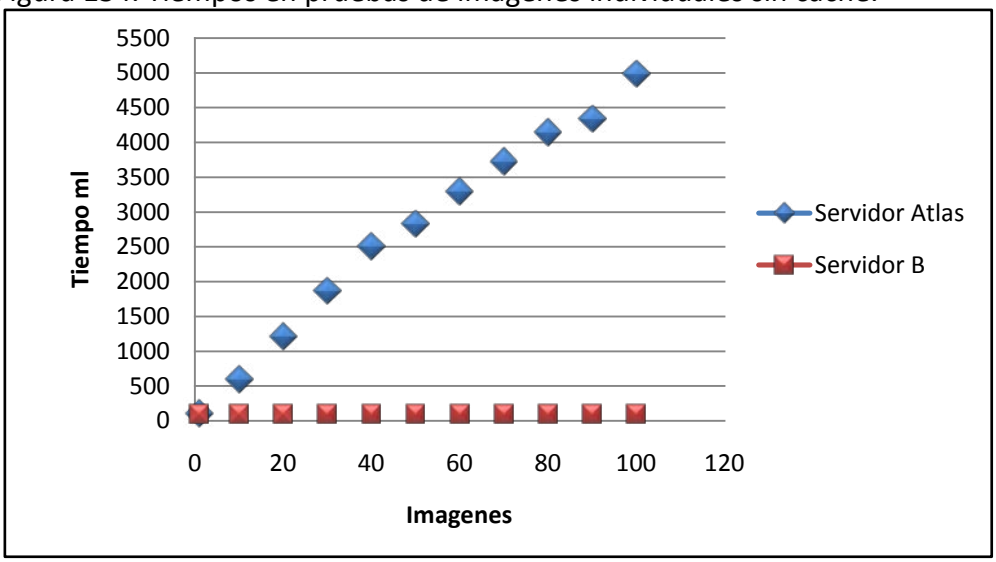


- **Prueba No 1.** Esta prueba consiste en enviar un número de peticiones de "imágenes individuales" aleatorias, simutaneamente al servidor Atlas y por cada una, medir el tiempo de respuesta del servidor con la opción de caché deshabilitada, la siguiente tabla muestra los resultados de esta prueba comparados 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 134 muestra los datos en una gráfica de dispersión.

Tabla 63. 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 134. 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.

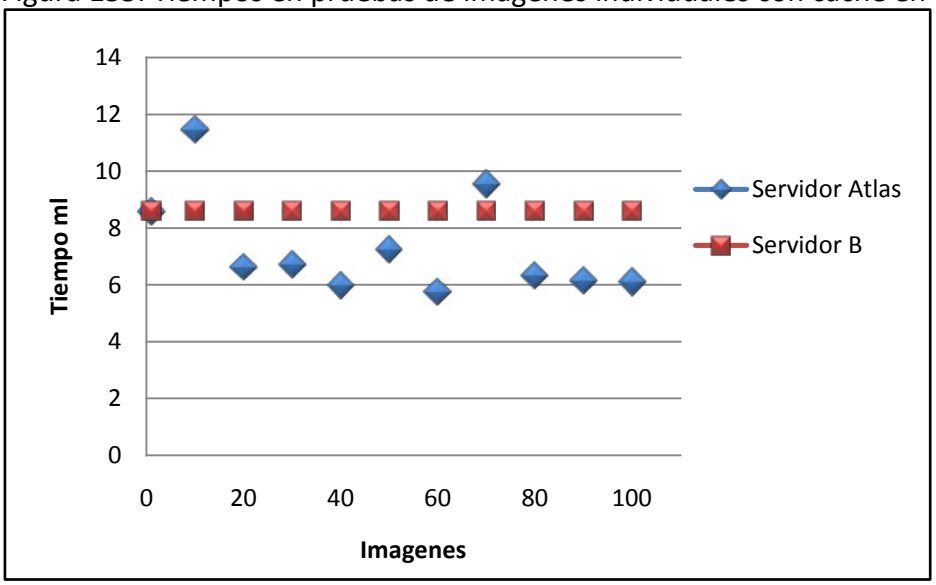
- **Prueba No 2**. Esta prueba consiste en enviar un número de peticiones de "imágenes individuales" aleatorias, simultáneamente al servidor Atlas y por cada una, medir el tiempo de respuesta del servidor con la opción de caché habilitada, para esta prueba en particular el caché se encuentra en formación. La siguiente tabla 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 135 muestra los datos en una gráfica de dispersión.

Tabla 64. 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 135. Tiempos en pruebas de imágenes individuales con caché en formación.



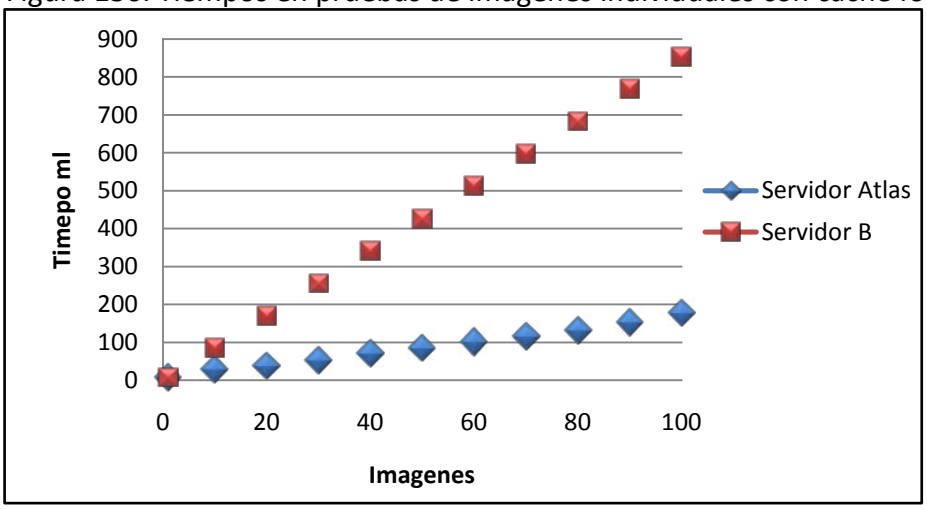
Al habilitar la opción de caché, que se encuentra en formación se observa como el tiempo de respuesta se mantiene prácticamente independiente del número de imágenes individuales simultáneas que se soliciten, De esta prueba se puede concluir que para mejorar el rendimiento del servidor Atlas es necesario 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 siguiente tabla muestra los resultados de esta prueba comparados con un servidor B que no soporta peticiones simultáneas, La figura 136 muestra los datos en una gráfica de dispersión.

Tabla 65. 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 136. 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.

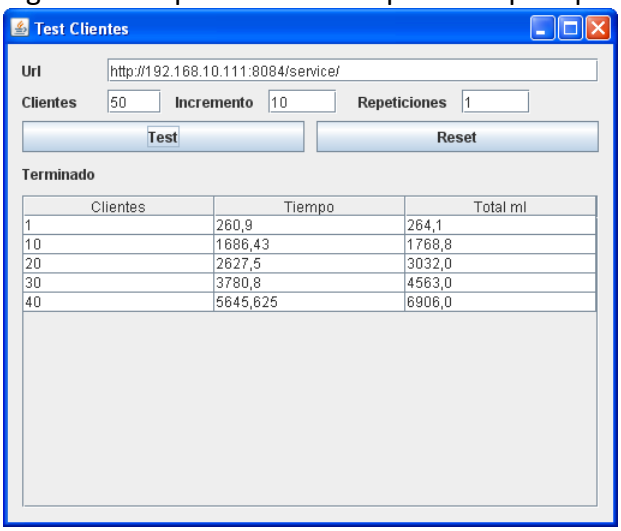
8.2.2. Pruebas de simulación de clientes. El objetivo de estas pruebas es comprobar el comportamiento del servidor Atlas al interactuar con clientes simultáneos, los cuales envían un conjunto de peticiones de imágenes individuales para formar una "imagen compuesta" (ver figura 132). Para esta prueba 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 137.

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 137. Apariencia de la aplicación para pruebas de clientes.



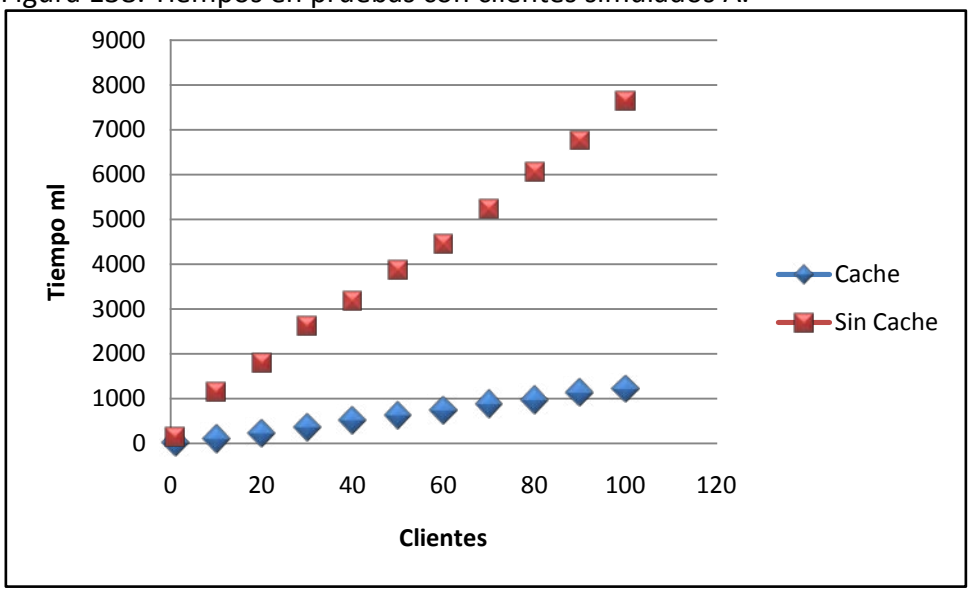
- **Prueba No 1.** Esta prueba consiste en enviar peticiones de "imágenes compuestas" aleatorias, simutaneamente al servidor Atlas, incrementado el número de clientes. Por cada imagen compuesta se mide el tiempo de respuesta del servidor con la opción de caché deshabilitada, En la siguiente tabla se muestran

los resultados de esta prueba, comparados con aquellos obtenidos al usar el servidor Atlas con el caché habilitado y formado. La figura 138, muestra los datos en una gráfica de dispersión.

Tabla 66. 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 138. 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

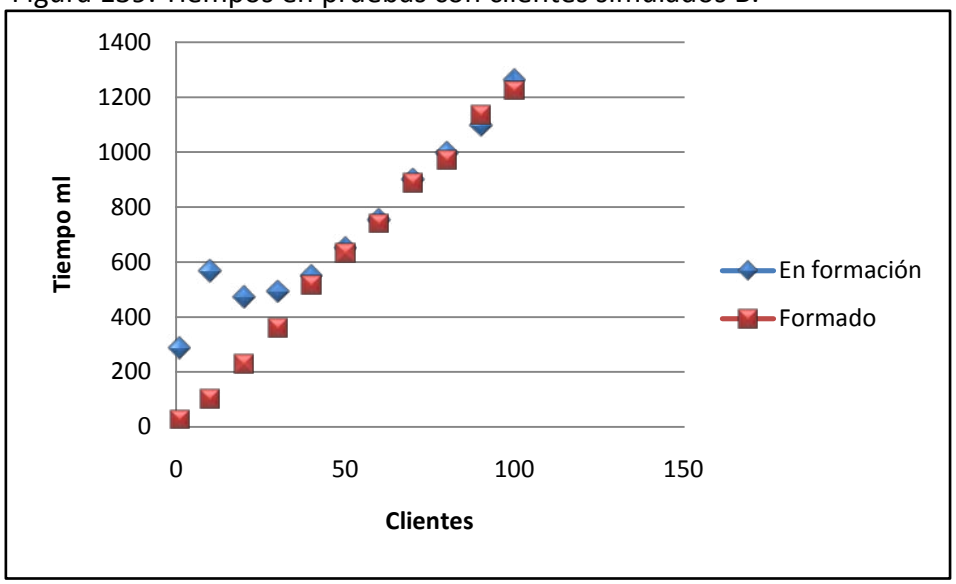
sustancialmente. De esta prueba se puede concluir que para manejar clientes el servidor Atlas necesita habilitar la opción de caché.

- **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 siguiente tabla se muestran los resultados de esta prueba, comparados con aquellos obtenidos al usar el servidor Atlas con el caché habilitado y formado. La figura 139, muestra los datos en una gráfica de dispersión.

Tabla 67. 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 139. 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.

8.3. 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 geocodificación 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,

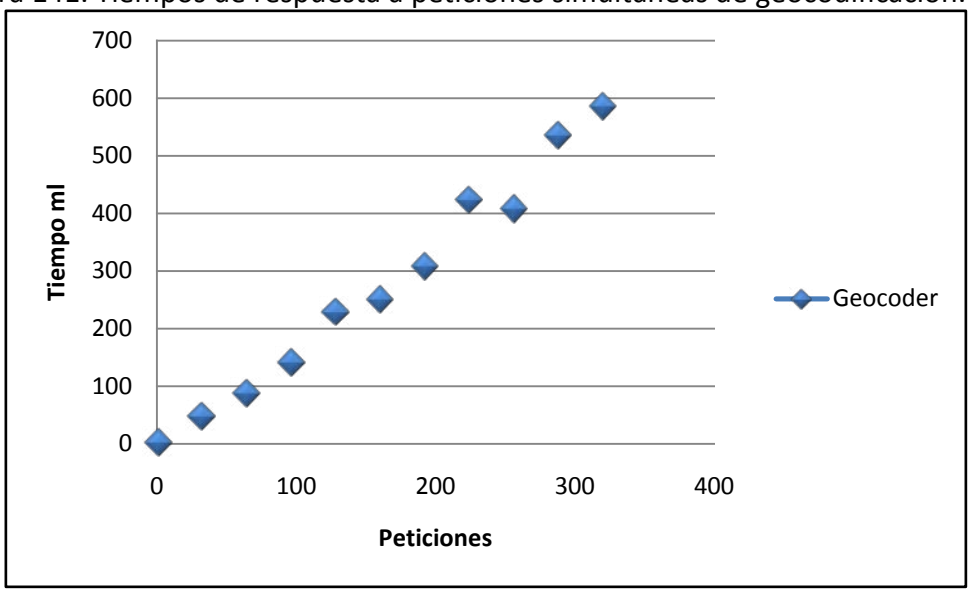
que sobre ellas se realice el proceso completo de geocodificación. Para esta prueba se desarrolló una aplicación que envía peticiones simultáneas al servidor de geocodificación (ver figura 140). Los tiempos obtenidos se muestran en la tabla 68, la figura 141 muestra los datos en una grafica de dispersión.

Figura 140. Apariencia de la aplicación para pruebas de geocodificación.

Tabla 68. 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 141. Tiempos de respuesta a peticiones simultáneas de geocodificación.



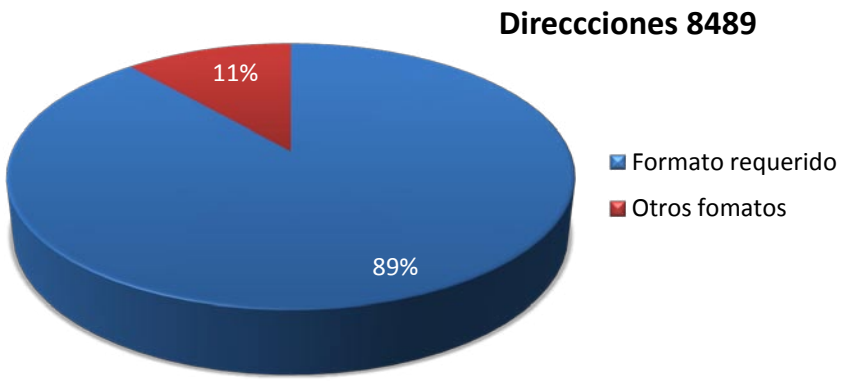
Gracias a la tecnología Java EE, se observa como la administración de peticiones simultáneas del servidor Atlas, en conjunto con la eficiencia del Atlas Street Crossing Geocoder, permiten al servidor Atlas entregar resultados de geocodificación en tiempos menores a un segundo para más de 300 peticiones 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 siguiente tabla. La figura 142 muestra los datos en una gráfica circular.

Tabla 69. Porcentajes de direcciones de la ciudad.

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

Figura 142. Porcentajes de direcciones de la ciudad.



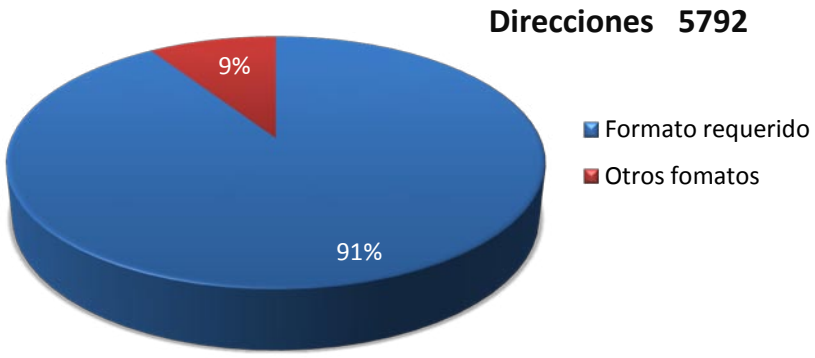
Dado que el algoritmo Atlas Street Crossing Geocoder se diseño e implemento con base en la nomenclatura oficial de la ciudad de Pasto se ve que la cobertura del servidor de geocodificación para el sector empresarial de la ciudad es satisfactoria.

- **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 siguiente tabla. La figura 143 muestra los datos en una gráfica circular.

Tabla 70. Porcentajes de direcciones comuna centro.

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

Figura 143. Porcentajes de direcciones comuna centro.



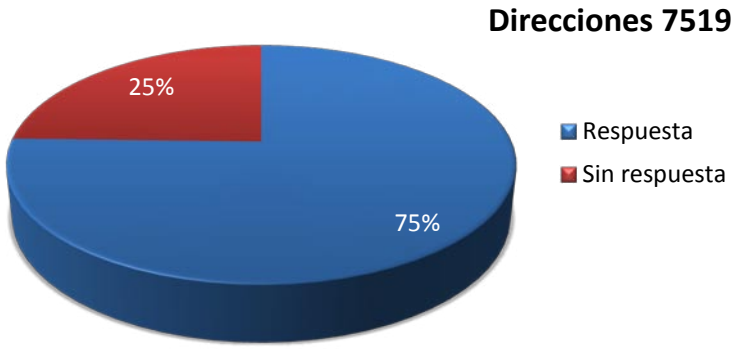
Dado que el algoritmo Atlas Street Crossing Geocoder se diseño e implemento con base en la nomenclatura oficial de la ciudad de Pasto se ve que la cobertura del servidor de geocodificación es mayor en la comuna centro y centro sur de la ciudad de Pasto, ya que en ella predomina la nomenclatura oficial 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 siguiente tabla. La figura 142 muestra los datos en una gráfica circular.

Tabla 71. Direcciones geocodificadas en la ciudad.

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

Figura 144. Direcciones geocodificadas en la ciudad.



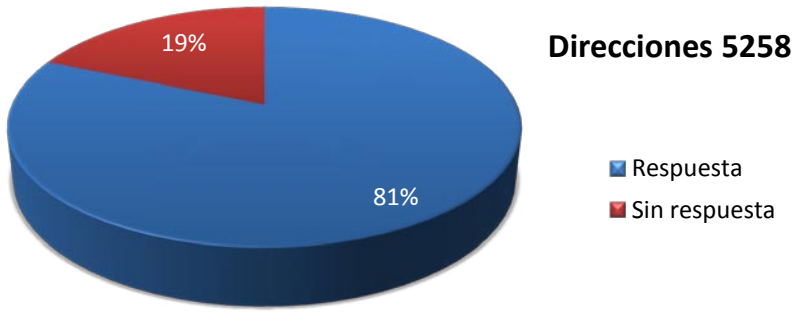
Dado que las peticiones enviadas al servidor con direcciones ubicadas en diferentes lugares de la ciudad de Pasto cumplen el formato requerido por el geocoder, se puede concluir que la eficacia de este, es adecuada para la ciudad de Pasto en general. Cabe señalar que la eficacia del geocoder depende 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 siguiente tabla. La figura 145 muestra los datos en una gráfica circular.

Tabla 72. Direcciones geocodificadas en la comuna centro

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

Figura 145. 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.

CONCLUSIONES

Como resultado de este proyecto se cuenta con la primera versión de Atlas una herramienta de cartografía Web y geocodificación compuesta por un servidor de cartografía Web, un servidor de geocodificación, una herramienta gráfica para administrar el contenido de estos servidores, y bibliotecas para el desarrollo de software compatible con dichos servidores.

Se planteó e implementó el algoritmo de geocodificación titulado “Atlas street crossing”, creado para resolver direcciones en la nomenclatura de la zona centro de la ciudad de San Juan de Pasto y demostrar las capacidades del servidor de geocodificación.

El análisis de herramientas existentes afines al objetivo del proyecto, permitieron que Atlas tenga diferentes características deseables como, facilidad de administración e instalación, cumplimiento de estándares internacionales, repuesta adecuada a la concurrencia y extensibilidad.

Atlas cuenta con bibliotecas para el desarrollo de aplicaciones hibridas, que permiten su extensibilidad y puede desempeñarse en una amplia gama de ambientes, como intranet, mediante aplicaciones J2SE, Internet, mediante Web 2.0 y Javascript, e inclusive, redes de cobertura urbana mediante Java ME.

El hecho de que Atlas haya sido desarrollado bajo tecnologías Web 2.0 mejora considerablemente la experiencia del usuario, el rendimiento y eficiencia de las aplicaciones desarrolladas con la biblioteca Atlas para navegadores Web.

Al almacenar los datos geográficos en compatibilidad con Postgis, Atlas permite la formación de interesantes arquitecturas, compuestas por diferentes herramientas para la edición del material geográfico, Postgis para realizar consultas espaciales y Atlas para llevar este material a la Web.

Atlas se construyó exclusivamente con herramientas de software libre.

La liberación de la herramienta bajo licencia GPL, permite que esta se encuentre a disposición de un público más amplio, como pequeñas y medianas empresas y la comunidad académica, a nivel regional y nacional. Al tratarse de software libre, también se beneficia de herramientas y bibliotecas de gran calidad disponibles bajo esta licencia.

Atlas cuenta con una biblioteca de desarrollo creada completamente con Javascript, se posibilita la creación de aplicaciones que integren los contenidos del servidor Atlas con servidores de diferentes tecnologías, como PHP, .Net, Java, etc.

Al ser Atlas una aplicación J2EE obtiene un mayor rendimiento en comparación con otras aplicaciones Web, especialmente si se compara con otras alternativas como CGI. Además lleva todas las ventajas de la programación orientada a objetos de Java a la Web.

La implementación de Atlas para dispositivos móviles depende explícitamente de la máquina virtual de java implementada en el dispositivo.

Gracias al uso de tecnologías de base de datos y J2EE, el servidor de cartografía Web Atlas, presenta un buen rendimiento, inclusive al gestionar peticiones simultáneas provenientes de diferentes clientes. El rendimiento mejora sustancialmente al habilitar la opción de formar un caché de imágenes en el servidor, ya que así, las peticiones realizadas sobre la misma imagen solo deben dibujarse una vez, las subsecuentes peticiones se limitan a entregar la imagen almacenada en dicho caché.

Gracias al uso de tecnologías de base de datos y J2EE del servidor de geocodificación Atlas y las técnicas de referencia lineal usadas por el algoritmo Atlas street crossing, se encontró, que estos ofrecen tiempos de respuesta inferiores a un segundo, inclusive con alta concurrencia.

RECOMENDACIONES

Como punto de partida para trabajos futuros se considera lo siguiente.

Ofrecer un servicio de extensión a la comunidad con base en Atlas, a través de la Universidad de Nariño.

Diseñar e implementar las adecuaciones que permitan la adopción de otros estándares internacionales como, Web Feature Service del OGC para compartir objetos geográficos.

Diseñar e implementar más algoritmos de geocodificación para el servidor Atlas, por ejemplo, un geocoder especializado en tratar descripciones de lugares con nomenclatura en formato de manzanas.

Diseñar e implementar las adecuaciones que permitan al servidor Atlas entregar recursos multimedia sobre distintos lugares de la ciudad.

Diseñar e implementar las adecuaciones que permitan la importación desde fuentes de datos adicionales como, ArcSDE, Oracle Spatial, MySQL, Microsoft SQL Server 2008 Spatial.

Liberar, compartir y difundir las futuras versiones de Atlas como herramienta de cartografía Web y geocodificación.

Mejorar la precisión, calidad, diversidad y cobertura de la cartografía digital sobre la ciudad de San Juan de Pasto.

Investigar sobre algoritmos que permitan encontrar el camino óptimo entre 2 puntos de la malla vial de una ciudad.

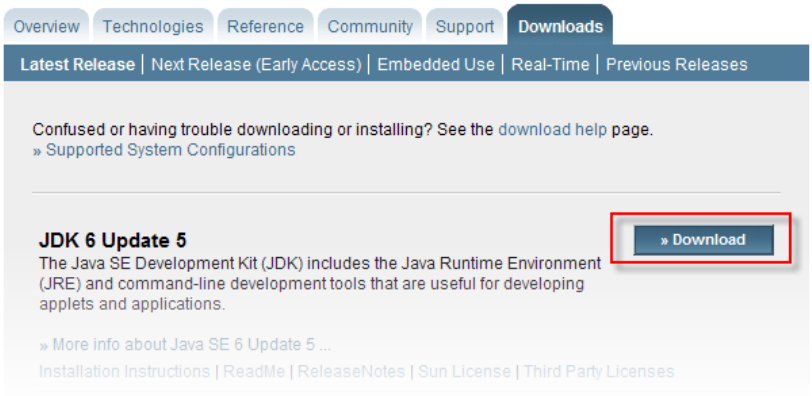
ANEXOS

A1. GUÍA DE INSTALACIÓN DE PRERREQUISITOS.

Instalación de Java SE Development Kit.

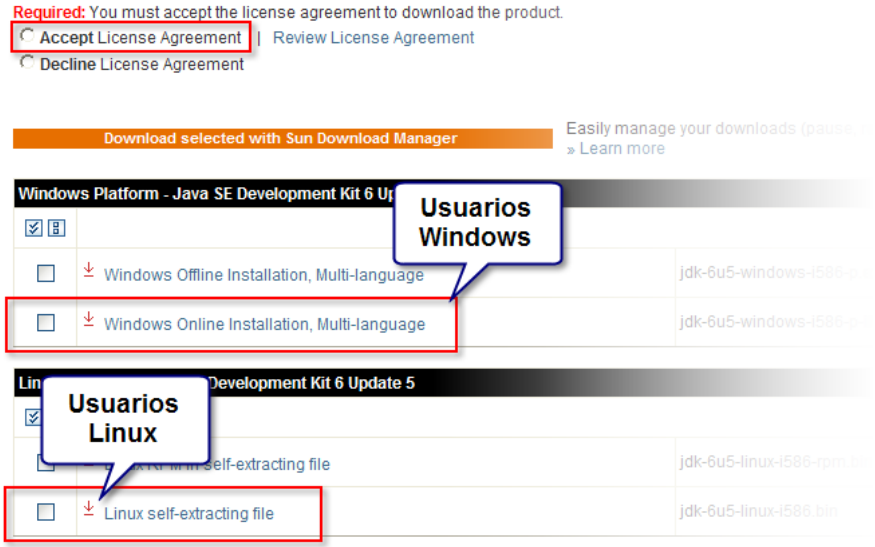
- Se debe ingresar en la dirección: http://java.sun.com/javase/downloads/index.jsp
- Una vez ahí debe seguirse el link que se muestra en la figura 146.

Figura 146. Home del sitio Web sobre Java en la página de Sun Microsystems.



Una vez ahí, se debe aceptar la licencia y descargar el archivo apropiado según el sistema operativo como se muestra en la figura 147.

Figura 147. Página Web de descargas de J2SDK.

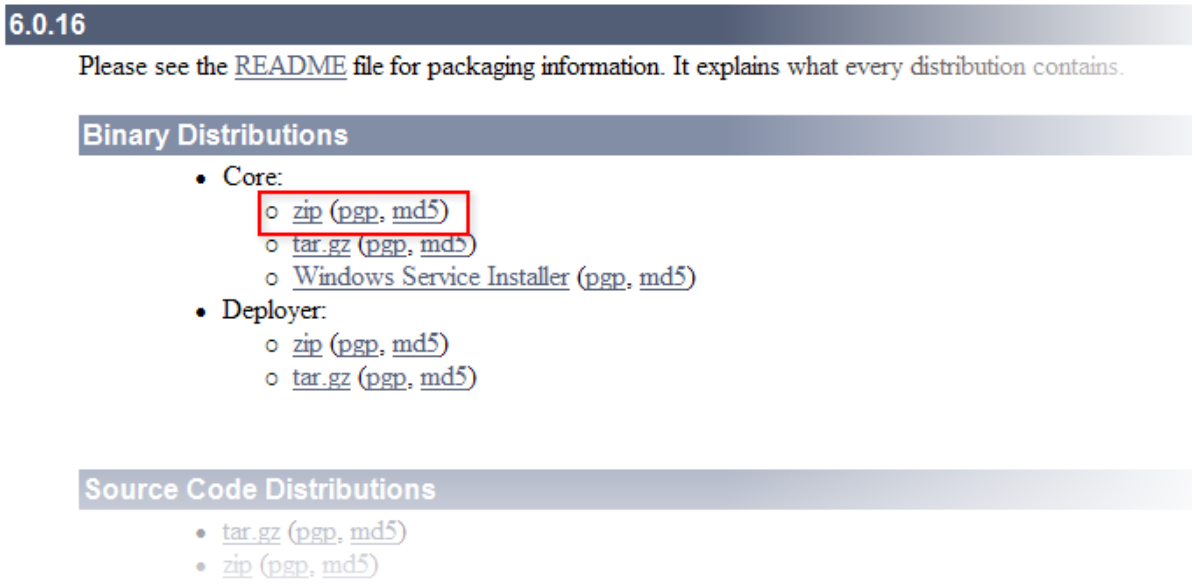


Tanto el Windows como el Linux el archivo descargado es un asistente de instalación, así que basta con seguir los pasos.

Instalación de Apache Tomcat.

- Se debe ingresar en esta dirección http://tomcat.apache.org/download-60.cgi
- Una vez ahí se debe seleccionar el link mostrado en la figura.

Figura 148. Página Web de descarga del Apache Tomcat.



- El contenido del archivo debe descomprimirse en algún lugar de sistema de archivos.

  Bajo Windows C:\Archivos de programa\Apache\apache-tomcat-6.0.16
  Bajo Linux /usr/local/apache/apache-tomcat-6.0.16

- En el directorio de instalación, en la carpeta conf, se encuentra el archivo tomcat-users.xml, el contenido de este archivo debe quedar como indica la figura 149, en la línea 5, los campos username y password, se colocan a modo de ejemplo.

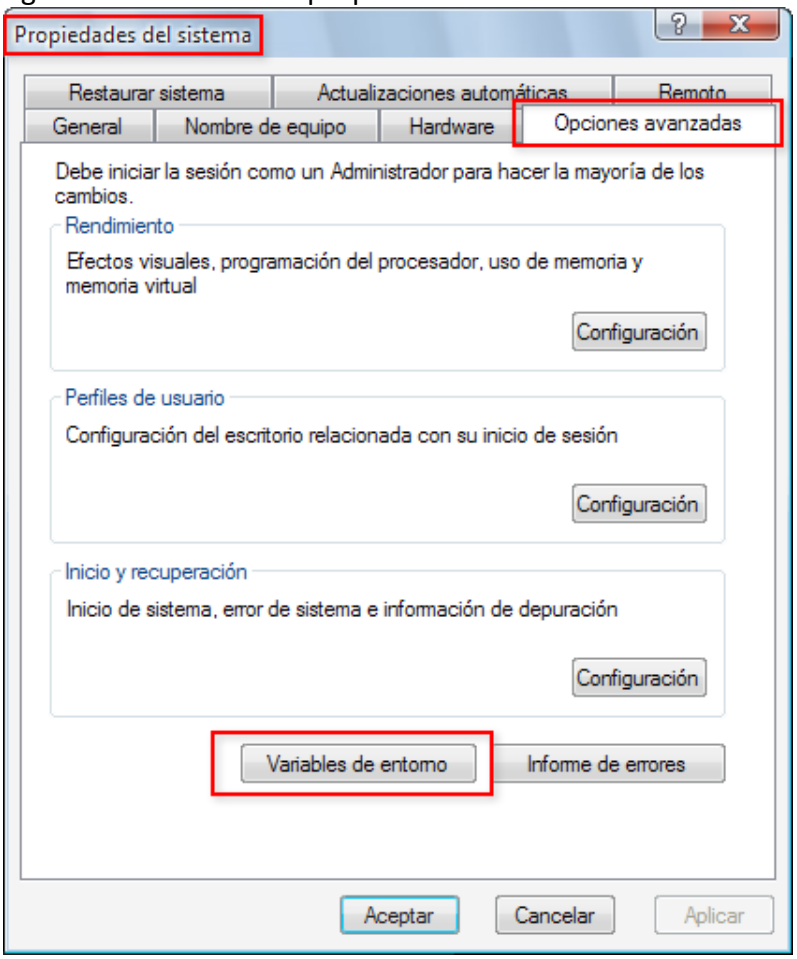
Figura 149. Contenido del archivo users.xml.

```xml
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="admin"/>
  <role rolename="manager"/>
  <user username="admin" password="admin" roles="admin,manager"/>
</tomcat-users>
```

- Luego, se procede a configurar las variables de entorno.

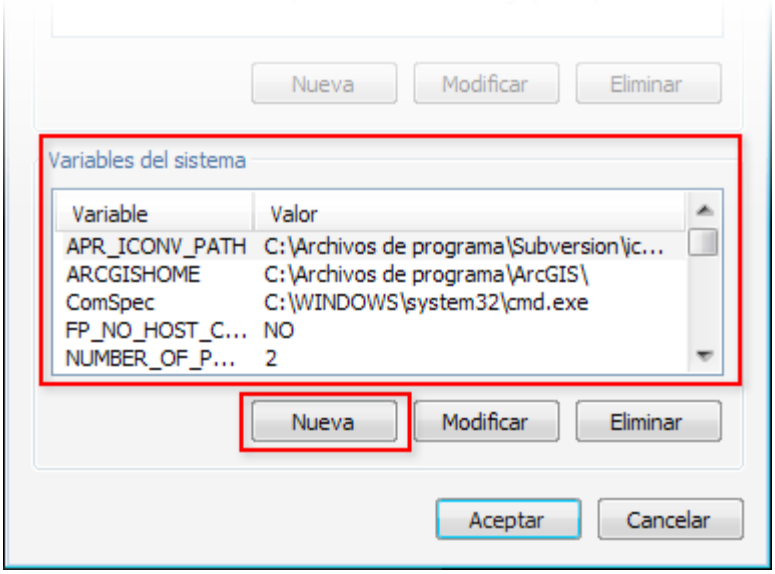
  Bajo Windows. Se debe ingresar a la opción propiedades del sistema, en el panel de control, una vez ahí, debe accederse a la pestaña, opciones avanzadas y a la opción variables de entorno, como se muestra en la figura.

Figura 150. Ventana de propiedades del sistema en Windows.



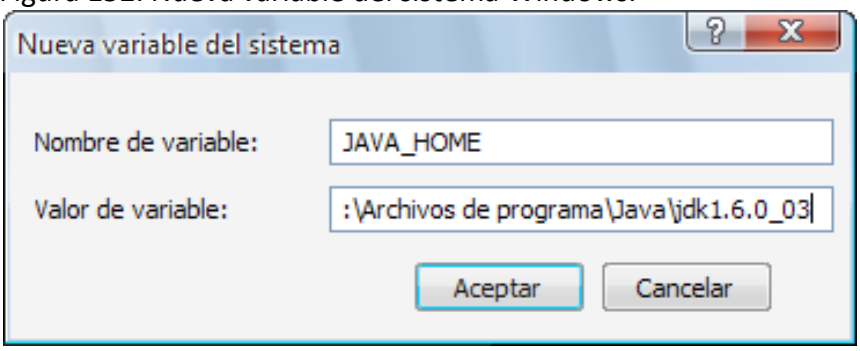
En la ventana de variables de entorno, debe seleccionar la opción nueva bajo la sección variables del sistema.

Figura 151. Variables de entorno del sistema Windows.



Aquí, el nombre de la variable debe especificarse como JAVA_HOME y debe apuntar a la ruta de la J2SDK, si la instalación de esta se llevo a cabo con las opciones por defecto, la ruta será C:\Archivos de programa\Java\jdk1.6.0_03, como indica la figura 152.

Figura 152. Nueva variable del sistema Windows.



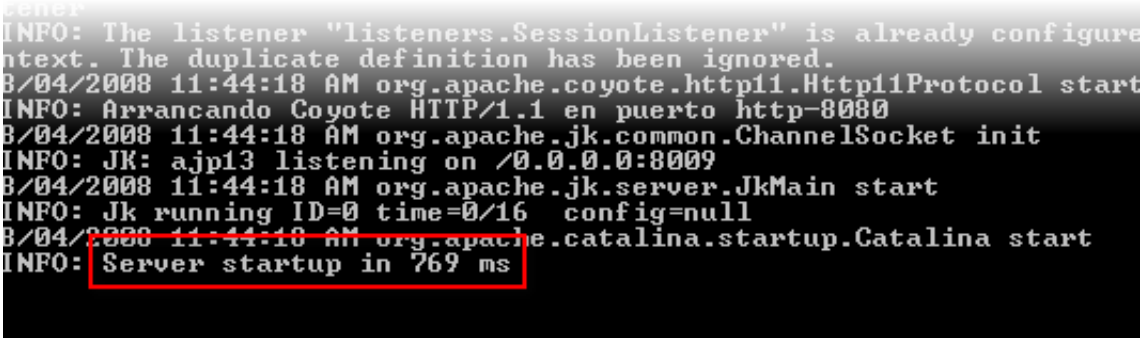
Bajo Linux. Desde un terminal, se establece la variable escribiendo el comando siguiente. Esta asignación es temporal, y debe repetirse cada vez que se inicie el sistema, existen diferentes alternativas para evitar esto que no se discuten en este documento.

```
[root@localhost ~]# export JAVA_HOME=<Ruta de instalación de J2DSK>
```

- Iniciando el servicio. En el directorio /bin, dentro de la ruta de instalación se encuentran los archivos startup.bat y startup.sh debe ejecutarse el .bat para Windows y el .sh para Linux.

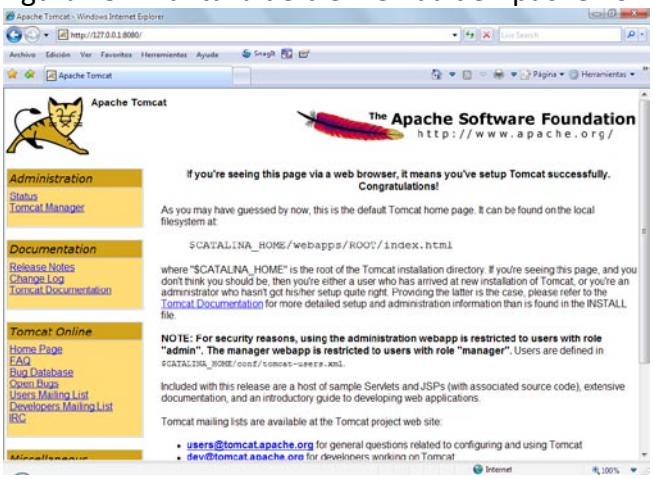
  Si el proceso fue correcto, debe obtenerse una pantalla como la que muestra la figura.

Figura 153. Éxito en la iniciación de Apache Tomcat.



Y al ingresar la dirección http://127.0.0.1:8080 en un navegador debe obtenerse pantalla como la siguiente.

Figura 154. Pantalla de bienvenida de Apache Tomcat.



Instalación de PostgreSQL 8.2

- **Bajo Windows.**

  Requisitos Previos. Antes de instalar PostgreSQL™, se debe comprobar si se tiene el siguiente software.

  Un descompresor de ficheros Zip: Windows™ XP los maneja automáticamente;.
  Sistema de Archivos NTFS: Es necesario que la partición donde se instala postgreSQL™ se encuentre en este formato.

  Instalación.

  - Descargar el instalador de postgresql™ 8.2 "postgresql-8.2.3-1.zip" desde http://www.postgresql.org/download.
  - Descomprimir el archivo .zip descargado.
  - Entrar a la carpeta extraída postgresql-8.2.3-1.
  - Ejecutar el instalador postgresql-8.2.msi y seguir los pasos del instalador.

- **Bajo Linux.**

  Se debe ingresar a la dirección http://www.postgresql.org/ftp/source/v8.2.4/ y obtener el archivo postgresql-8.2.4.tar.gz.

  Se descomprime el archivo con el siguiente comando:

```
[root@localhost ~]# tar -zxf postgresql-8.2.5.tar.gz -C /opt/
```

  En el directorio donde se descomprime el archivo:

```
[root@localhost ~]# cd /opt/postgresql-8.2.5/
```

  Se crea un archivo de compilación llamado compile.sh para especificar las opciones de compilación y preservarlas en caso de error, el contenido de este archivo será como se indica a continuación.

```
LDFLAGS=-lstdc++ ./configure \
--prefix=/usr/local/pgsql \
--with-perl \
--with-python \
--with-krb5 \
--with-openssl
```

Se cambia los atributos para hacerlo ejecutable:

```
[root@localhost postgresql-8.2.4]# chmod 755 compile.sh
```

Luego se ejecuta el script:

```
[root@localhost postgresql-8.2.4]# ./compile.sh
```

Se compila el paquete:

```
[root@localhost postgresql-8.2.4]# make
```

Se instala el paquete:

```
[root@localhost postgresql-8.2.4]# make install
```

Una vez todos los pasos anteriores han sido ejecutados sin ningún problema, se procede a realizar los últimos ajustes:

Se debe crear un usuario para la administración de postgreSQL™:

```
[root@localhost postgresql-8.2.4]# adduser postgres
```

Se asigna una contraseña al usuario:

```
[root@localhost postgresql-8.2.4]# passwd postgres
```

Se crea el directorio “data”:

```
[root@localhost postgresql-8.2.4]# mkdir /usr/local/pgsql/data
```

Al directorio, se le asigna su nuevo propietario:

```
[root@localhost postgresql-8.2.4]# chown postgres  /usr/local/pgsql/data/
```

Ya que se trabaja con postgreSQL™, el login debe realizarse con el usuario adecuado: (postgreSQL no funciona desde una cuenta de súper usuario).

```
[root@localhost postgresql-8.2.4]# su – postgres
```

Ahora se a inicializa el motor de base de datos:

```
[postgres@localhost ~]# /usr/local/pgsql/bin/initdb -D /usr/local/pgsql/data/ &
```

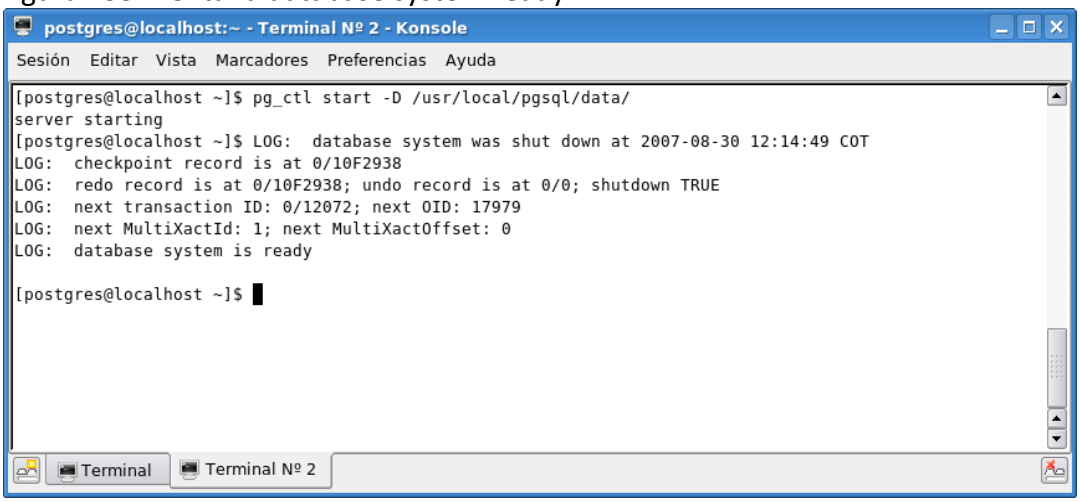
Finalizado el paso anterior, se observa un mensaje que indica que el servidor de bases de datos está listo para trabajar.

Se procede a iniciar el servidor:

```
[postgres@localhost ~]# /usr/local/pgsql/bin/pg_ctl start -D /usr/local/pgsql/data/ &
```

Todo habrá salido correctamente si se aprecia la figura 155.

Figura 155. Ventana database system ready.

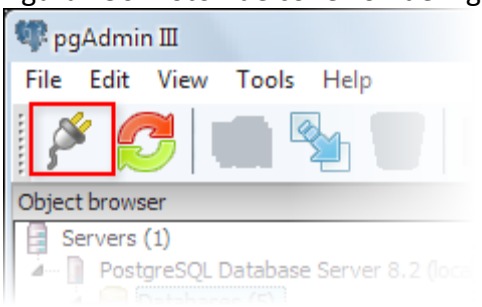


Creación de la base de datos del sistema.

Para crear la base de datos del sistema se usa la herramienta PgAdmin III, aunque también puede hacerse con otras herramientas o inclusive desde la consola del sistema.

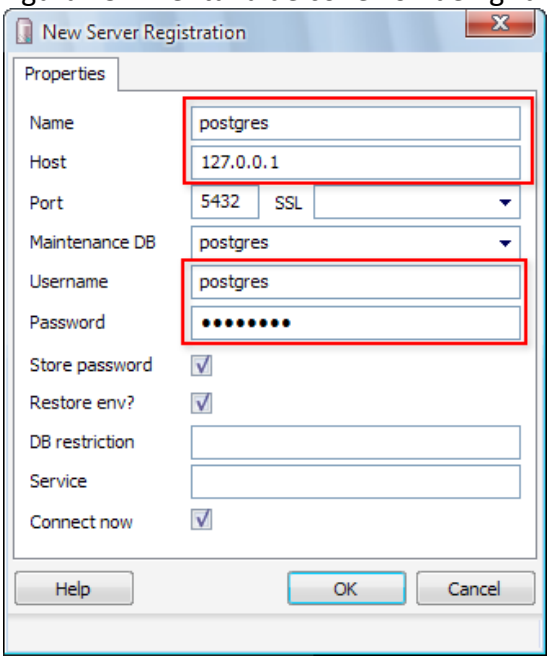
Una vez iniciado el PgAdmin III hay que establecer conexión con el sistema de bases de datos, para ello debe hacerse clic sobre el botón marcado en la figura 156.

Figura 156. Botón de conexión del PgAdmin III.



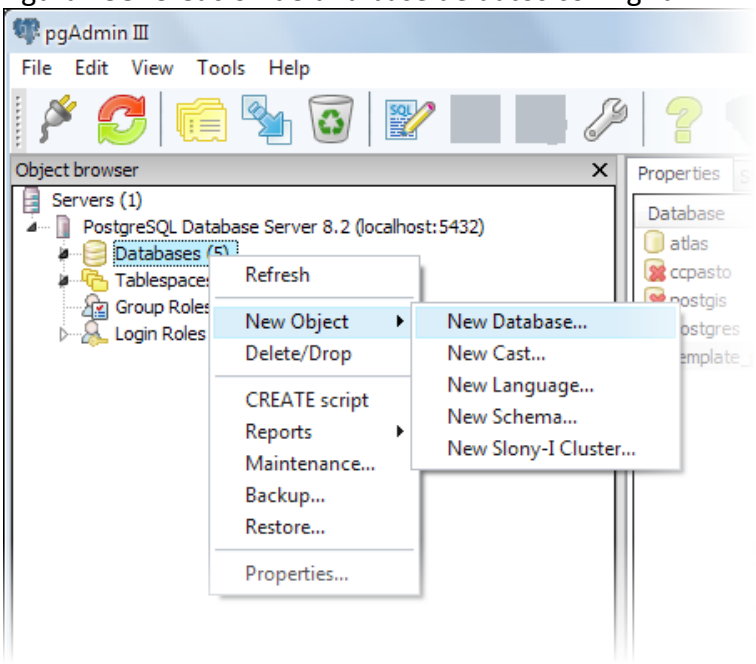
En el formulario que se despliega deben ingresarse los datos del usuario del sistema PostgreSQL, modificando los campos que se muestran en la figura 157.

Figura 157. Ventana de conexión de PgAdmin III.



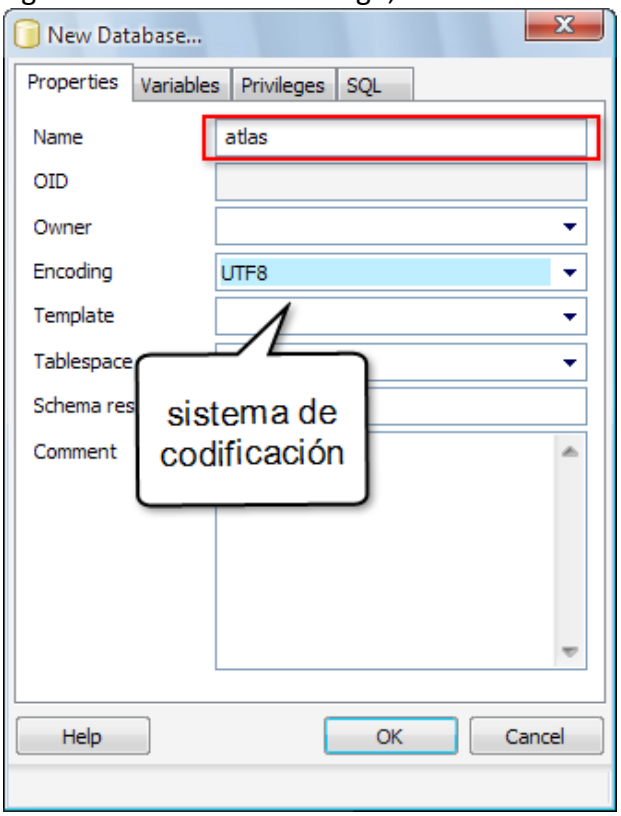
Luego de ingresar al sistema debe hacerse clic derecho sobre la opción “Databases” y seleccionar las opciones que se muestran en la figura 158.

Figura 158. Creación de una base de datos con PgAdmin III.



En el fomulario que aparece, el sistema de codificación debe establecerse como UTF8 y para este ejemplo se usará el nombre “atlas”.

Figura 159. Cuadro de diálogo, nueva base de datos.

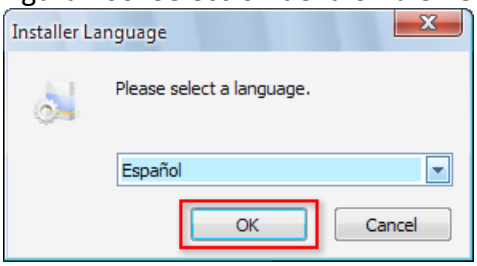


A2. GUÍA DE INSTALACIÓN DE LAS HERRAMIENTAS ATLAS.

Para Windows debe ejecutarse el instalador atlas_setup.exe.

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

Figura 160. Selección de idioma en el instalador de Atlas.

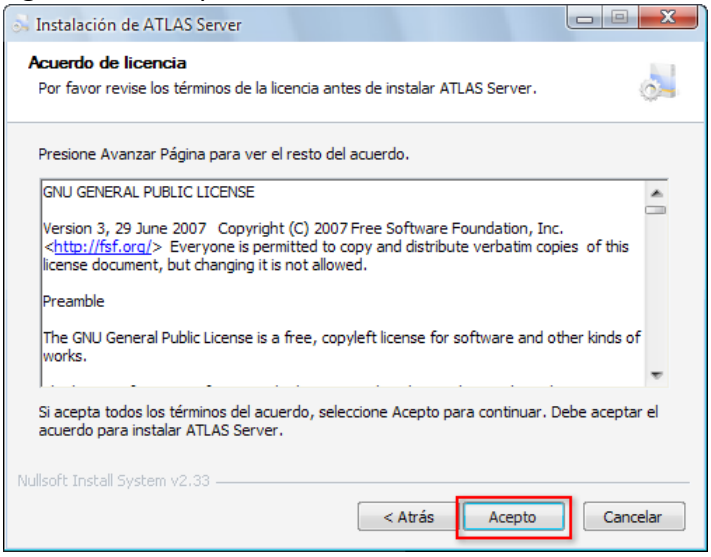


A continuación el instalador presenta una pantalla de bienvenida como la siguiente.

Figura 161. Pantalla de bienvenida en el instalador de Atlas.

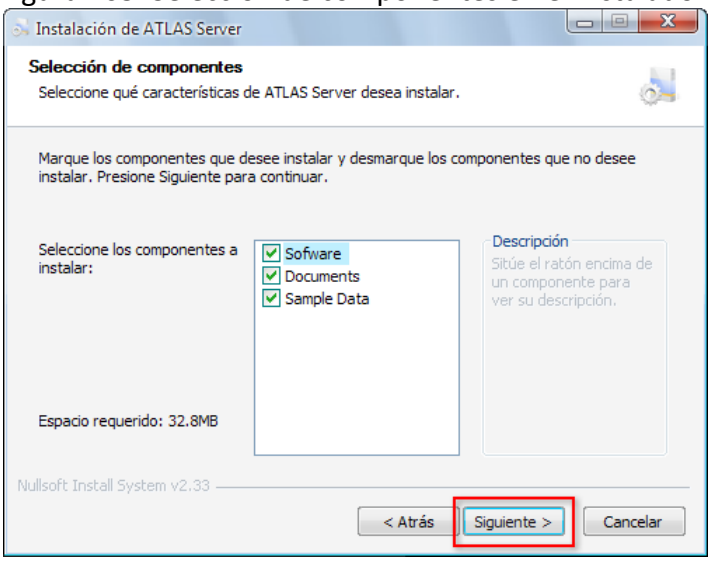
Luego, debe aceptarse los términos de la licencia GNU, como se muestra en la figura.

Figura 162. Aceptación de licencia en el instalador de Atlas.



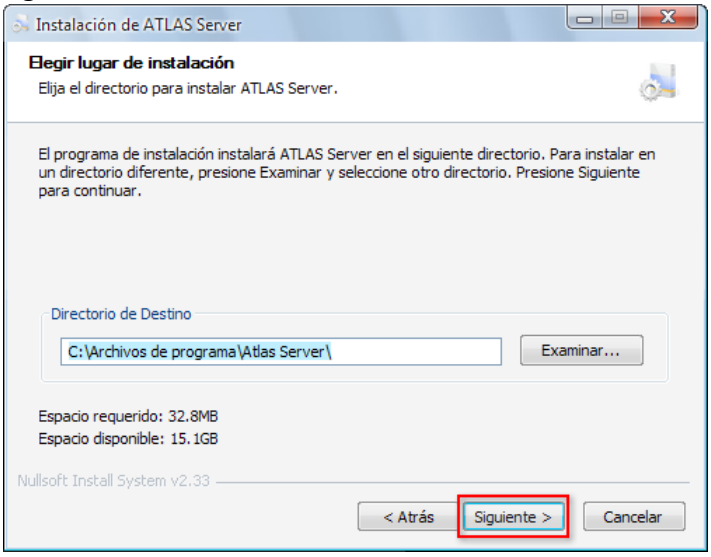
Luego debe seleccionarse los componentes a instalar, para este caso se instalarán todos, como se muestra en la figura.

Figura 163. Selección de componentes en el instalador de Atlas.



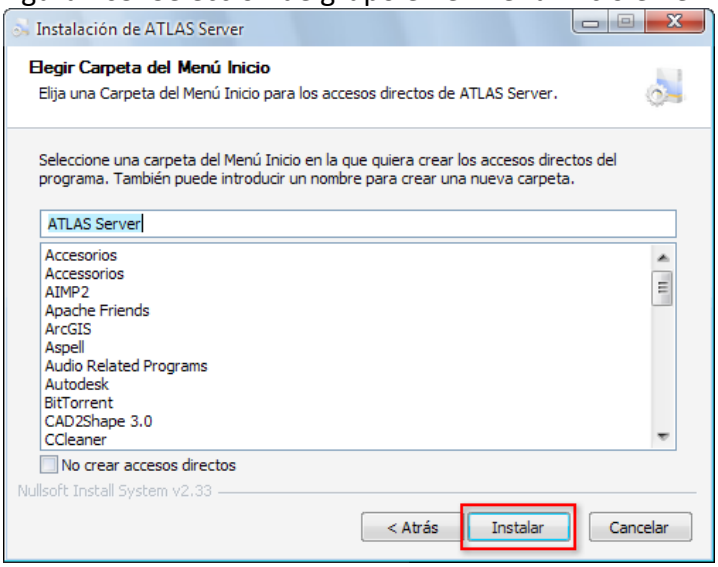
A continuación se selecciona la ruta para la instalación de los archivos, esta puede ser modificada, sin embargo, en este caso se tomará la opción por defecto, como se muestra en la figura 164.

Figura 164. Selección de ruta instalación en el instalador de Atlas.



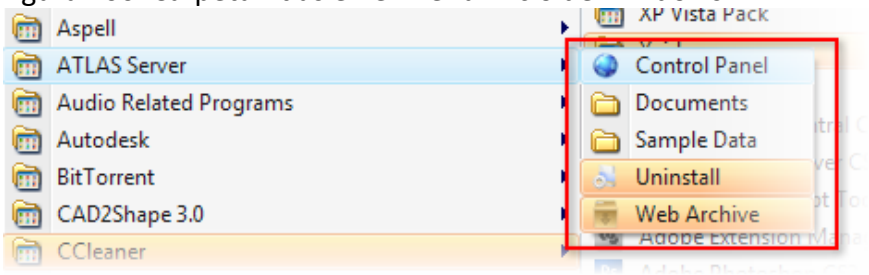
A continuación se debe seleccionar el grupo de menú inicio donde se crearán los accesos directos, como se muestra en la figura.

Figura 165. 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.

Figura 166. Carpeta Atlas en el menú inicio de Windows.



Los usuarios Linux deben descomprimir el archivo atlas.tar.gz y ubicarse sobre el directorio raíz, una vez ahí deben ejecutar.

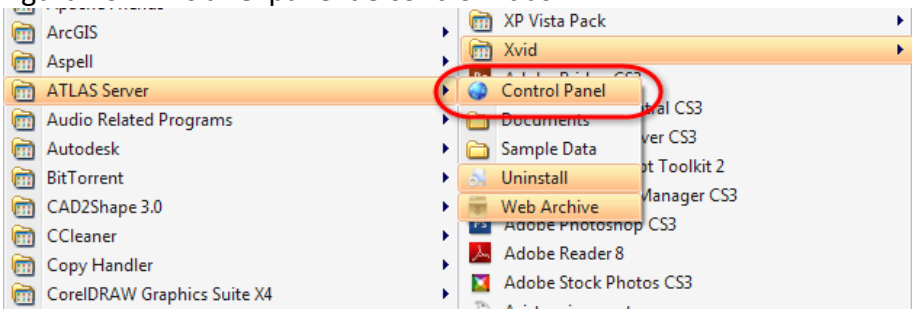
```
[root@localhost postgresql-8.2.4]# java –jar ./admin.jar
[root@localhost postgresql-8.2.4]# java –jar ./wartool.jar
```

A3. GUÍA DE INSTALACIÓN DEL SERVIDOR ATLAS.

Esta guía cubre los pasos necesarios para instalar el servidor Atlas. Una vez completada la guía de instalación del pre requisitos, se debe iniciar el panel de control Atlas.

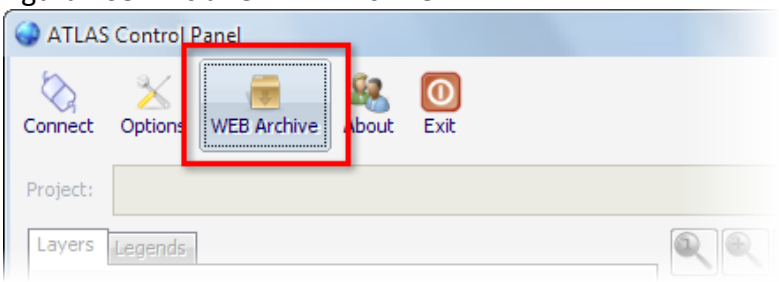
Bajo Windows, debe iniciarse la aplicación desde el menú inicio como se muestra en la figura.

Figura 167. Iniciar el panel de control Atlas.



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

Figura 168. Iniciar el WEB Archive.



La ventana que se presenta (ver figura 169) 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.

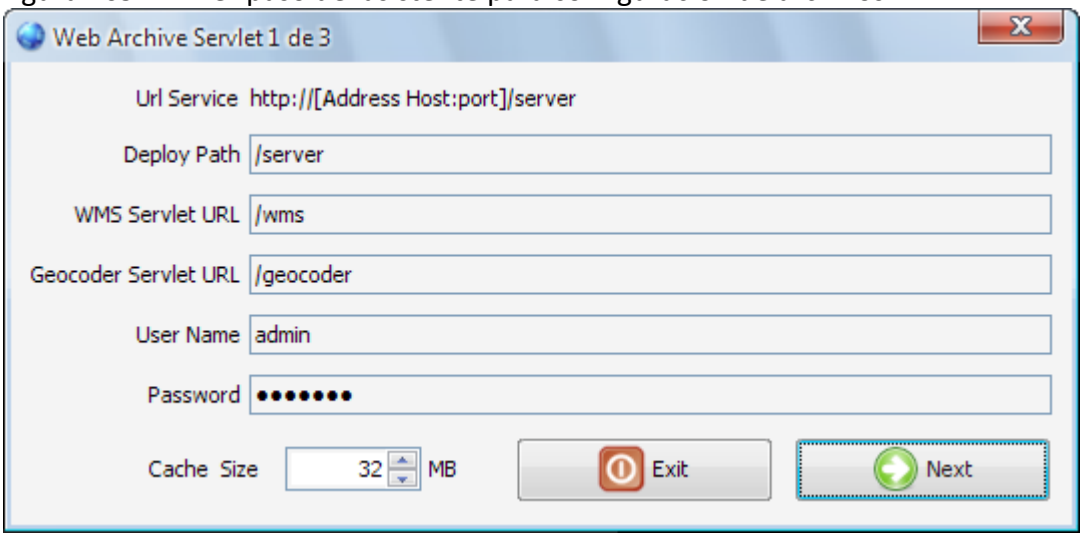
Los campos **"WMS Servlet URL"** y **"Geocoder Servlet URL"**, son informativos, y muestran las rutas de los servidores de catografia y geocodificación respectivamente.

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 169. 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 170.

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.

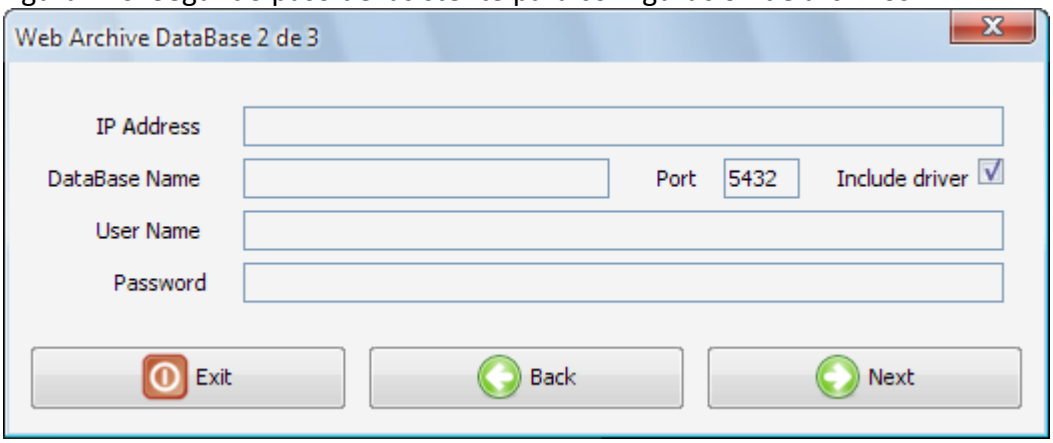
El campo **"DataBase Name"**, corresponde al nombre de la base de datos donde se instalarán las tablas del servidor Atlas, este campo debe ser diligenciado.

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 170. 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 171).

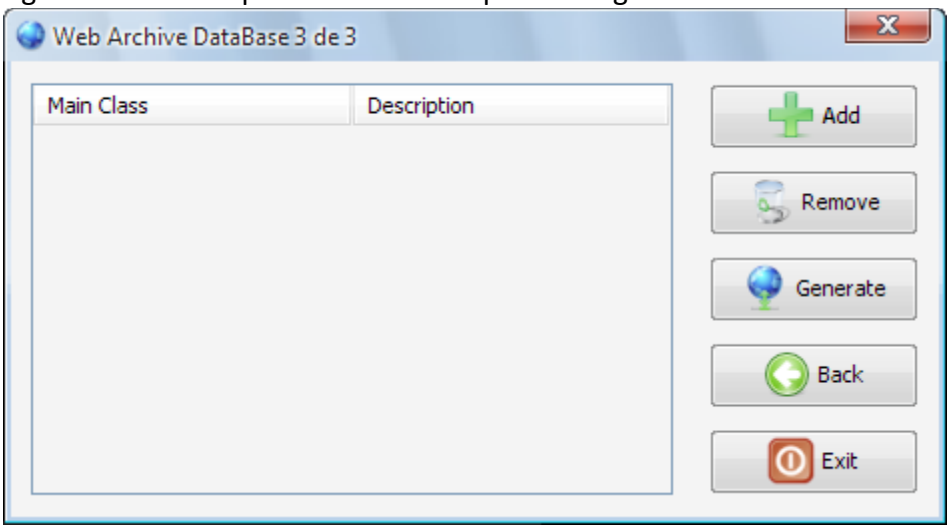
Si se desea agregar un plugin, debe hacerse clic en el botón **“Add”**, este despliega el diálogo de abrir archivos, mostrando solo archivos .JAR, una vez ahí se debe indicar la ruta al plugin y este aparecerá en la lista, si el JAR seleccionado no es un plugin de geocodificación Atlas, se notificará 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 171. Tercer paso del asistente para configuración de archivos WAR.

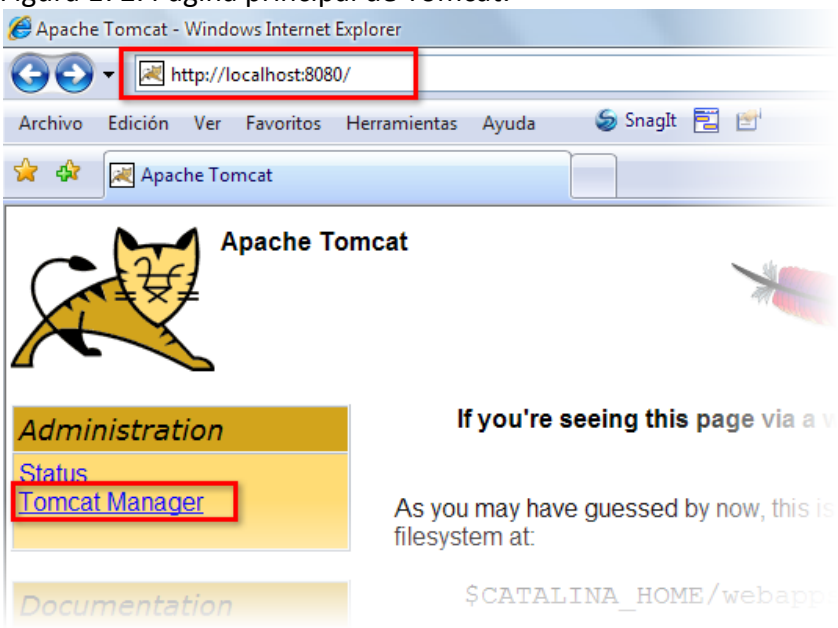


Ahora el archivo generado puede ser instalado en el contenedor de servlets. Para este caso se cubre el proceso de instalación bajo Apache Tomcat.

Tomcat debe estar configurado y corriendo la como se indicó en la guía de instalación de prerrequisitos.

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 172. Una vez ahí debe hacerse clic sobre la opción “Tomcat Manager”.

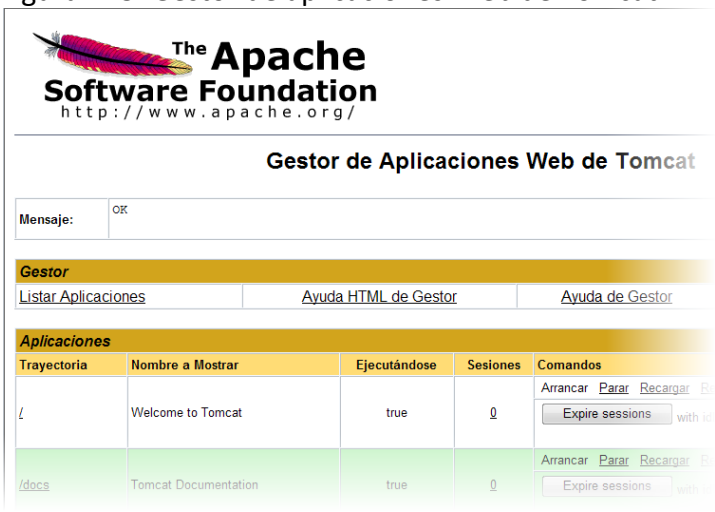
Figura 172. Página principal de Tomcat.



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 173.

Figura 173. 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.

Figura 174. 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 175.

Figura 175. Aplicación correctamente instalada en el servidor Tomcat.

A4. GUÍA DE 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.

Figura 176. Iniciar el panel de control Atlas.

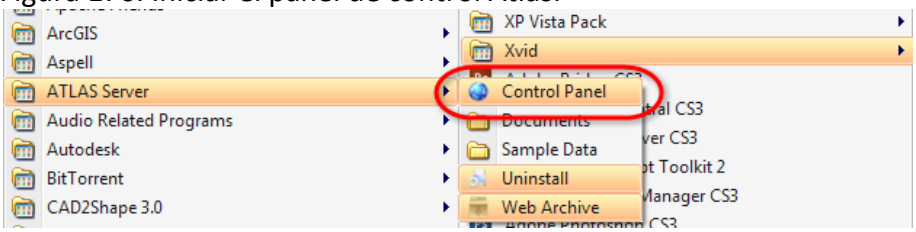
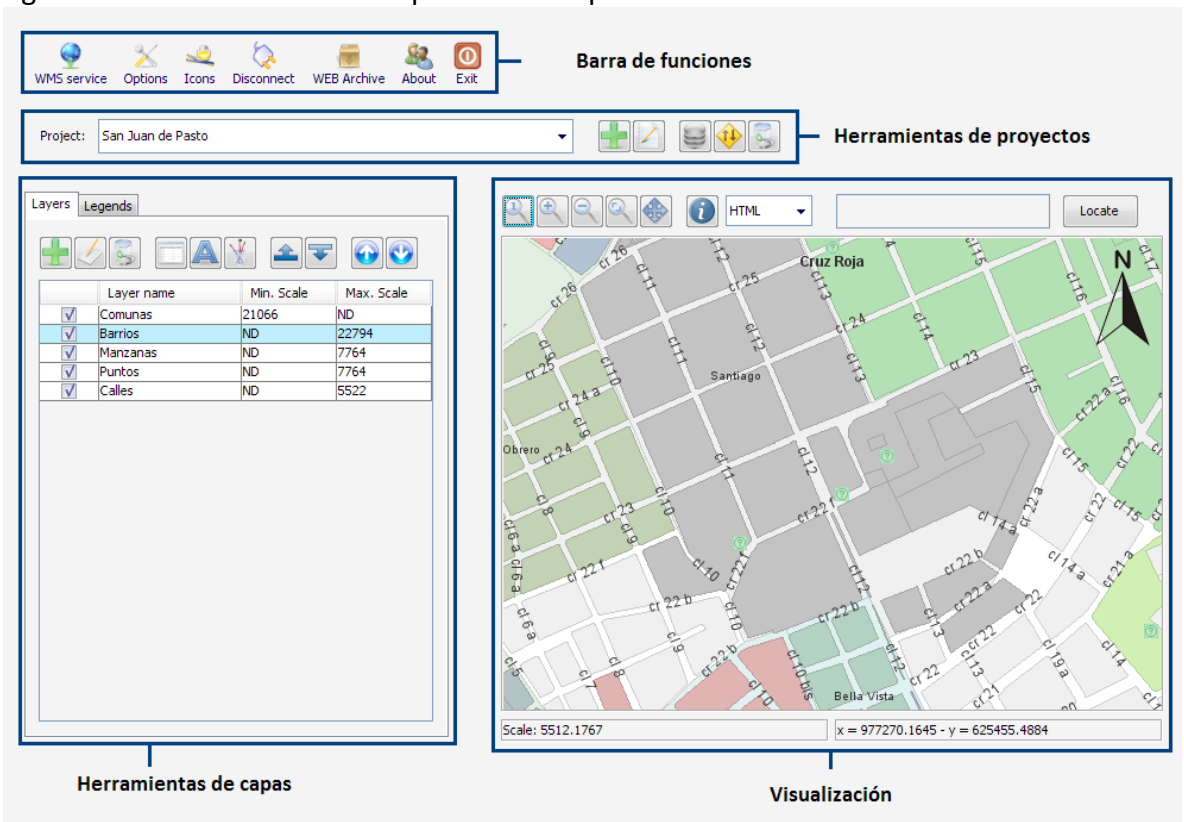


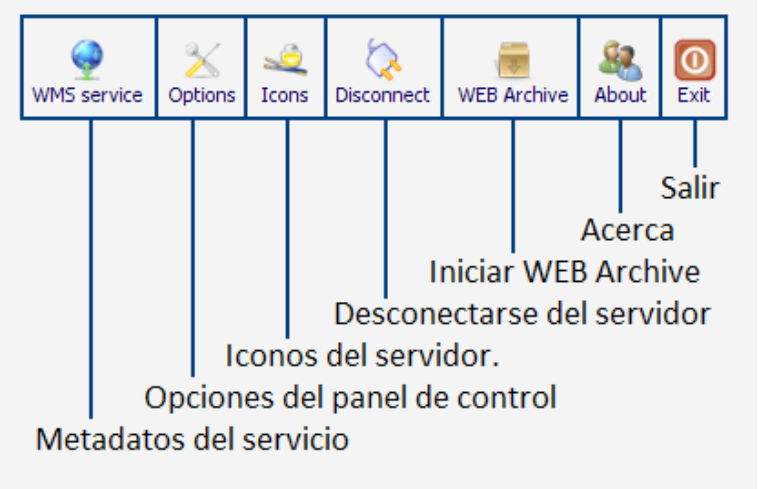
Figura 177. Distribución de componentes del panel de control.



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 177.

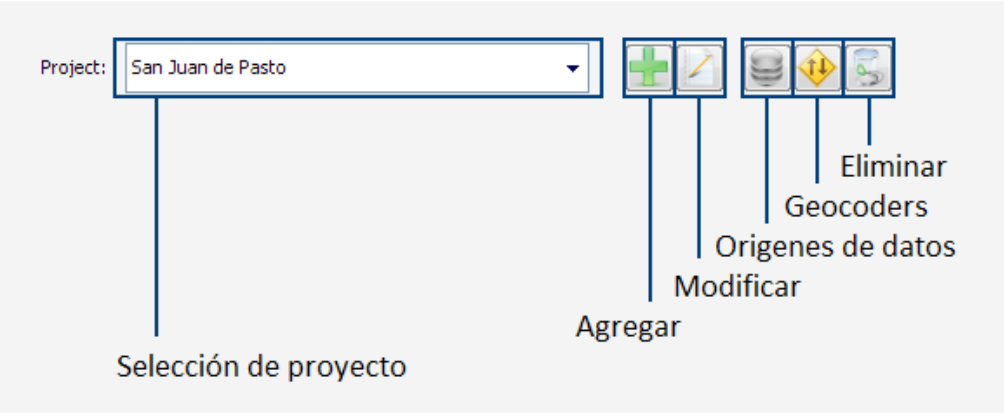
La barra de funciones, contiene botones para ejecutar procedimientos que afectan a todo el servidor y al panel de control, estas funciones se detallan más adelante y la apariencia de la barra de muestra en la figura.

Figura 178. Barra de funciones del panel de control.



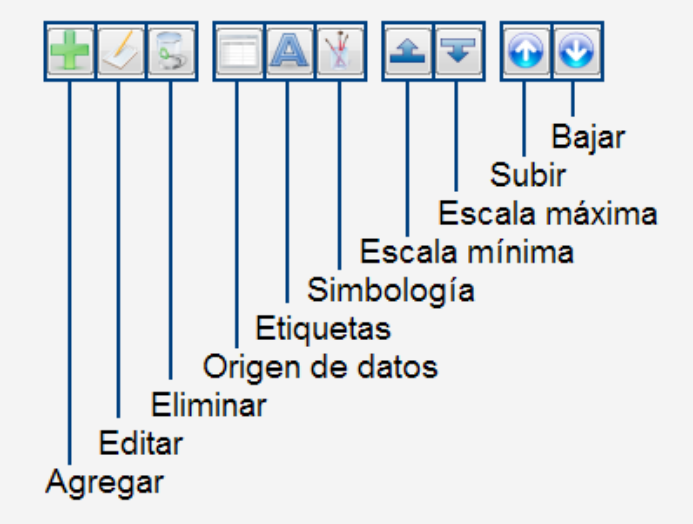
La barra de proyectos, contiene funciones para seleccionar el proyecto actual y aplicar diferentes operaciones sobre él, como se muestra en la figura 179.

Figura 179. Barra de proyectos.



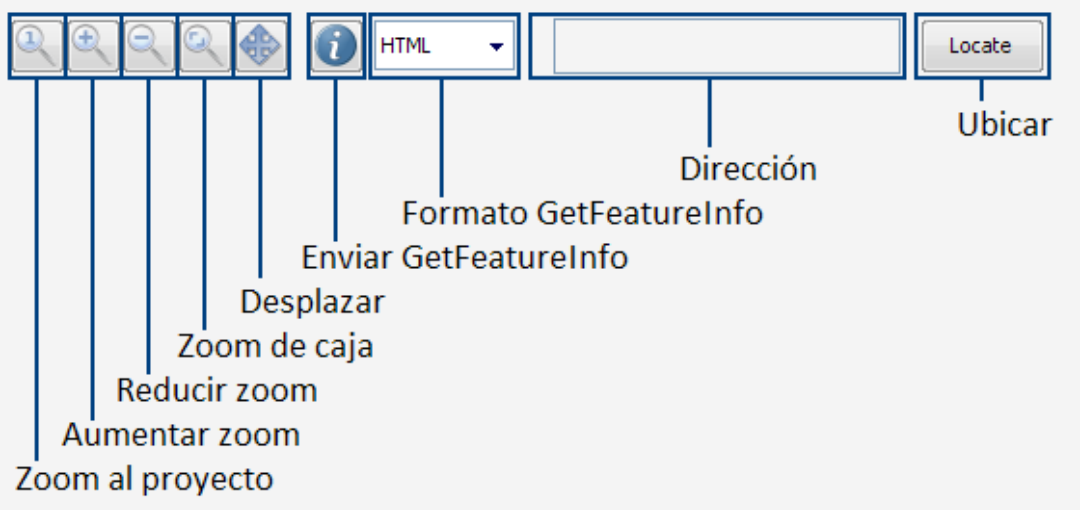
La sección de **herramientas de capas** lista las capas y leyendas del proyecto actual y permite realizar diferentes operaciones sobre ellas, como se muestra en la figura.

Figura 180. 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 181.

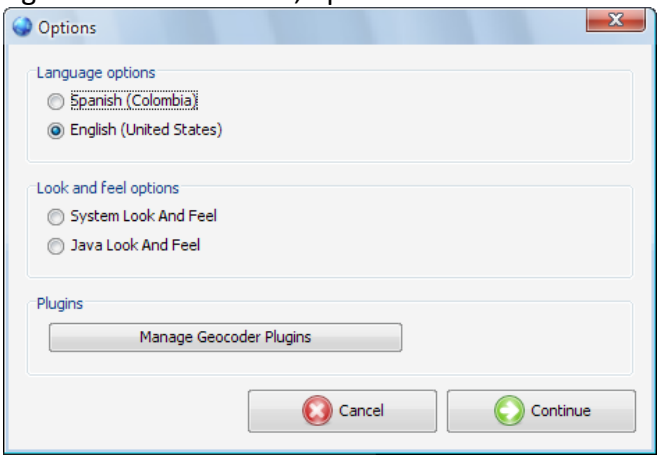
Figura 181. Sección de visualización.



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 182.

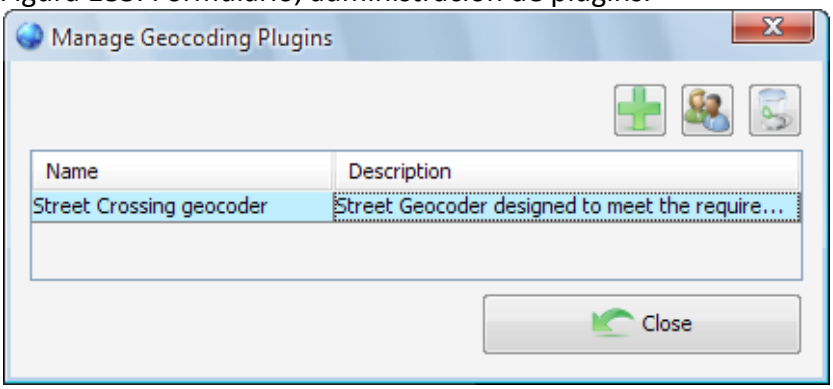
Figura 182. Formulario, opciones de la herramienta.



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 183. 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.

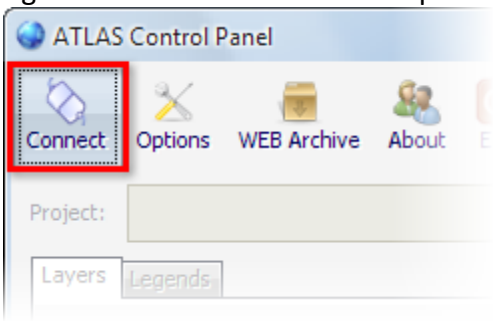
Figura 183. Formulario, administración de plugins.



Establecer conexión con un servidor.

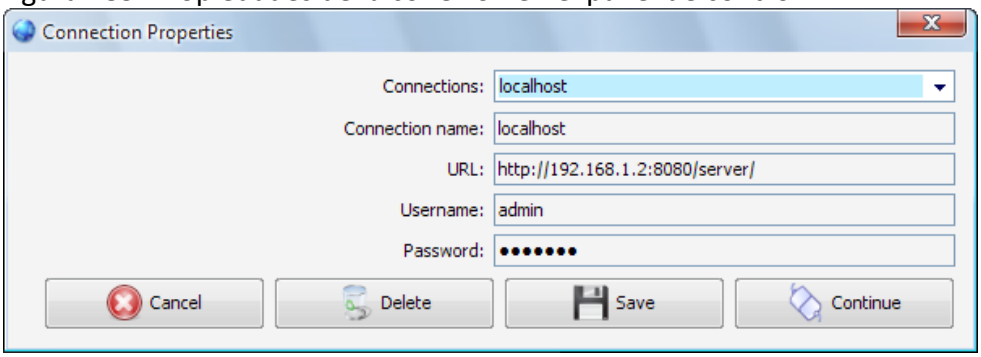
El primer paso para utilizar el panel de control es conectarse a un servidor ya establecido, para esto, debe hacerse clic en el botón **"Connect"**, ubicado en la esquina superior izquierda de la barra de funciones, que se muestra en la figura.

Figura 184. 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.

Figura 185. 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.

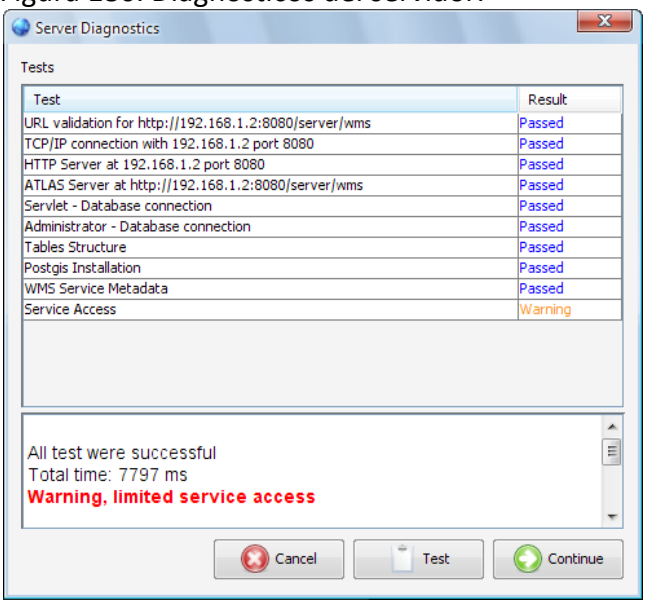
Todos los campos en este formulario son obligatorios, cuando están completos, debe hacerse clic en el botón **“Continue”**.

Esto conduce al formulario que valida los datos suministrados (ver figura 186), 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.

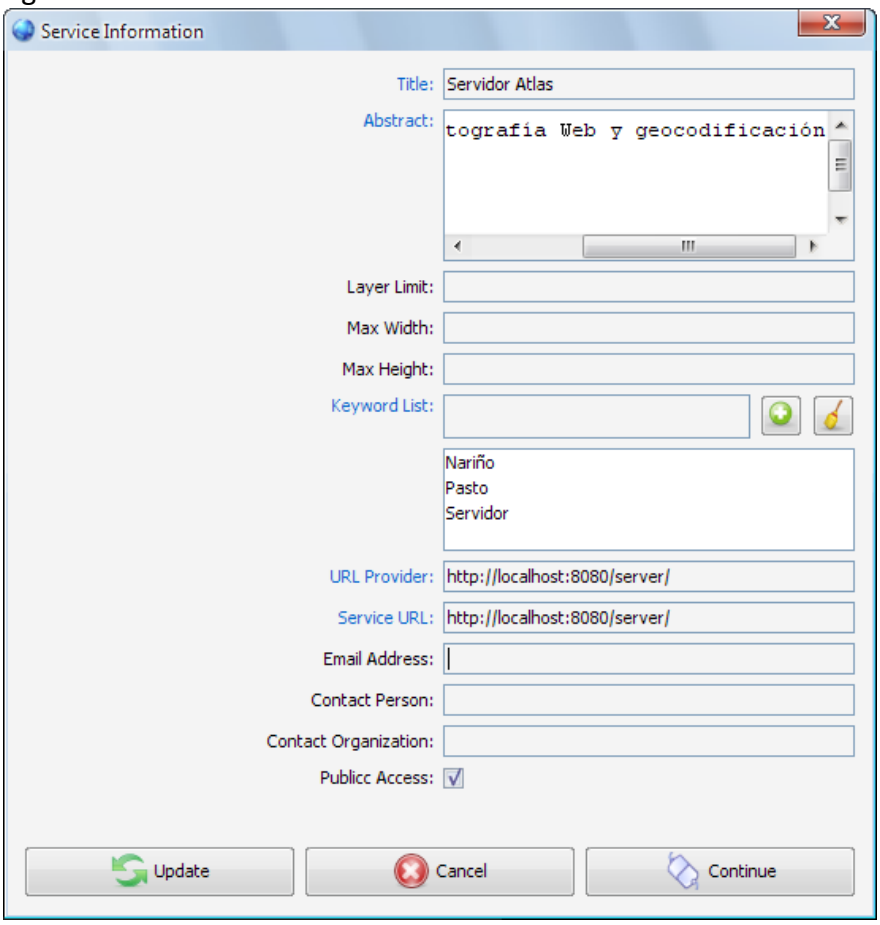
El botón **“test”** permite repetir las pruebas, y el botón **“Cancel”,** regresa al formulario de datos de la conexión, si todas las pruebas concluyen de forma satisfactoria, el boton **“continue”**, estará habilitado, el permite cerrar el diálogo para continuar a la herramienta.

Figura 186. 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.

Figura 187. 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.

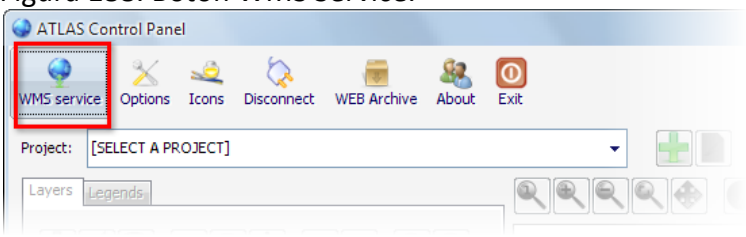
El campo **“Contact Person”** corresponde al nombre del administrador del servicio.

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 188.

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

Figura 188. Botón WMS Service.

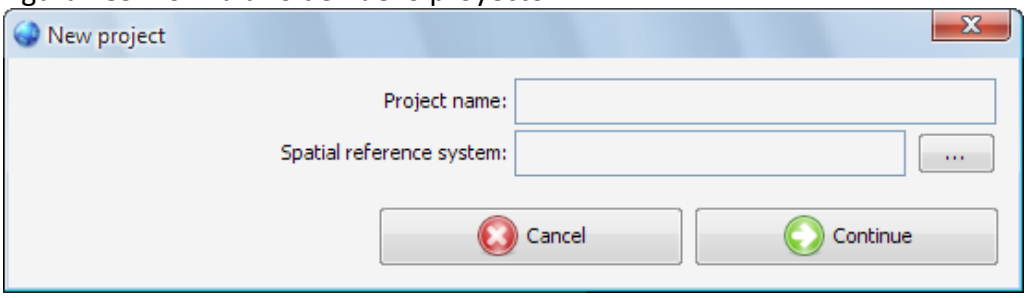


Administrar proyectos.

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.

Figura 189. Formulario de nuevo proyecto.

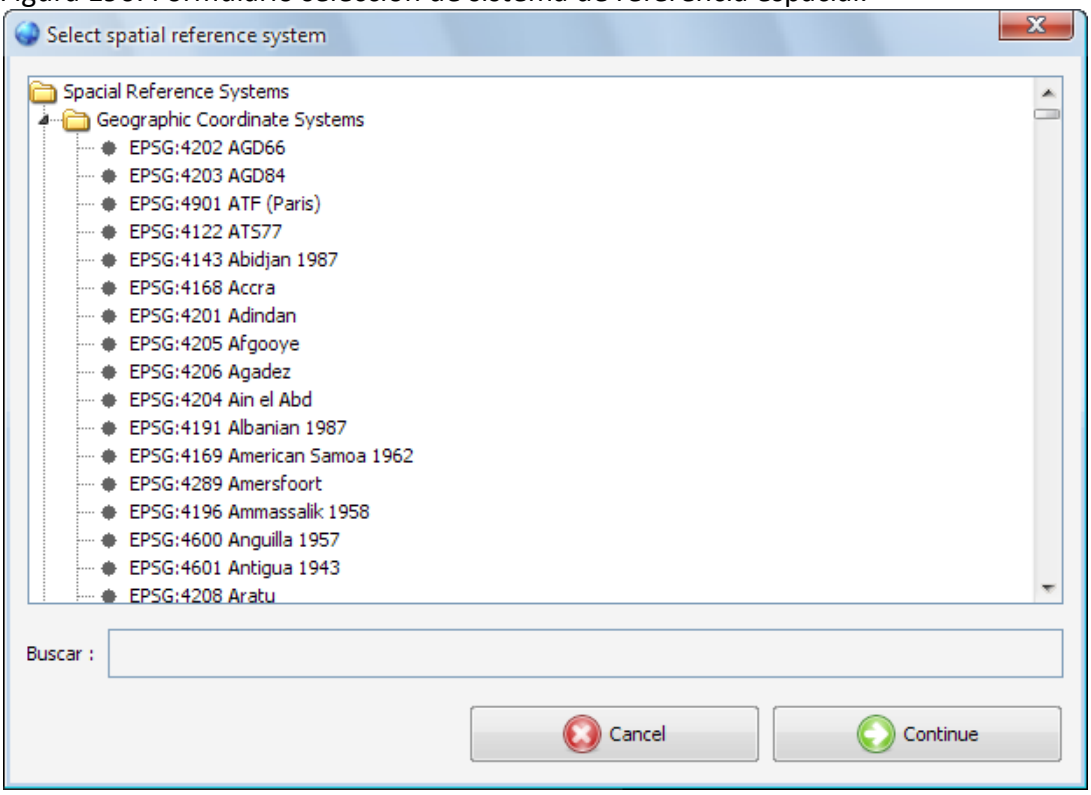


En este formulario, **"Project Name"** corresponde al nombre del proyecto y **"Spatial Reference System"** al sistema espacial de referencia, el botón de **"Cancel"**, abandona el proceso sin realizan cambios, y el botón **"Continue"**, verifica la información proporcionada y guarda el proyecto. Los 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 **"selección 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 190.

Figura 190. Formulario selección de sistema de referencia espacial.



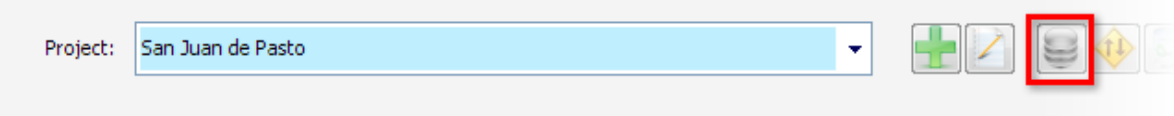
En el formulario de selección de sistema de referencia espacial, el cuadro **"Buscar"**, permite filtrar los sistemas de referencia por su nombre a medida que se escribe.

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.

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 191.

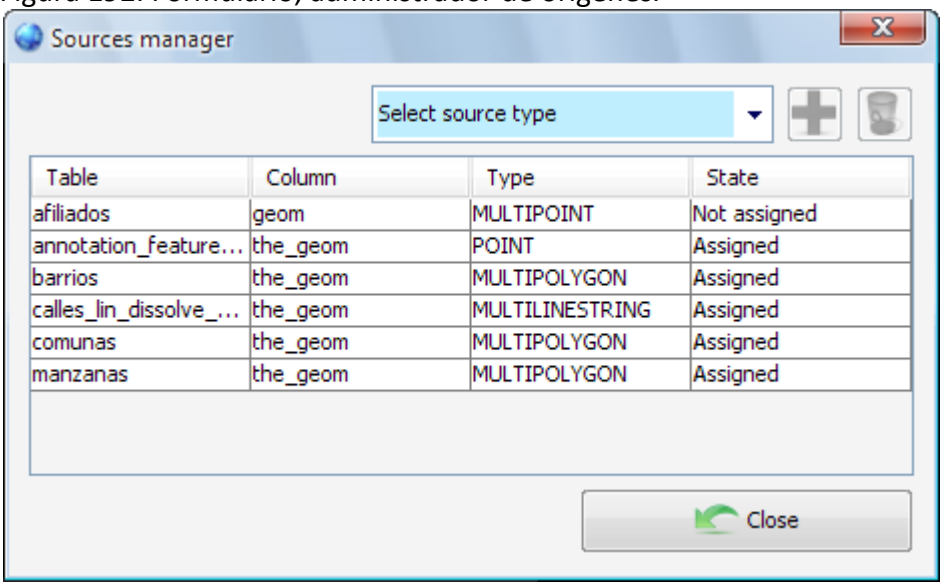
Figura 191. Botón administrar orígenes de datos.



Al presionar este botón, debe aparecer el formulario, administrador de orígenes, 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.

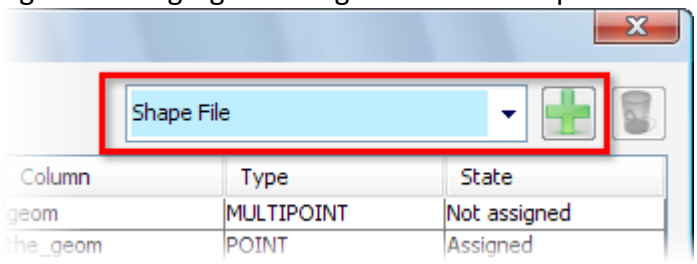
Figura 192. Formulario, administrador de orígenes.



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.

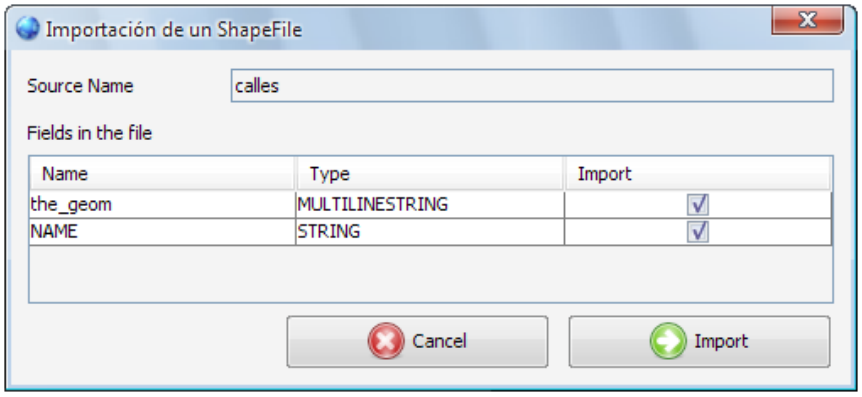
Para agregar un origen de datos desde un ShapeFile de ESRI, la selección debe estár como indica la figura.

Figura 193. 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.

Figura 194. Formulario, importación de un ShapeFile.

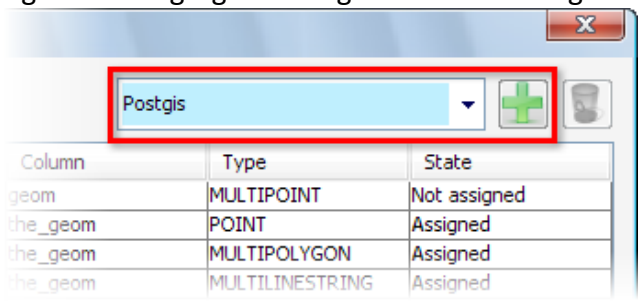


En este formulario, **"Source Name"**, indica el nombre que tendrá el origen en el proyecto, la tabla **"Fields in the file"**, muestra los campos hallados en el archivo, se puede seleccionar cuáles de ellos se importarán modificando las casillas de selección de la columna **"import"**.

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.

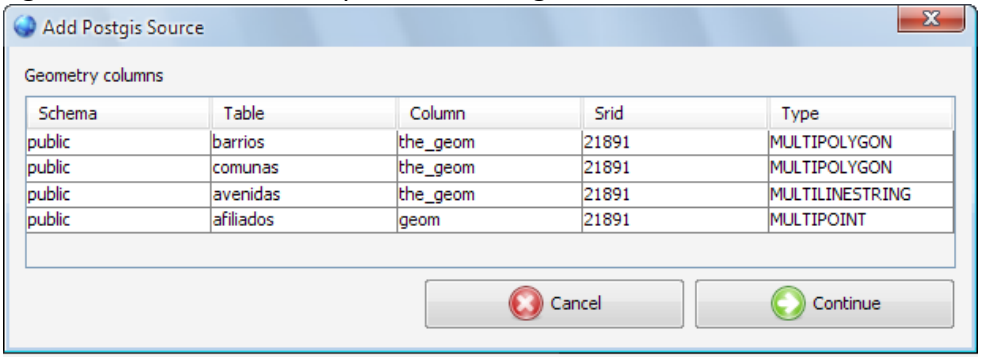
Para agregar un origen de datos desde Postgis, la selección debe estár como indica la figura 195, además, postgis debe estár instalado en la base de datos del sistema.

Figura 195. Agregar un origen de datos Postgis.



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

Figura 196. 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.

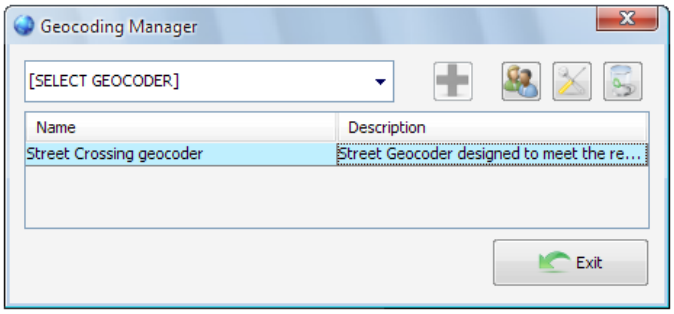
Administrar geocoders del proyecto.

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.

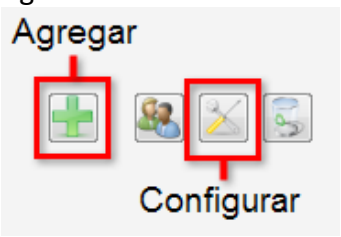
Luego de hacer clic sobre el botón, debe aparecer un formulario como el que se muestra en la figura.

Figura 197. 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. Las opciones a configurar de cada geocoder dependerán del proveedor del plugin.

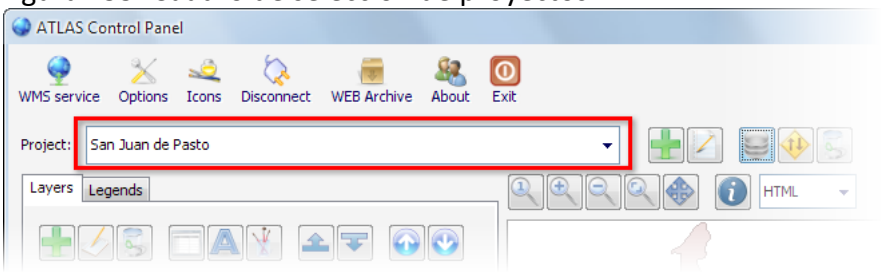
Figura 198. Botones en la administración de geocoders.



Administrar capas.

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

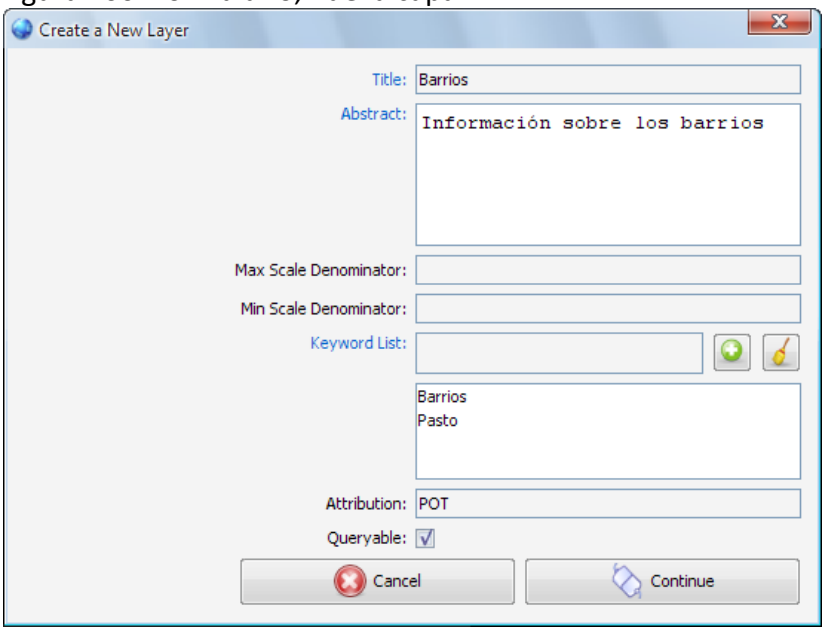
Figura 199. Cuadro de selección de proyectos.



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 200) 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 200. 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.

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 201 y presionar el botón que se muestra en la figura 202.

Figura 201. Lista de capas.

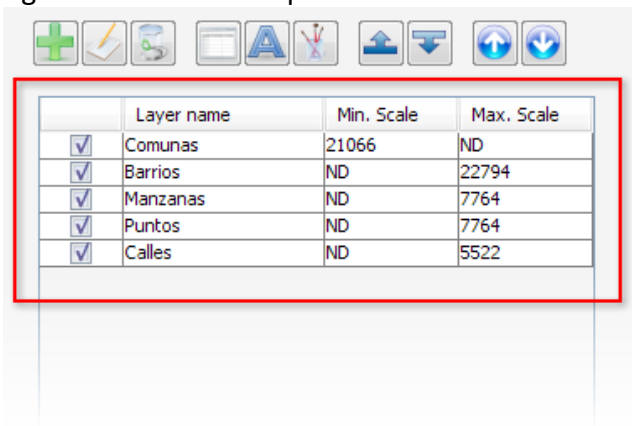
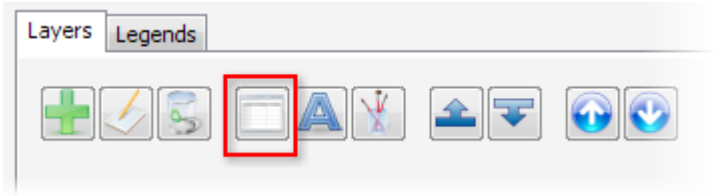
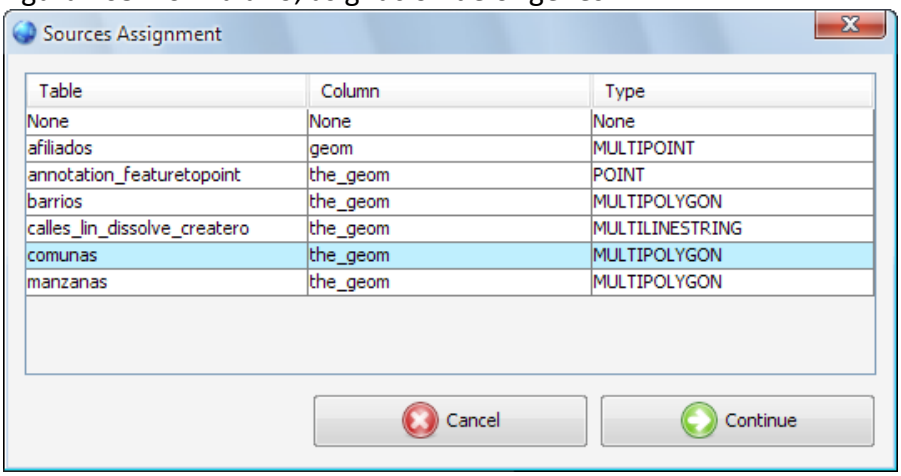


Figura 202. Botón, configurar orígenes de datos.



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

Figura 203. Formulario, asignación de orígenes.



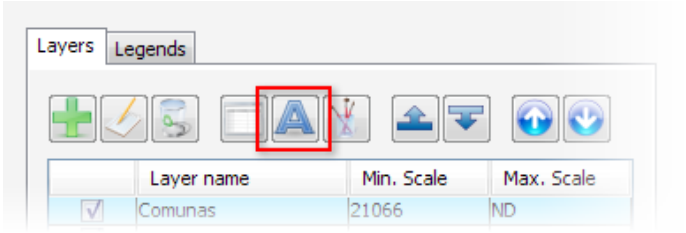
El formulario, asignación de orígenes, muestra un listado de los orígenes de datos que contiene el proyecto, para asignar uno de estos orígenes a la capa, basta con seleccionarlo de la lista y hacer clic en **“Continue”**, presionar **“Cancel”**, ocaciona que se cierre el formulario sin hacer cambios.

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.

Administración de etiquetado.

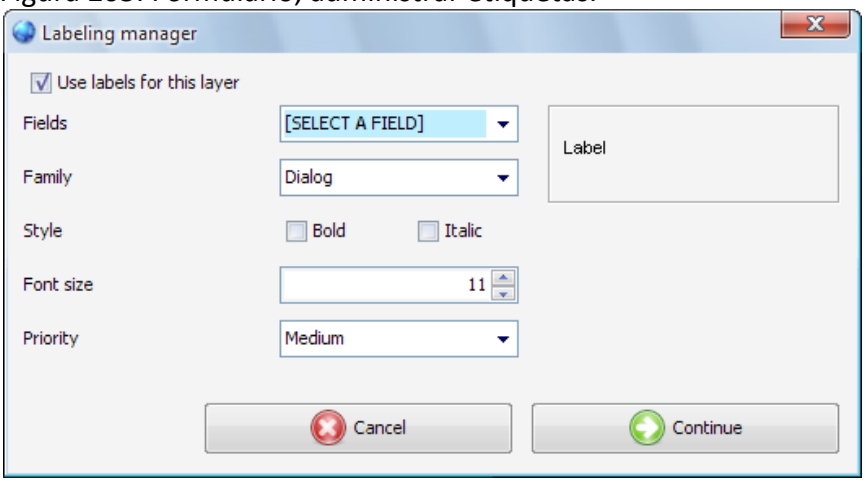
El etiquetado, permite que junto con la geometría de una capa se dibujen letreros que describen lo que se está presentando. Para modificar las opciones de etiquetado de una capa debe hacerse clic sobre el botón que se muestra en la figura.

Figura 204. Botón, administrar etiquetado.



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

Figura 205. Formulario, administrar etiquetas.



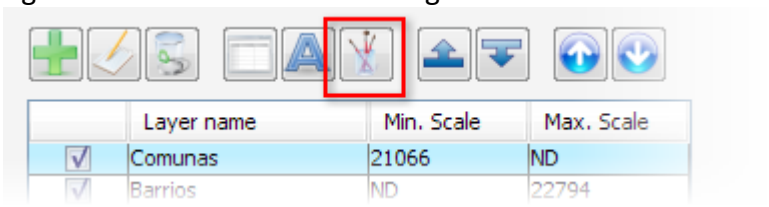
En este formulario, la opción **"use labels for this layer"**, indica si esta capa usará o no etiquetas, el cuadro de selección **"Fields"**, muestra los campos que contiene el origen de datos de la capa, debe seleccionarse el campo cuyos valores se mostrarán, el cuadro de selección **"Family"** presenta 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.

Simbología

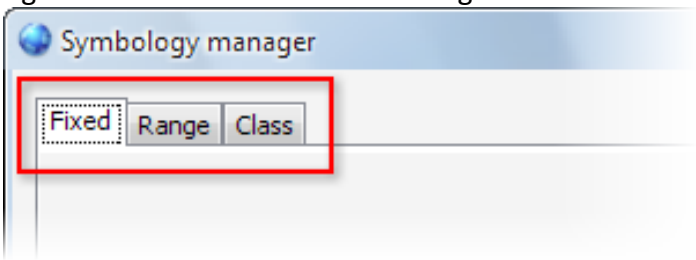
La simbología determina la forma en que presentarán las geometrías, para editar la simbología de una capa, hay que seleccionar una de la lista y hacer clic en el botón que muestra la figura.

Figura 206. 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.

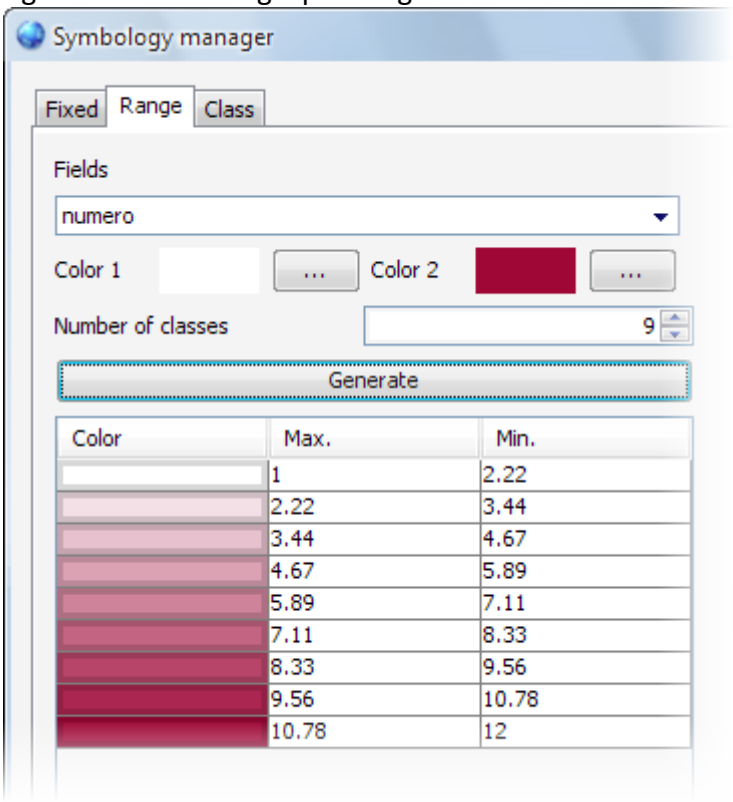
Figura 207. Alternativas de simbología.



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 208), 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.

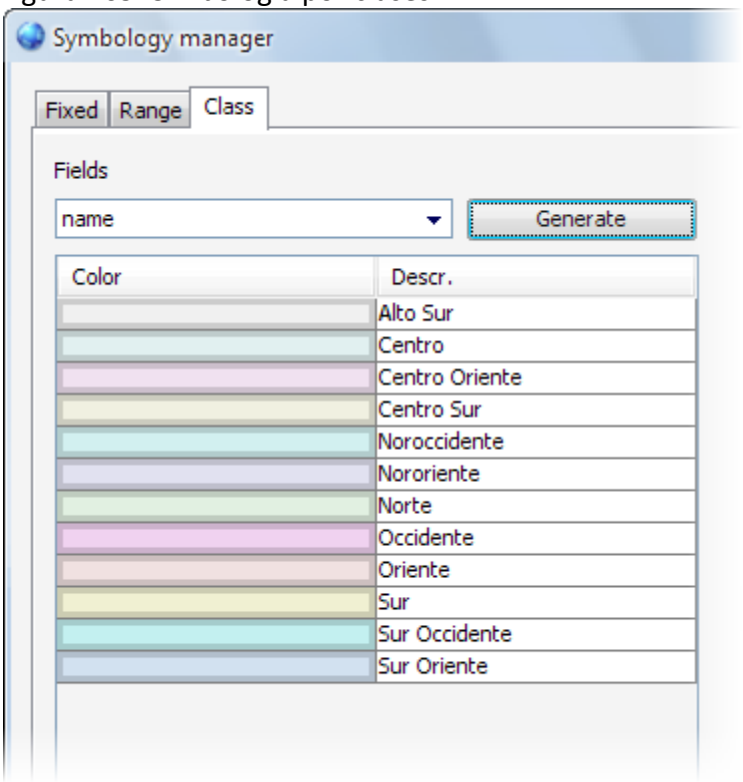
Figura 208. Simbología por rangos.



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.

Figura 209. Simbología por clases.

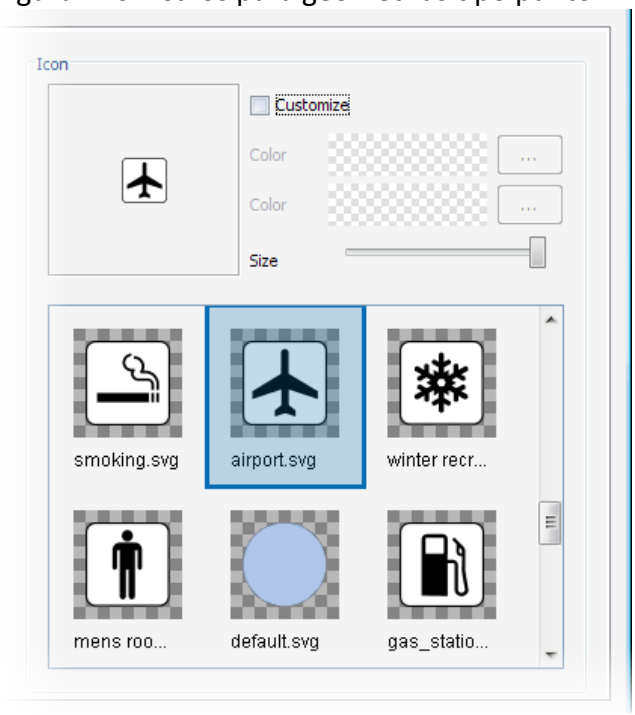


Estilos

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

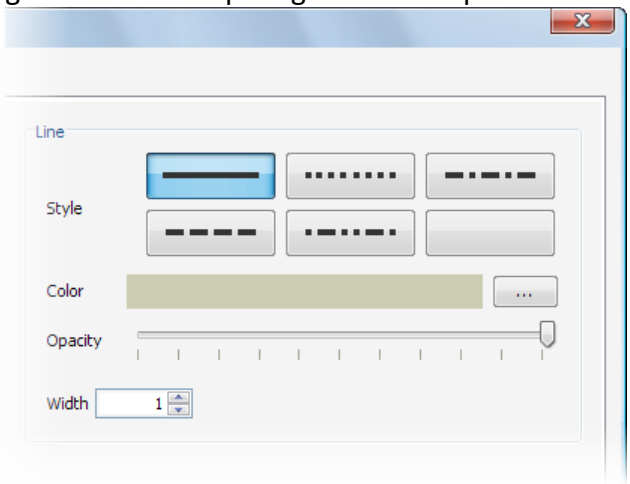
Para una geometría tipo punto (ver figura 210), es posible seleccionar un icono que se colocará en la ubicación del punto, el campo **"Customize"**, indica si los colores del icono deben o no ser reemplazados, en caso de haber elegido personalizar los colores, se puede elegir uno de fondo y uno de relleno, la barra **"Size"**, permite modificar el tamaño del icono.

Figura 210. Estilos para geometrías tipo punto.



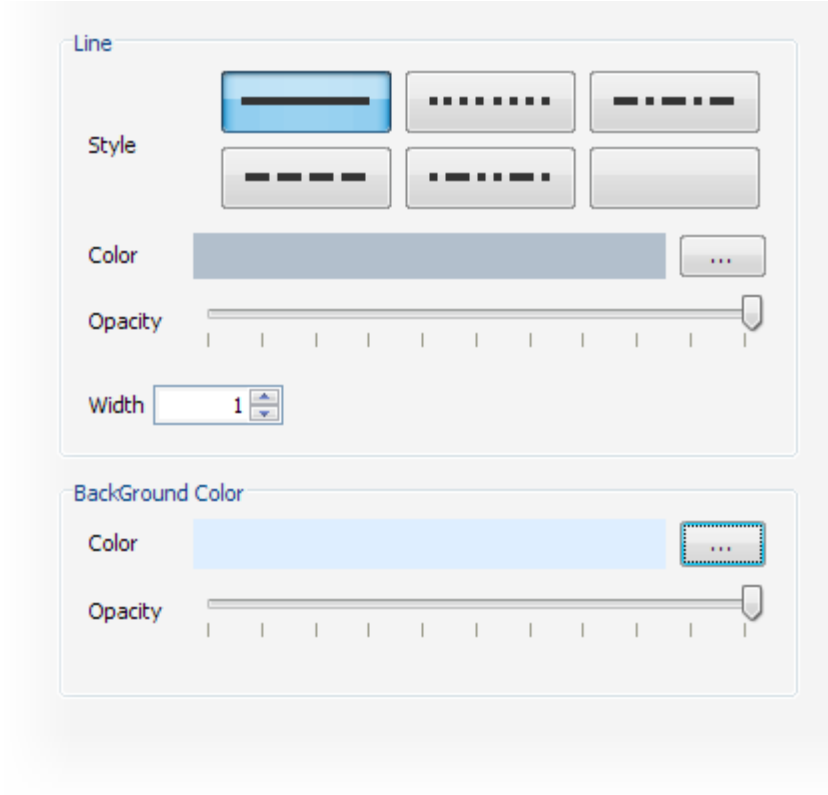
Para una geometría tipo línea, es posible cambiar su estilo, su color y su grosor, como se muestra en la figura.

Figura 211. 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.

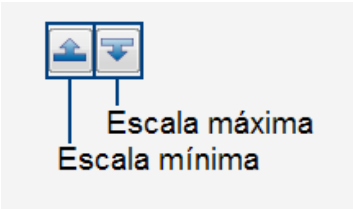
Figura 212. Estilos para geometrías tipo polígono.



Cambiar los denominadores de escala.

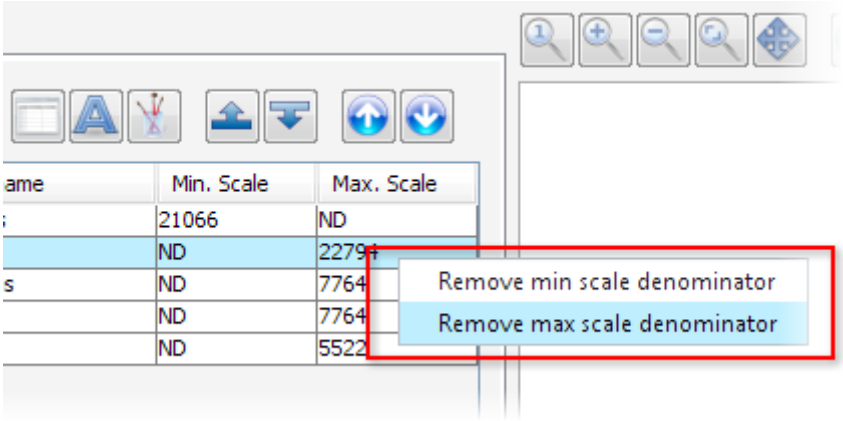
Para modificar los denominadores de escala de una capa, se requiere usar la sección de visualización para determinar la escala deseada, y presionar el botón de escala máxima o de escala mínima.

Figura 213. 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.

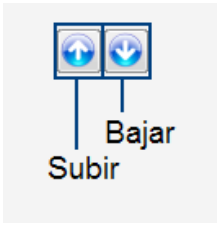
Figura 214. Remover denominadores de escala.



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.

Figura 215. Botones de subir y bajar capa.



A5. GUÍA DE 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 guía de instalación de servidor, y se usan en el panel de control, como se describe en la guía de administración, a continuación se aborda el tema de la creación de estos plugins.

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

Tabla 73. 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

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.

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.

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.

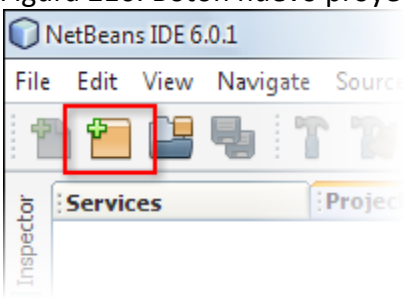
A6. GUÍA DE DESARROLLO DE APLICACIONES JAVA SE.

En esta guía se cubre la realización de una sencilla aplicación de escritorio que demuestra los conceptos básicos sobre la creación de aplicaciones para este entorno, dicho desarrollo puede hacerse inclusive con un editor de texto y el compilador de java, sin embargo, para este caso se usará el IDE NetBeans 6.0.

Configuración de entorno.

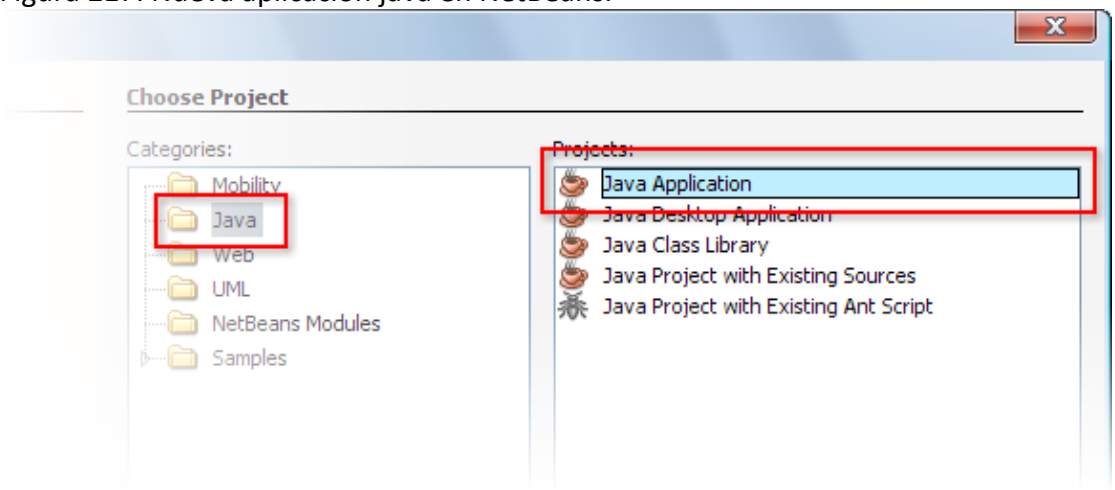
Primero, debe crearse un nuevo proyecto en NetBeans, para ello debe hacerse clic sobre el botón mostrado en la figura.

Figura 216. Botón nuevo proyecto en NetBeans.



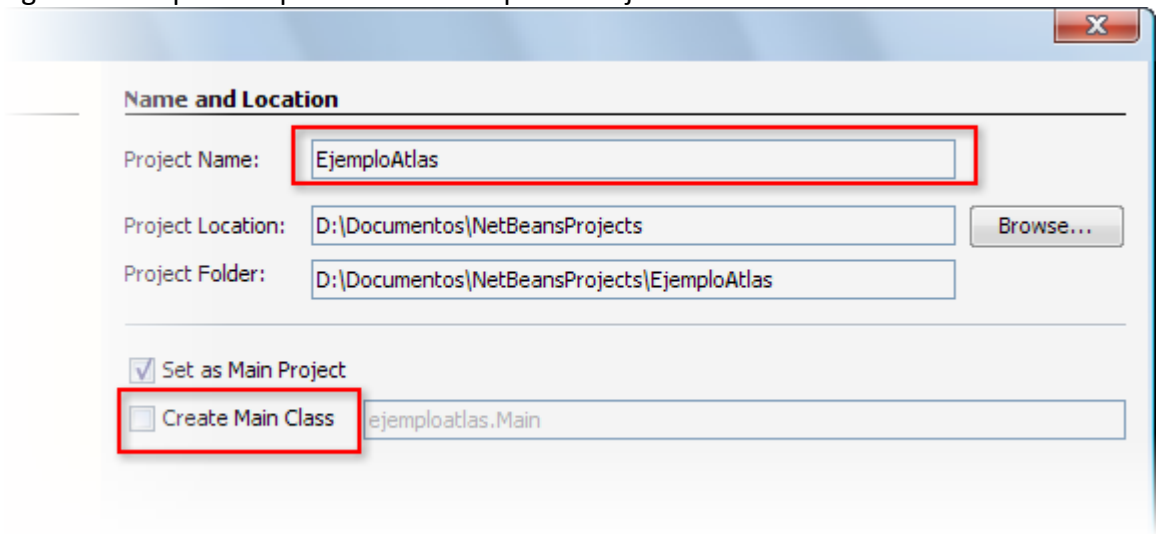
Luego, en la ventana que aparece deben seleccionase las opciones que muestra la figura siguiente y hacer clic en “next”.

Figura 217. Nueva aplicación java en NetBeans.



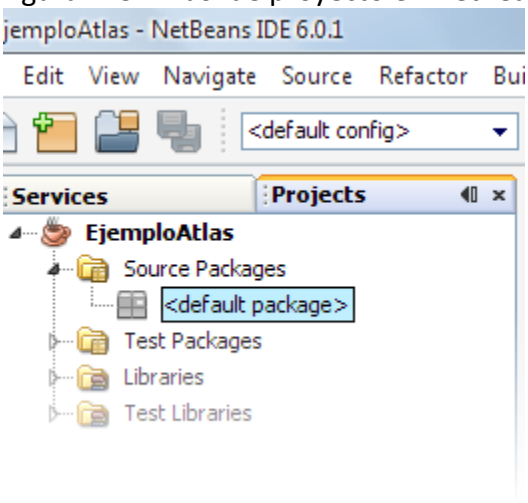
En el diálogo siguiente se indica el nombre de la nueva aplicación y se indica a NetBeans que no debe crear una clase principal, por último debe hacerse click en finish.

Figura 218. Opciones para una nueva aplicación java en NetBeans.



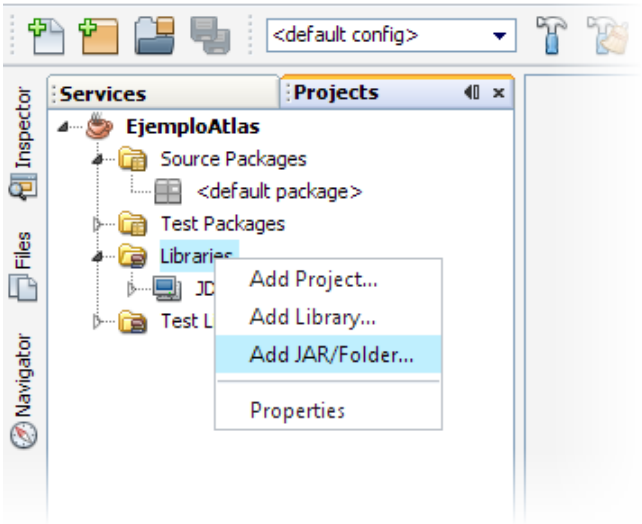
La aplicación creada aparece ahora en el árbol de proyectos como se muestra en la figura 219.

Figura 219. Árbol de proyecto en NetBeans.



Luego se deben agregar las librerías requeridas. Para ello debe desplegarse el menú que se muestra en la siguiente figura.

Figura 220. Menu agregar archivo jar en NetBeans.

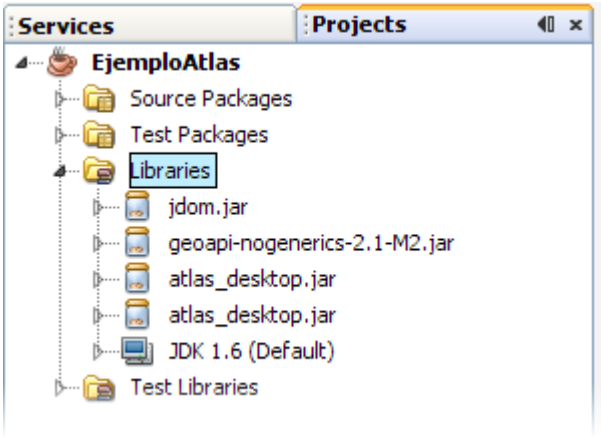


Luego, debe agregarse los siguientes archivos.

- atlas_desktop.jar
- atlas_common.jar
- jdom.jar
- geoapi-nogenerics-2.1-M2.jar

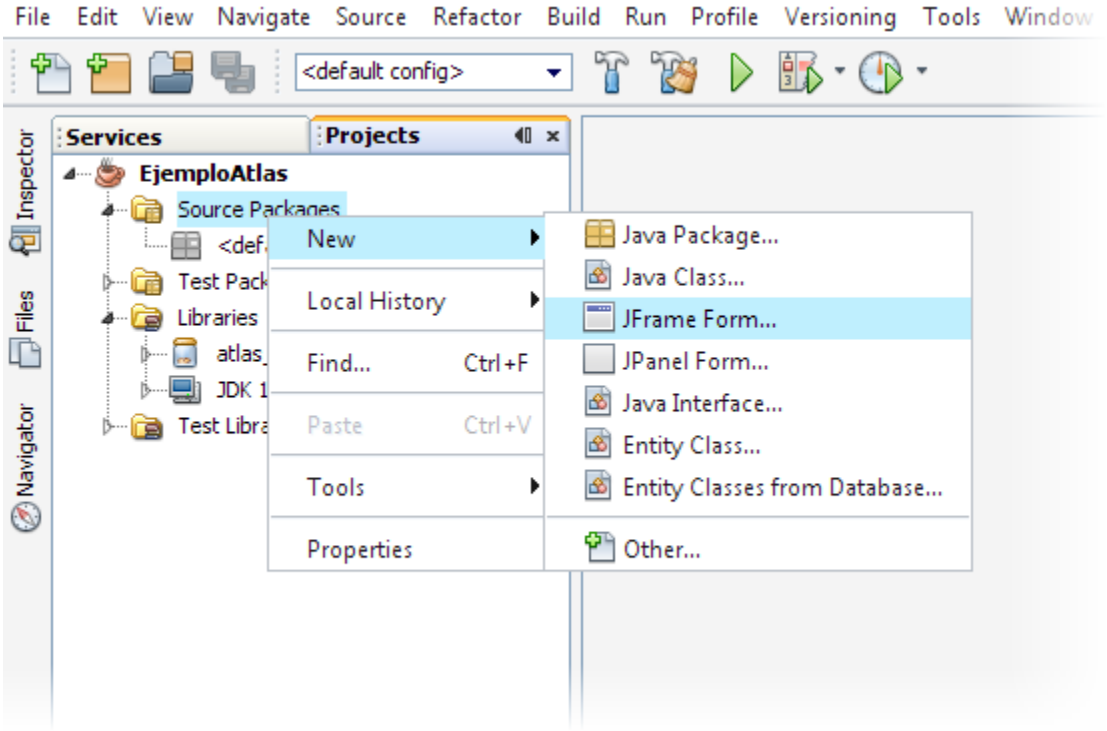
El árbol de proyecto debe verse como se muestra en la figura 221.

Figura 221. Libraries en un proyecto NetBeans.



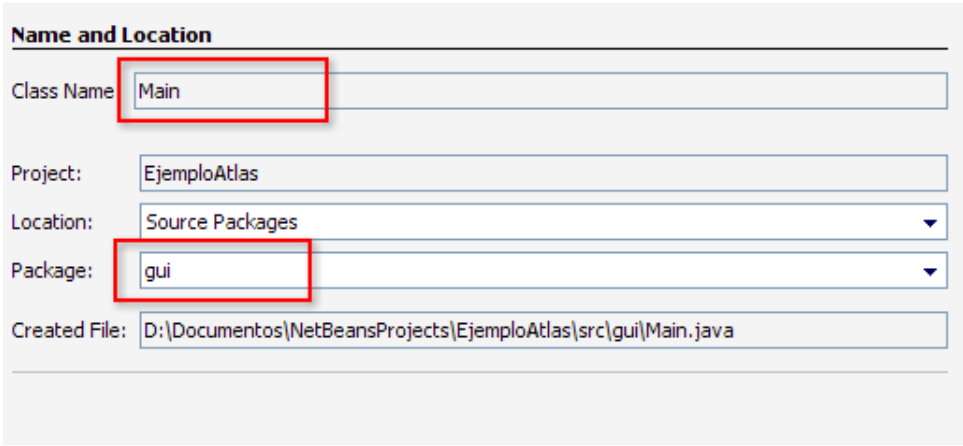
Ahora, se debe crear un formulario en la aplicación, usando el menú que se muestra en la figura 222.

Figura 222. Creación de un formulario en NetBenans.



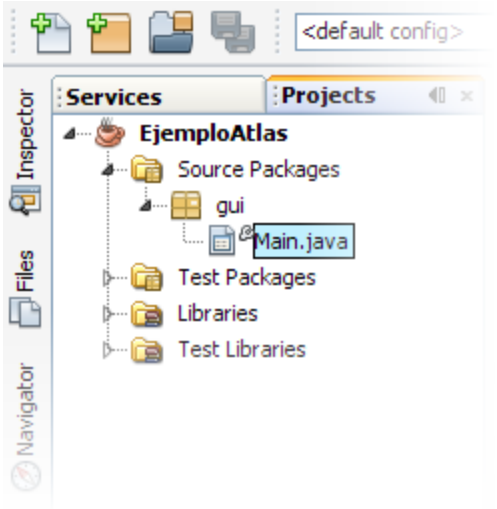
En la ventana que aparece, se debe establecer el nombre de la clase como “Main”, y el paquete “gui”, como se muestra en la figura 223 y luego se debe hacer clic en finish.

Figura 223. Nombre y ubicación de un proyecto NetBeans.



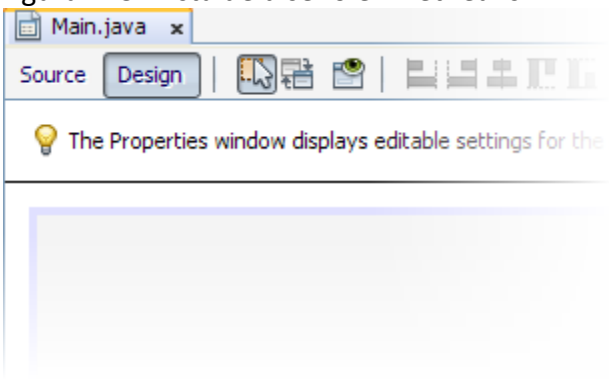
El árbol del proyecto debe verse ahora como indica la figura 224.

Figura 224. Árbol de proyecto en NetBeans con una clase nueva.



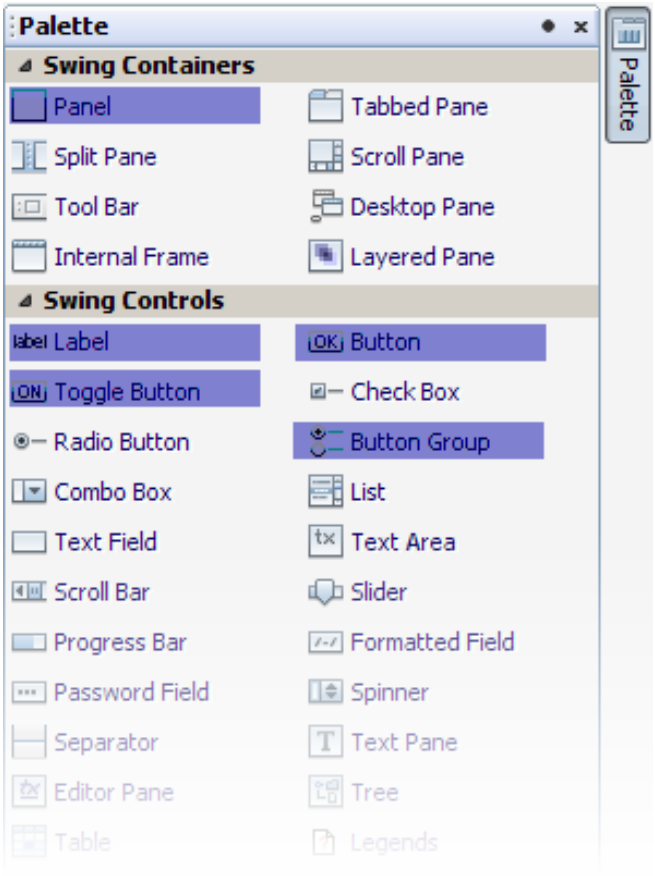
Al hacer doble clic sobre el nodo Main.java, este se mostrará en el recuadro central, debe hacerse clic sobre el botón “design” (ver figura 225) en caso de que este no esté marcado.

Figura 225. Vista de diseño en NetBeans.



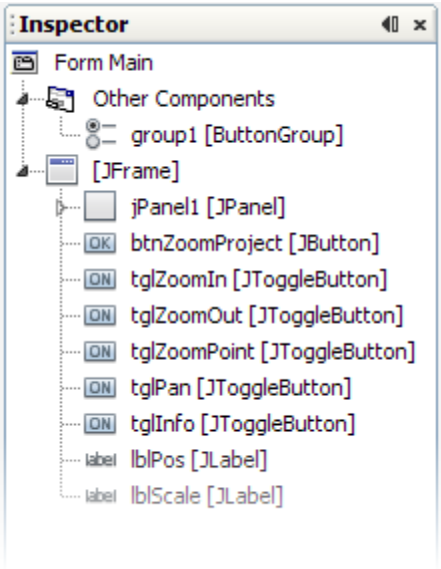
Ahora es posible arrastrar componentes desde la paleta hasta el formulario. La paleta se puede desplegar presionando Ctrl + Mayusculas + 8 en el teclado. De esta paleta se usarán los componentes que se muestran en la figura 226.

Figura 226. Paleta de componentes Swing en java.



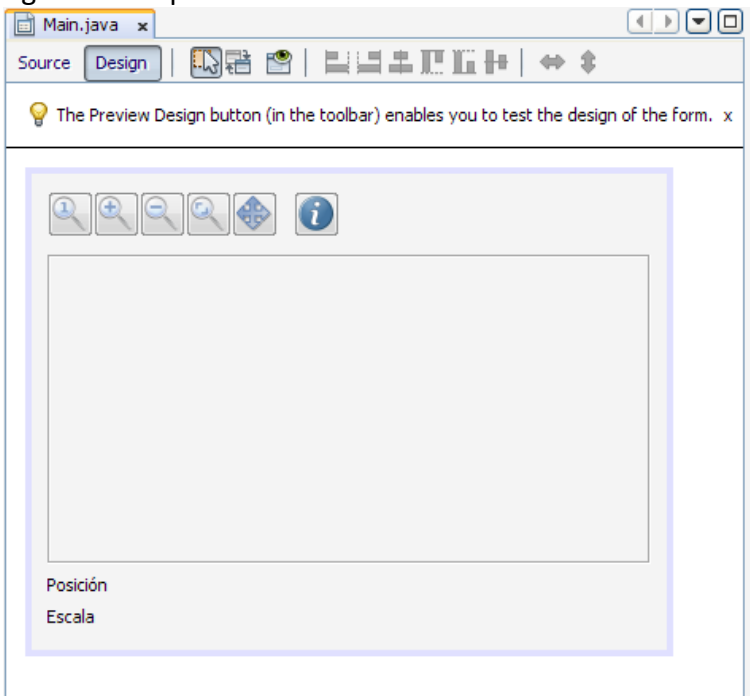
De la paleta se deben agregar diferentes componentes de los tipos resaltados, de modo que el árbol del formulario se vea como se muestra en la figura 227.

Figura 227. Arbol de componentes del formulario principal.



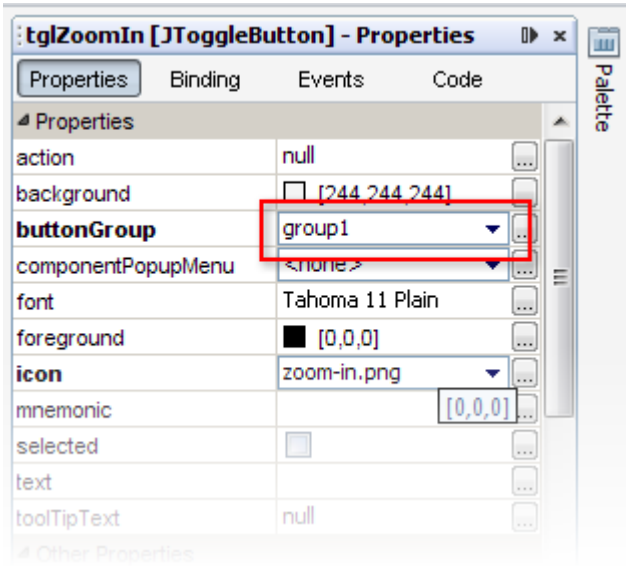
Estos componentes ubicados sobre el formulario deben verse como se muestra en la figura.

Figura 228. Apariencia de la clase Main.



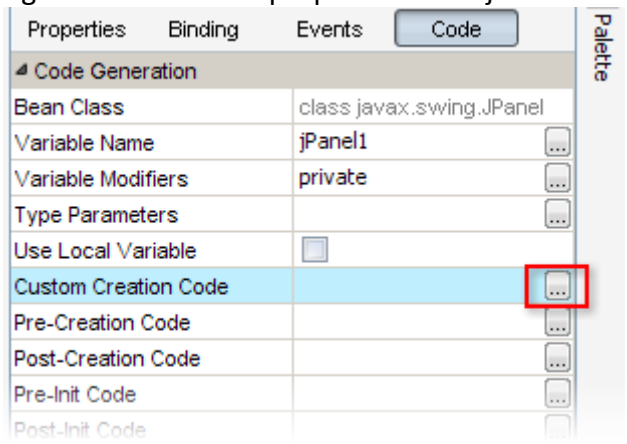
Ahora deben conectarse todos los botones toogle al group 1, para ello se debe hacer clic en cada uno de ellos, consultar su tabla de propiedades y dejarla como se muestra en la figura 229.

Figura 229. Grupos de botones.



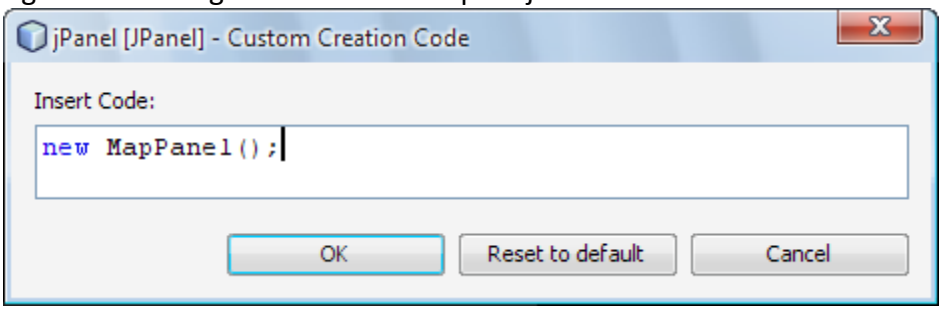
Ahora se debe hacer clic sobre el JPanel y en su tabla de propiedades hacer clic en “code” y luego en el botón al lado de “Custom Creation Code”, como se muestra en la figura 230.

Figura 230. Tabla de propiedades del jPanel.



En el formulario que aparece se debe escribir el contenido que muestra la figura.

Figura 231. Código de inicialización para jPanel.



Luego en la vista de código del formulario principal, bajo la línea de package debe agregarse el texto que indica la figura 232.

Figura 232. Import de la clase MapPanel.

```
7   package gui;
8
9   import org.atlas.desk.gui.MapPanel;
10
```

Bajo la declaración de la clase se debe colocar el código que indica la figura 233.

Figura 233. Código de acceso a una instancia de MapPanel.

```
5   public class Main extends javax.swing.JFrame {
6
7       private MapPanel mapPanel;
8
9
10      public MapPanel getMapPanel() {
11          return mapPanel;
12      }
13
14      public void setMapPanel(MapPanel mapPanel) {
15          this.mapPanel = mapPanel;
16      }
```

El constructor de la clase debe modificarse de modo que se vea como muestra la siguiente figura.

Figura 234. Constructor de la clase Main.

```
public Main() {
    initComponents();
    this.mapPanel = (MapPanel)this.jPanel;
}
```

Al realizar correctamente el anterior ejercicio, se ha establecido un proyecto base para continuar con los demás puntos.

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

La sección de imports del la clase Main, debe quedar como se muestra en la figura 235.

Figura 235. Imports en la clase Main para establecer conexión con un servidor.

```
1  package gui;
2
3  import clientdata.Layer;
4  import clientdata.Project;
5  import java.net.URL;
6  import java.util.logging.Level;
7  import java.util.logging.Logger;
8  import org.atlas.desk.gui.MapPanel;
9
```

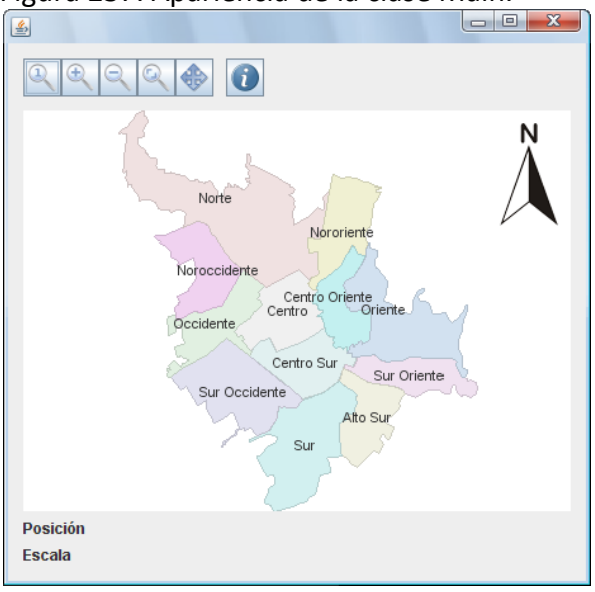
También el constructor de la clase debe modificarse y quedar como se muestra en la figura 236.

Figura 236. Constructor de la clase Main, preparado para conectarse con un servidor.

```
22  public Main() {
23
24      initComponents();
25      setMapPanel((MapPanel) jPanel);
26
27      try {
28          getMapPanel().setURL(new URL("http://192.168.10.111:8084/service"));
29          Project project = getMapPanel().getProjects()[0];
30          Layer[] layers = project.getLayerArray();
31          getMapPanel().setCurrentImageFormat("image/png");
32          getMapPanel().setCurrentProject(project);
33          getMapPanel().setLayers(layers);
34          getMapPanel().zoomToProject();
35          getMapPanel().updateView();
36      } catch (Exception ex) {
37          Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
38      }
39  }
```

Para ejecutar la aplicación se debe presionar F6, si todo va bien, la aplicación debe verse como en la figura 237, y al ejecutarse no debe arrojar ninguna excepción.

Figura 237. Apariencia de la clase Main.



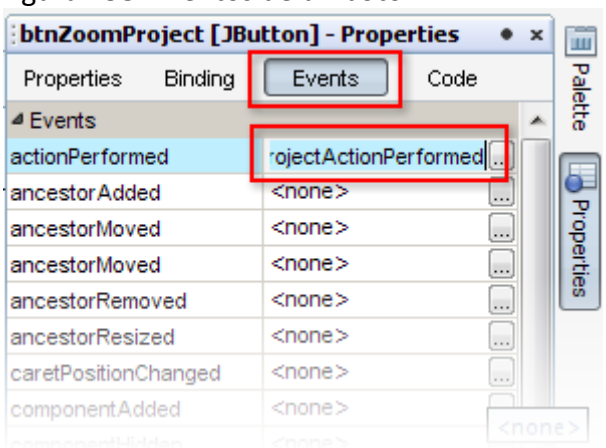
Interactuar con un mapa.

La forma de permitir interacción con un mapa, es mediante el uso de modos, en el ejemplo anterior, el mapa es estático, se agrega dinamismo mediante el uso de modos.

Los botones toggle ubicados sobre el mapa servirán para seleccionar el modo deseado, cada vez que el usuario presione un botón se cambiará dicho modo, para ello hay que agregar escuchadores a los eventos del botón.

En modo "design" debe hacerse clic sobre el botón al que se va a agregar el escuchador, luego, en la ventana de propiedades del botón (ver figura 238), bajo la opción de "Events", debe hacerse clic sobre la celda contigua a "actionPerformed" y presionar enter.

Figura 238. Eventos de un botón.



Los modos de operación disponibles para el MapPanel, se listan en la siguiente tabla.

Tabla 74. Modos de operación del MapPanel.

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.
NOMODESELECTED Modo de operación en que las acciones del usuario son ignoradas.

Luego para los botones en el formulario el código debe quedar como muestra la figura 239.

Figura 239. Métodos para interactuar con el Mapa.

```
    private void btnZoomProjectActionPerformed(java.awt.event.ActionEvent evt) {
        getMapPanel().zoomToProject();
}

    private void tglZoomInActionPerformed(java.awt.event.ActionEvent evt) {
        getMapPanel().setMode(MapPanel.MODEZOOMPLUS);
    }

    private void tglZoomOutActionPerformed(java.awt.event.ActionEvent evt) {
        getMapPanel().setMode(MapPanel.MODEZOOMMINUS);
    }

    private void tglZoomPointActionPerformed(java.awt.event.ActionEvent evt) {
        getMapPanel().setMode(MapPanel.MODEZOOMBOX);
    }

    private void tglPanActionPerformed(java.awt.event.ActionEvent evt) {
        getMapPanel().setMode(MapPanel.MODEPAN);
    }
```

Ahora, es cuestión de ejecutar la aplicación y probar el efecto de cada modo.

Procesar eventos generados por el control.

El MapPanel, produce eventos de los tipos mostrados en la tabla 75.

Tabla 75. Tipos de eventos generados por el MapPanel.

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.

A continuación se registra un escuchador para el evento MouseMoveEvent, de tal modo que pueda indicarse al usuario las coordenadas respecto al mapa de la posición del puntero de mouse.

Para ello debe crearse un método llamado registrarEscuchadores, que debe llamarse en el constructor de la clase, tal como se indica en la figura.

Figura 240. Llamado a registrarEscuchadores en el constructor la clase Main.

```
25
26  public Main() {
27      initComponents();
28      setMapPanel((MapPanel) jPanel);
29      regristrarEscuchadores();
30      try {
31          getMapPanel().setURL(new URL("http://127.0.0.1:80/server"));
32          Project project = getMapPanel().getProjects()[0];
33          Layer[] layers = project.getLayerArray();
34          getMapPanel().setCurrentImageFormat("image/png");
35          getMapPanel().setCurrentProject(project);
36          getMapPanel().setLayers(layers);
37          getMapPanel().zoomToProject();
38          getMapPanel().setEnabled(true);
```

El contenido del metodo registrar escuchadores debe ser como se muestra en la figura 241.

Figura 241. Contenido el metodo registrar escuchadores.

```
44
45  private void regristrarEscuchadores(){
46      MouseMoveListener mml = new MouseMoveListener() {
            public void actionPerformed(MouseMoveEvent mme) {
48              double x = mme.getXYCoordinate().getX();
49              double y = mme.getXYCoordinate().getX();
50              String msg = String.format("Posición %.3f%n , %.3f%n", x, y);
51              lblPos.setText(msg);
52          }
53      };
54
55
56      DisplayListener dl = new DisplayListener() {
            public void actionPerformed(DisplayEvent de) {
58              String msg = String.format("Escala %.3f%n", de.getWmsScale());
59              lblScale.setText(msg);
60          }
61      };
62
63      getMapPanel().getEventManager().addMouseMoveListener(mml);
64      getMapPanel().getEventManager().addDisplayListener(dl);
65  }
```

Al ejecutar la aplicación, los escuchadores registrados hacen que se complete la información en la parte inferior del formulario, permitiendo conocer la posición del puntero del mouse y la escala WMS de la presentacion, como se muestra en la figura.

Figura 242. Indicadores de posición y escala en Main.

Procesar información GetFeatureInfo.

El servidor Atlas entrega información GetFeatureInfo en formato HTML, GML y texto plano, para los formatos HTML y GML la biblioteca de componentes provee las herramientas de visualización HTMLViewer y GMLViewer que se encuentran en el paquete org.atlas.desk.viewers.

En este caso se usará el visor HTML para recibir respuestas en este formato, para ello, debe modificarse el método regristrarEscuchadores de modo que se vea como en el figura 243.

Figura 243. Modificaciones al método regristrarEscuchadores para GetFeatureInfo.

```
69
70      FeatureInfoListener fl = new FeatureInfoListener() {
71
72          public void actionPerformed(FeatureInfoEvent fi) {
73              HTMLViewer viewer = new HTMLViewer((JFrame)null, true);
74              viewer.setHtml(fi.getFeatureInfo());
75              viewer.setVisible(true);
76          }
77      };
78
79      getMapPanel().getEventManager().addMouseMoveListener(mml);
80      getMapPanel().getEventManager().addDisplayListener(dl);
81      getMapPanel().getEventManager().addFeatureInfoListener(fl);
82  }
```

El constructor también debe modificarse de modo que quede como en la figura 244.

Figura 244. Modificaciones al constructor para recibir respuestas GetFeatureInfo.

```
30  public Main() {
31      initComponents();
32      setMapPanel((MapPanel) jPanel);
33      regristrarEscuchadores();
34      try {
35          getMapPanel().setURL(new URL("http://127.0.0.1:80/server"));
36          Project project = getMapPanel().getProjects()[0];
37          Layer[] layers = project.getLayerArray();
38          getMapPanel().setCurrentImageFormat("image/png");
39          getMapPanel().setCurrentFeatureInfoFormat("text/html");
40          getMapPanel().setCurrentProject(project);
41          getMapPanel().setLayers(layers);
42          getMapPanel().zoomToProject();
```

El escuchador del boton tglInfo debe quedar como muestra la figura 245.

Figura 245. Escuchador del botón tglInfo.

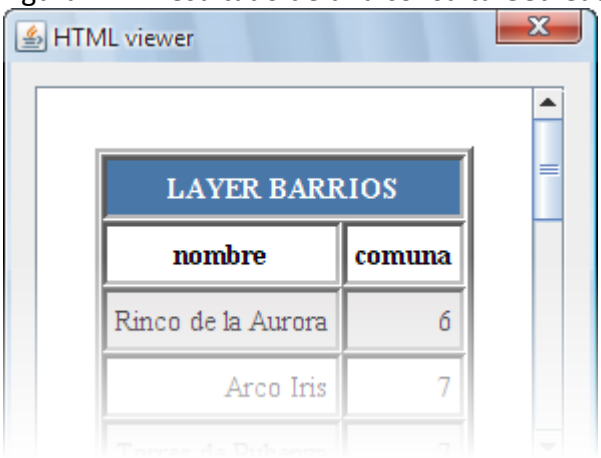
```
237
238     private void tglInfoActionPerformed(java.awt.event.ActionEvent evt) {
239         getMapPanel().setMode(MapPanel.MODEFEATUREINFO);
240     }
241
```

Con las modificaciones anteriores al establecer el modo de GetFeatureInfo y arrastrar el mouse en el mapa debe obtenerse un resultado como el mostrado en la figura 246.

Figura 246. Área de interés de una consulta GetFeatureInfo.

Al soltar el botón del mouse debe mostrarse una ventana como la siguiente.

Figura 247. Resultado de una consulta GetFeatureInfo.



Administrar marcadores.

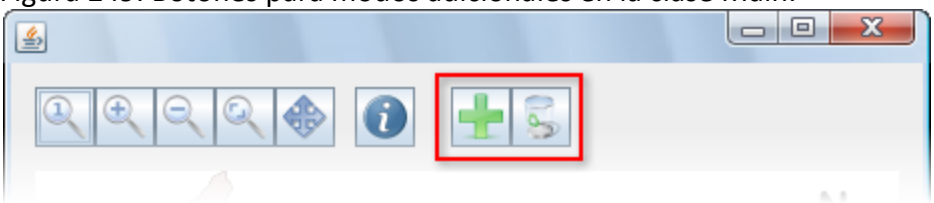
Un marcador es un punto que se agrega al mapa para resaltar una ubicación, en este ejemplo se va a permitir al usuario agregar puntos al mapa haciendo clic con el mouse, para ello se usa el modo MODEGETCOORD del MapPanel y se agrega dos modos a la clase Main, para ello, en los atributos de la clase debe agregarse el código que se muestra en la figura 248.

Figura 248. Modos para la clase Main.

```
21  import org.atlas.desk.events.MouseMoveListener;
22  import org.atlas.desk.gui.MapPanel;
23  import org.atlas.desk.mashup.Marker;
24  import org.atlas.desk.viewers.HTMLViewer;
25
26  public class Main extends javax.swing.JFrame {
27
28      public static final int ADDMARKER = 1;
29      public static final int REMOVEMARKER = 2;
30      private int currentMode;
31
32      private MapPanel mapPanel;
```

Además se deben agregar un par de botones toggle, y asignarlos al grupo, como se hizo anteriormente, el resultado debe verse como se muestra en la figura 249.

Figura 249. Botones para modos adicionales en la clase Main.



El código para estos botones debe ser como se muestra en la figura 250.

Figura 250. Código en los botones para modos adicionales en la clase Main.

```
313
314     private void tglMarkerAddActionPerformed(java.awt.event.ActionEvent evt) {
315         getMapPanel().setMode(MapPanel.MODEGETCOORD);
316         currentMode = ADDMARKER;
317 }
318
319     private void tglMarkerRemoveActionPerformed(java.awt.event.ActionEvent evt) {
320         getMapPanel().setMode(MapPanel.MODEGETCOORD);
321         currentMode = REMOVEMARKER;
322     }
323
```

El método regristrarEscuchadores también debe ser modificado agregando las siguientes líneas (ver figura 251).

Figura 251. Modificaciones al método regristrarEscuchadores para marcadores.

```
DisplayListener dl = new DisplayListener() {...};

FeatureInfoListener fl = new FeatureInfoListener() {...};

MouseClickListener mcl = new MouseClickListener() {

    public void actionPerformed(MouseClickEvent ev) {
        if (currentMode == ADDMARKER) {
            MapPanel mp = (MapPanel) ev.getSource();
            BufferedImage home = null;
            try {
                home = ImageIO.read(this.getClass().getResourceAsStream("/org/atlas/desk/media/home.png"));
            } catch (IOException ex) {
                Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
            }

            Marker m = new Marker(mp, ev.getMap(), home);
            mp.addMarker(m);
        }
    }
};

MarkerClickedListener macl = new MarkerClickedListener() {

    public void actionPerformed(MarkerClickedEvent ev) {
        if (currentMode == REMOVEMARKER) {
            MapPanel mp = (MapPanel) ev.getSource();
            mp.removeMarker(ev.getMarker());
        }
    }
};

getMapPanel().getEventManager().addMouseMoveListener(mml);
getMapPanel().getEventManager().addDisplayListener(dl);
getMapPanel().getEventManager().addFeatureInfoListener(fl);
getMapPanel().getEventManager().addMouseClickListener(mcl);
getMapPanel().getEventManager().addMarkerClickedListener(macl);
```

Una vez aplicados estos cambios, al seleccionar el modo agregar marcador y hacer clic sobre el mapa, se agrega un marcador, y el seleccionar el modo remover marcador y hacer clic sobre un marcador, este es removido.

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, el método regristrarEscuchadores debe modificarse agregando el código que muestra la figura 252.

Figura 252. Modificaciones al método regristrarEscuchadores para geocoding.

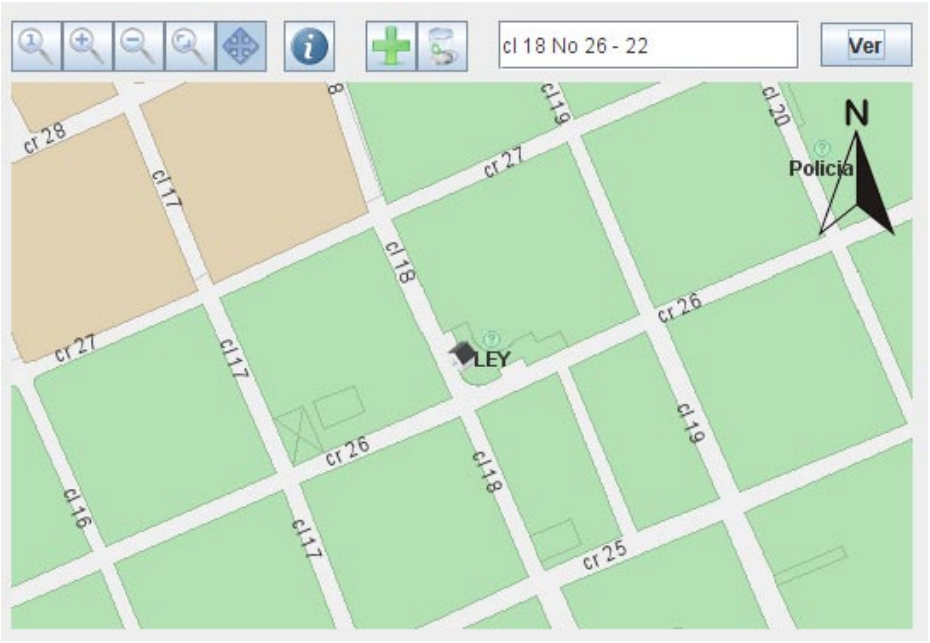
```
GeocodingListener gl = new GeocodingListener() {

    public void actionPerformed(GeocodingEvent ge) {
        MapPanel mp = (MapPanel) ge.getSource();
        Point2D pt = ge.getAddresses()[0].getLocation();
        mp.setCenter(pt);
        mp.setZoomLevel(-15);
        mp.updateView();
        BufferedImage home = null;
        try {
            String imgPath = "/org/atlas/desk/media/home.png";
            InputStream is = this.getClass().getResourceAsStream(imgPath);
            home = ImageIO.read(is);
            is.close();
        } catch (IOException ex) {
            Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex);
        }
        Marker m = new Marker(mp, pt, home);
        mp.removeAllMarkers();
        mp.addMarker(m);
    }
};

getMapPanel().getEventManager().addMouseMoveListener(mml);
getMapPanel().getEventManager().addDisplayListener(dl);
getMapPanel().getEventManager().addFeatureInfoListener(fl);
getMapPanel().getEventManager().addMouseClickListener(mcl);
getMapPanel().getEventManager().addMarkerClickedListener(macl);
getMapPanel().getEventManager().addGeocodingListener(gl);
```

Se debe agregar una caja de texto y un botón a Main, como indica la figura 253.

Figura 253. Componentes de Main, para peticiones al geocoder.

Luego de aplicar estos cambios, al escribir una dirección en la caja de texto y hacer clic sobre el botón “ver”, la dirección geocodificada debe verse en el mapa representada por un marcador, como se muestra en la figura 254.

Figura 254. Geocodificación de una dirección.



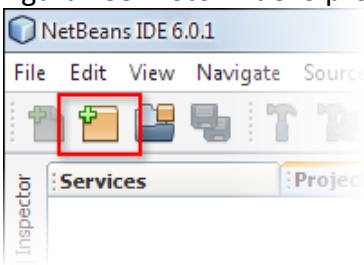
A7. GUÍA DE DESARROLLO DE APLICACIONES JAVA ME.

En esta guía 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.

Configuración de entorno.

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

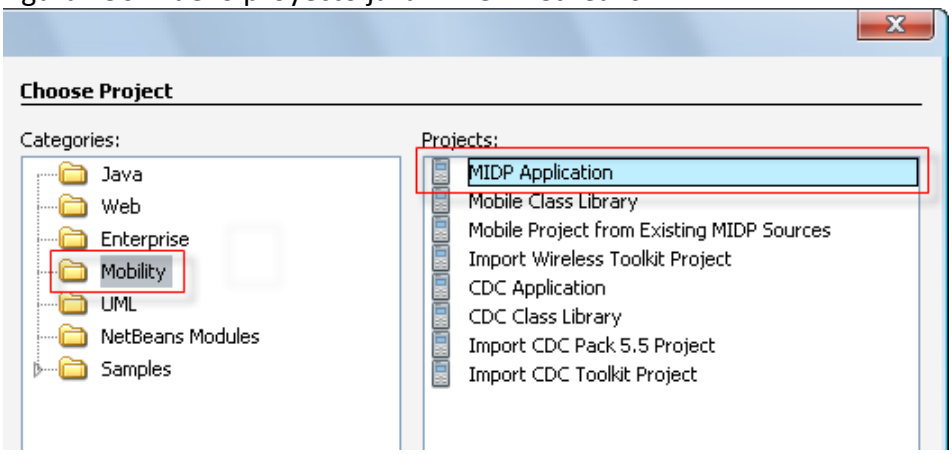
Figura 255. Botón nuevo proyecto en NetBeans.



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 siguiente, finalmente debe hacer clic en “next”.

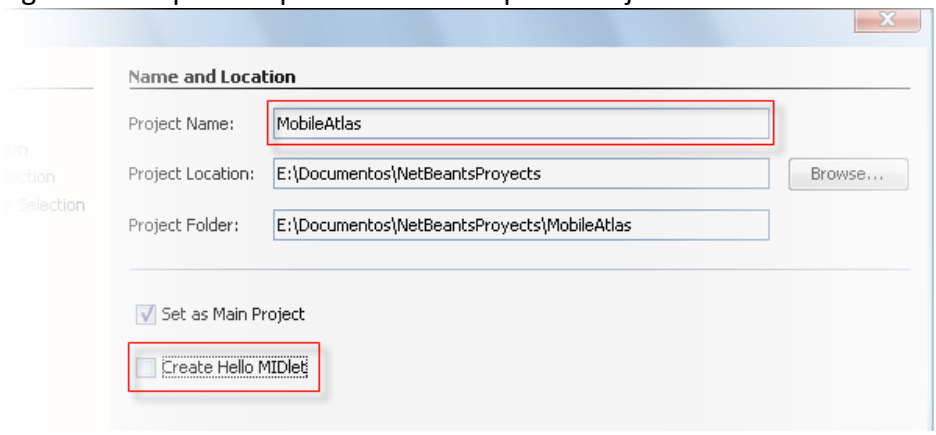
Nota. Si la categoría Mobility no se lista, vaya al menú Tools, Pluings y actualice la versión de NetBeans, agregando el paquete Mobility.

Figura 256. Nuevo proyecto java ME en NetBeans.



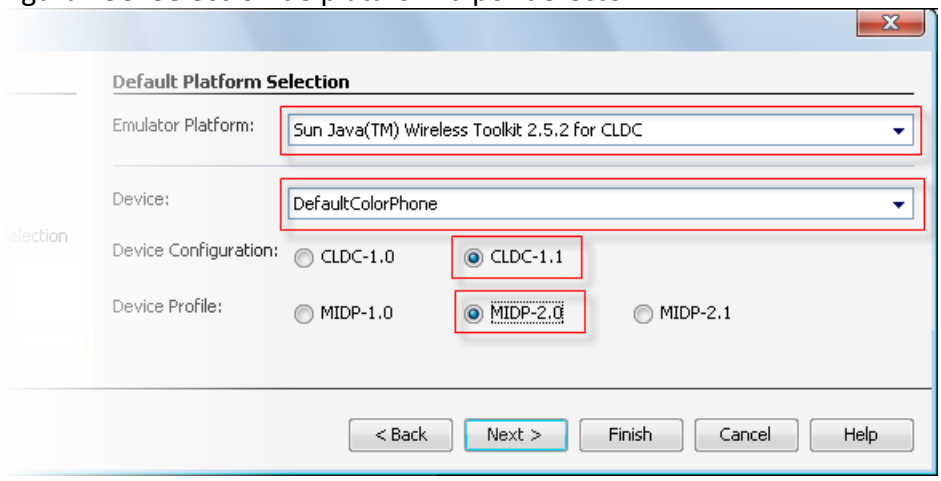
En el diálogo siguiente se indica el nombre del proyecto y se indica a NetBeans que no debe crear una clase principal, luego debe hacerse click en next.

Figura 257. Opciones para una nueva aplicación java en NetBeans.



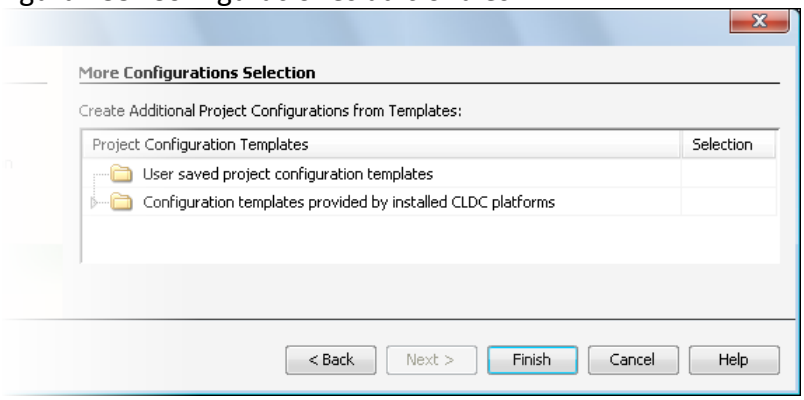
El diálogo siguiente indica el tipo de emulador, dipositvo, configuracción del dipositivo y perfil del dispositivo, que se deben seleccionar como se muestra en la figura, luego debe hacerce clic en next.

Figura 258. Selección de plataforma por defecto.



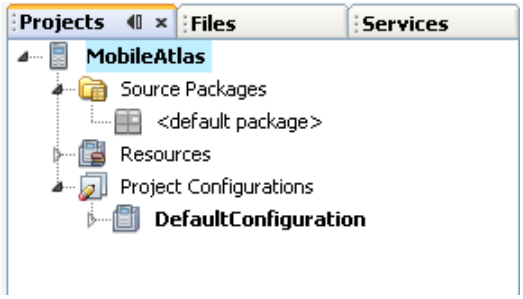
El diálogo siguiente agrega más configuraciones al proyecto, para este ejemplo se usan las opciones por defecto, como se muestra en la siguiente figura, finalmente debe hacer clic finalizar.

Figura 259. Configuraciones adicionales.



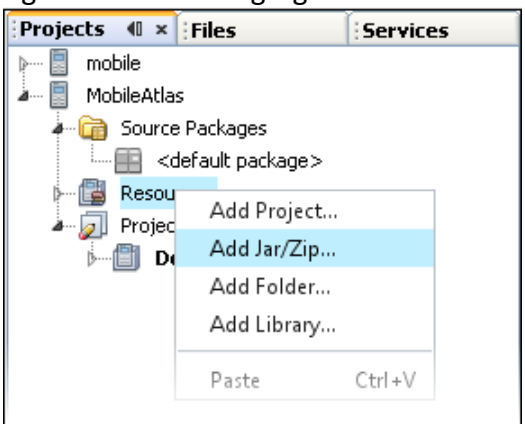
El proyecto aparece ahora en el árbol de proyectos.

Figura 260. Árbol de proyecto en NetBeans.



Luego se deben agregar los recursos requeridos. Para ello debe desplegarse el menú que se muestra en la figura 261.

Figura 261. Menu agregar recurso en NetBeans.

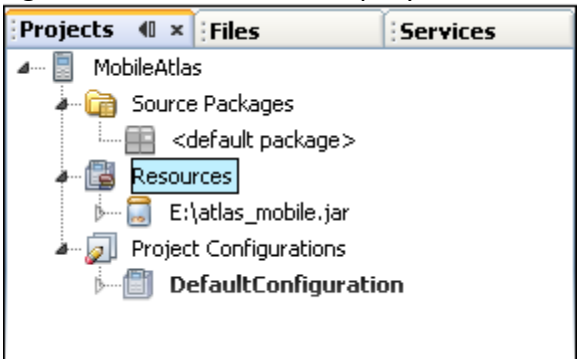


Luego, debe agregarse el siguiente archivo.

- atlas_mobile.jar

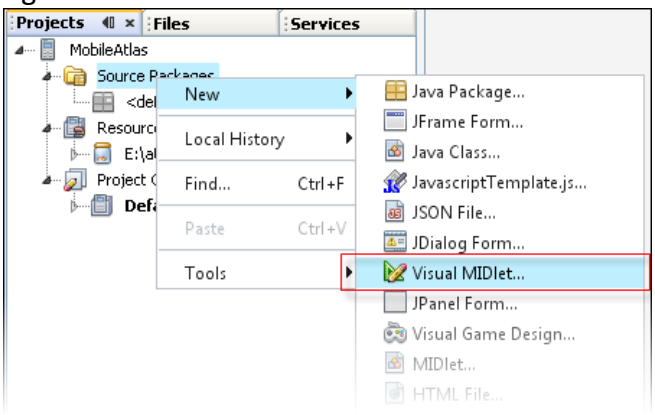
El árbol de proyecto debe verse como se muestra en la figura.

Figura 262. Resource en un proyecto NetBeans.



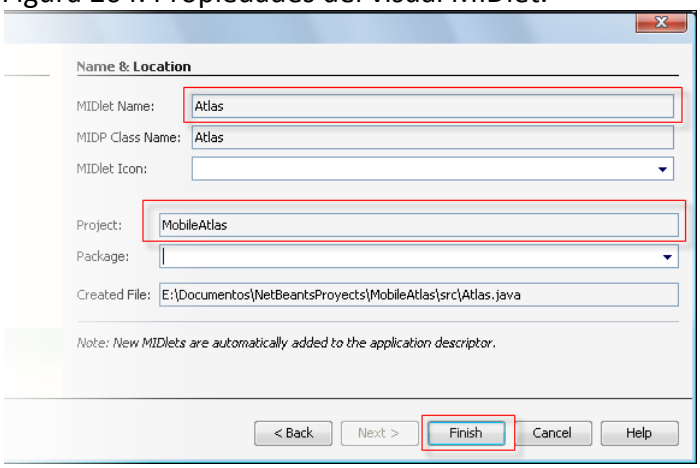
Ahora, se debe crear un “Visual MIDlet” en la aplicación, usando el menú que se muestra en la figura.

Figura 263. Creación de un Visual MIDlet en NetBeans.



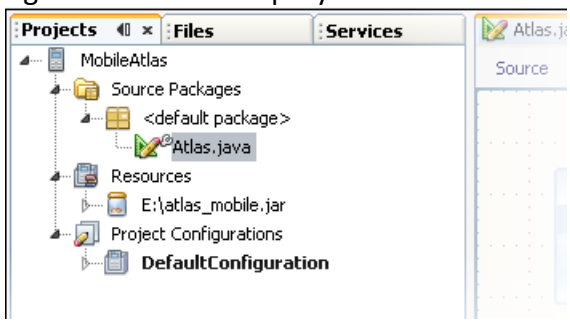
En la ventana que aparece, se debe establecer el nombre del MIDlet para este ejemplo denominado “Atlas”, como se muestra en la figura y luego se debe hacer clic en finish.

Figura 264. Propiedades del visual MIDlet.



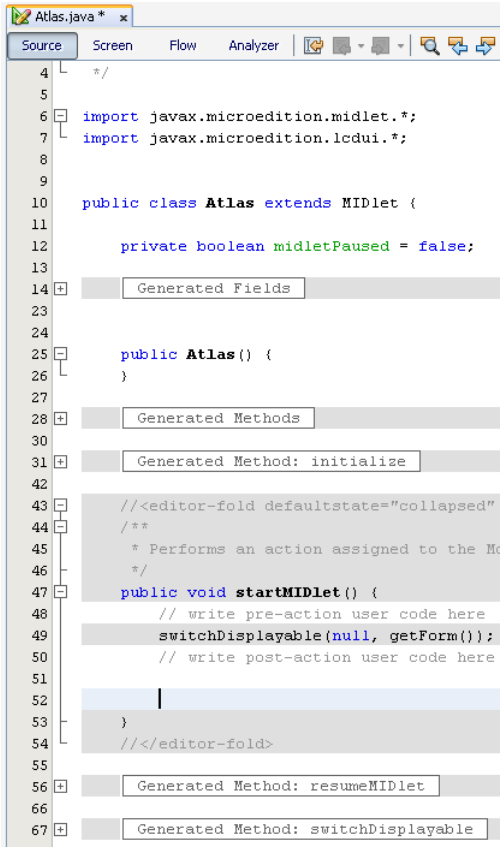
El árbol del proyecto debe verse ahora como indica la figura.

Figura 265. Árbol de proyecto en NetBeans con un MIDlet nuevo.



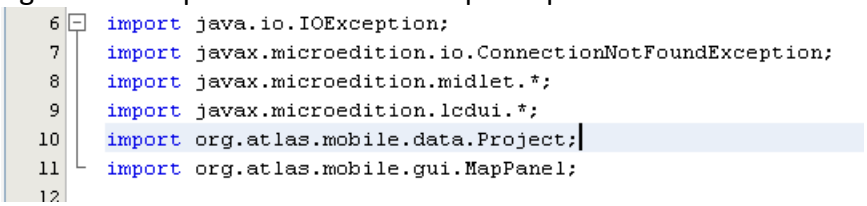
Al hacer doble clic sobre el nodo Atlas.java, este se mostrará en el recuadro central, debe hacerse clic sobre el botón “Source” en caso de que este no esté marcado.

Figura 266. Vista Source en NetBeans.



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

Figura 267. Imports biblioteca Atlas para apliciones Java ME.



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

Figura 268. Implementación de CommandListener.

```
public class Atlas extends MIDlet implements CommandListener {
```

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

Figura 269. Atributos de la clase MIDlet.

```
public class Atlas extends MIDlet {

    private boolean midletPaused = false;

    private Form form;
    private MapPanel mapPanel;
    private Command mode;

    Generated Fields
```

El constructor de MIDlet Atlas debe modificase así.

Figura 270. Constructor de la clase Atlas.

```
public Atlas() {
    mode = new Command("No mode", Command.SCREEN, 1);
    exit = new Command("Exit", Command.SCREEN, 1);
}
```

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

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.

Figura 271. Código de acceso a una instancia de MapPanel.

```
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 272. Código de acceso a una instancia de Formulario.

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

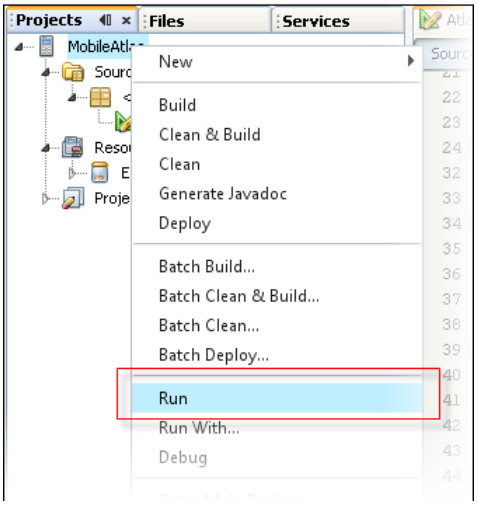
Para lanzar la aplicación se debe modificar el método startMIDlet como se muestra en la siguiente figura.

Figura 273. Código de lanzamiento del MIDlet.

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

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

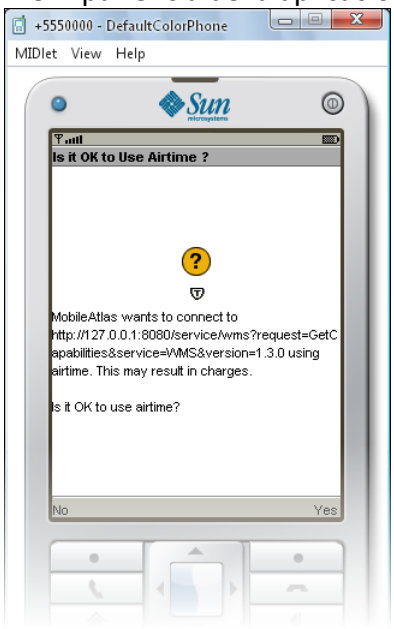
Para ejecutar la aplicación se debe desplegar el menú del proyecto MobileAtlas y seleccionar la opción “Run”, como se muestra en la figura.

Figura 274. Submenu para ejecutar la aplicación.



Si todo esta correcto, la aplicación debe verse como en la figura, 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 275. Apariencia de la aplicación MobileAtlas.



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 el componente tiene la capacidad de acercarse, alejarse y desplazarse por el mapa.

Figura 276. Código para administrar el manejo de modos.

```
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());

}
```

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, se listan en la siguiente tabla.

Tabla 76. Modos de operación del MapPanel Java ME.

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.

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.

Figura 277. Modificación al método startMIDlet para GetFeatureInfo.

```
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());
                }
            }
        });

    }
});
```

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

Figura 278. Modificación código administrador de modos.

```
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());

}
```

El constructor también debe modificarse como se muestra en la figura 279.

Figura 279. Modificación al metodo getMapPanel para FeatureInfo.

```
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;
}
```

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.

Figura 280. 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.

Figura 281. Respuesta getFeatureInfo en formato texto plano.

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.

Figura 282. Modificación al método startMIDlet para detectar toques en la pantalla.

```
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();
        }
    }
});
```

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

Figura 283. Código para agremas el modo MODEGETCOORD.

```
    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;
```

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

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.

Figura 284. Modificaciones al método startMIDlet para manejo de geocoding.

```
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();
        }
    }
});
```

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

Figura 285. Código para agregar botón del geocoder.

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

Figura 286. Código para enviar la petición al servidor de geocodificación.

```
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;
}
```

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.

Figura 287. Apariencia de la aplicación respuesta geocoder.

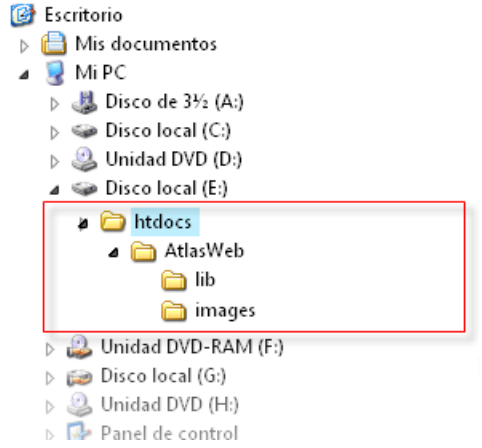
A8. GUÍA DE DESARROLLO DE APLICACIONES WEB.

En esta guía se cubre la realización de una sencilla aplicación para navegadores web que demuestra los conceptos básicos sobre la creación de aplicaciones con la biblioteca web de Atlas.

Configuración de entorno.

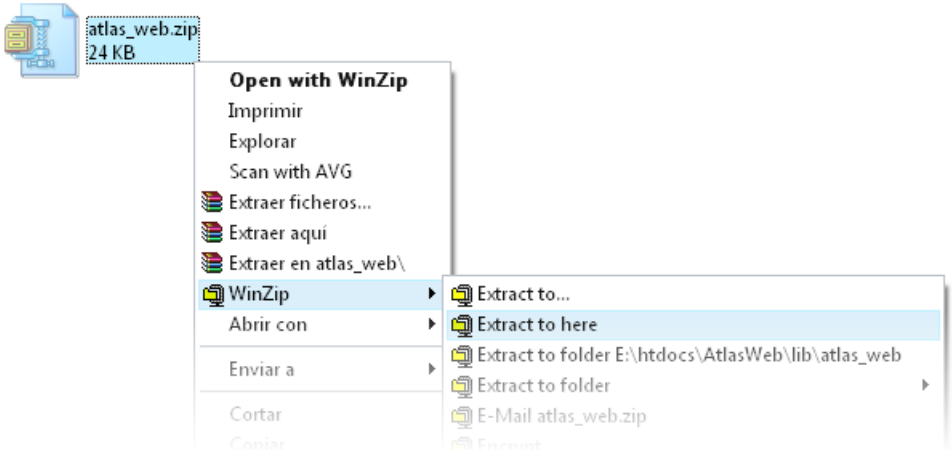
Para realizar la configuración del entorno se forma una estructura de directorios como se muestra a continuación.

Figura 288. Estructura de directorios aplicación web



En la carpeta lib se copia y se descomprime el archivo zip atlas_web.zip como se muestra en la figura.

Figura 289. Descomprimir el archivo atlas_web.zip



En la carpeta AtlasWeb se debe crear un archivo denominado index.html, se agrega una carpeta images con una imagen pequeña (20x21) en formato PNG, la carpeta AtlasWeb tendrá el contenido que se muestra en la figura.

Figura 290. Crear archivo index.html.

Se debe abrir el archivo index.html con un editor de código, agregando el código que se muestra en la siguiente figura.

Figura 291. Código básico de una pagina html.

```
1  <html xmlns="http://www.w3.org/1999/xhtml">
2      <head>
3          <title>Atlas Web</title>
4      </head>
5
6      <body>
7      </body>
8  </html>
9
```

Se debe iniciar agregado en la etiqueta <head> una etiqueta <script> indicado la ubicación del archivo “atlas.js” archivo fundamental para el manejo de la biblioteca Web de Atlas, como se muestra en la siguiente figura.

Figura 292. Código para incluir la biblioteca atlas.js

```
1  <html xmlns="http://www.w3.org/1999/xhtml">
2
3      <head>
4          <title>Atlas Web</title>
5          <script src="lib/atlas.js"></script>
6      </head>
7
8      <body>
9      </body>
10 </html>
11
```

En este ejemplo para iniciar la funcionalidad de la página se crea una función init() para lanzarse en el método onload de la etiqueta <body>, como se muestra en la siguiente figura.

Figura 293. Código de inicio para la carga de la página

```
1  <html xmlns="http://www.w3.org/1999/xhtml">
2
3      <head>
4          <title>Atlas Web</title>
5          <script src="lib/atlas.js"></script>
6
7          <script language="javascript">
8              function init(){
9
10             }
11         </script>
12
13     </head>
14
15     <body onLoad="init();" >
16
17     </body>
18 </html>
```

Para la visualización del mapa se crea una etiqueta estableciendo el identificador del div, el borde y las dimensiones como se muestra en la siguiente figura.

Figura 294. Código del div del mapa.

```
<body onLoad="init();" >
    <div id="map" style=" width:600px; height:500px; border:#0066FF 1px solid;"  ></div>
</body>
```

Al realizar correctamente los anteriores pasos, la configuración del entorno está lista para tomarse como base para continuar con el manejo del MapPanel.

Conectarse con un servidor.

Para esta y las siguientes secciones se asume que existe un servidor Atlas instalado y funcionando en la dirección http://192.168.10.111:8080/service.

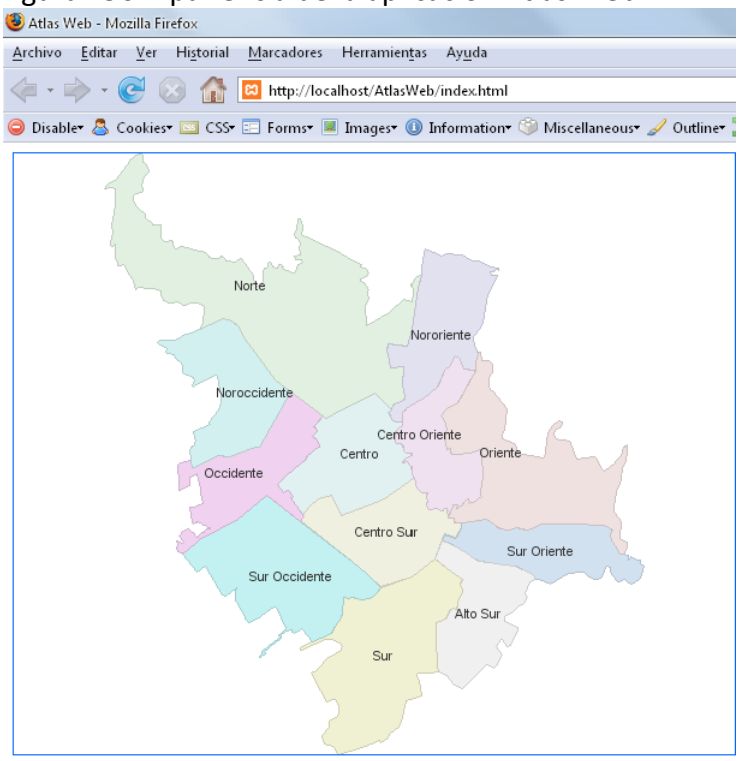
Para conectar al servidor se debe agregar en la función init(), el código que se muestra a continuación.

Figura 295. Código conectarse al servidor

```
<script language="javascript">
    var map = null;
    function init(){
        map = new atlas.MapPanel("map","lib");
        map.setURL("http://192.168.10.111:8080/service");

        map.addListener("onloaddata", function(){
            var project = map.getProjects()[0];
            map.setCurrentProject(project);
            var layers = project.getLayers();
            map.setLayers(layers);
            map.updateView();
        });
    }
</script>
```

Para ejecutar la aplicación se debe hacer doble clic en el archivo indenx.html, Si todo esta correcto, la aplicación debe verse como en la figura 296, y al ejecutarse no debe presentar ningún error Javascript.

Figura 296. Apariencia de la aplicación Atlas Web.



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 el componente permite al usuario la capacidad de acercarse, alejarse y desplazarse por el mapa.

Para lograr esto se debe agregar algunos botones, para intercambiar cada uno de los modos del MapPanel. El código se muestra en la siguiente figura.

Figura 297. Código botones de modo MapPanel.

```
<body onLoad="init();" >

    <div id="map" style=" width:600px; height:500px; border:#0066FF 1px solid;"  ></div>

    <div style="margin:10px;">
        <input type="button" value="SIN MODO" onclick="map.setMode('NOMODESELECTED');" />
        <input type="button" value="MODO ZOOMPLUS" onclick="map.setMode('MODEZOOMPLUS');" />
        <input type="button" value="MODO ZOOMMINUS" onclick="map.setMode('MODEZOOMMINUS');" />
        <input type="button" value="MODO PAN" onclick="map.setMode('MODEPAN');" />
        <input type="button" value="MODO ZOOMBOX" onclick="map.setMode('MODEZOOMBOX');" />
    </div>

</body>
```

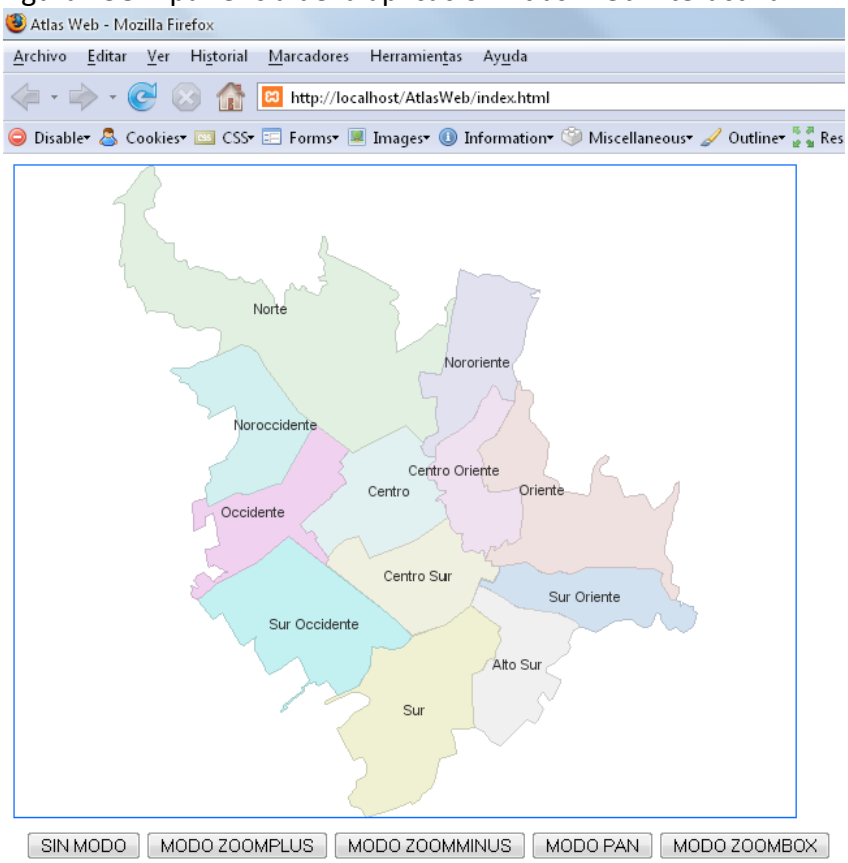
Los modos de operación disponibles para el MapPanel, se listan en la siguiente tabla.

Tabla 77. Modos de operación del MapPanel Web 2.0.

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 mouse.

Figura 298. Apariencia de la aplicación Atlas Web interactiva.



Procesar eventos generados por el control.

A continuación se registra un escuchador para el evento MouseMove, de tal modo que pueda indicarse al usuario las coordenadas respecto al mapa de la posición del puntero de mouse.

Figura 299. Código Escuchador mouse move.

```
map.addListener("onmousemove", function(){
    var coor = atlas.CoordinateUtils.translateIJtoXY(  map.getVisibleBbox(),
                                                       map.getWidth(),
                                                       map.getHeight(),
                                                       map.getCurrentMousePosition().getX(),
                                                       map.getCurrentMousePosition().getY(),
                                                       map.getCurrentProject().getEpsgCrs());
    document.getElementById("log").innerHTML = coor.getX() + " , " + coor.getY();
});
```

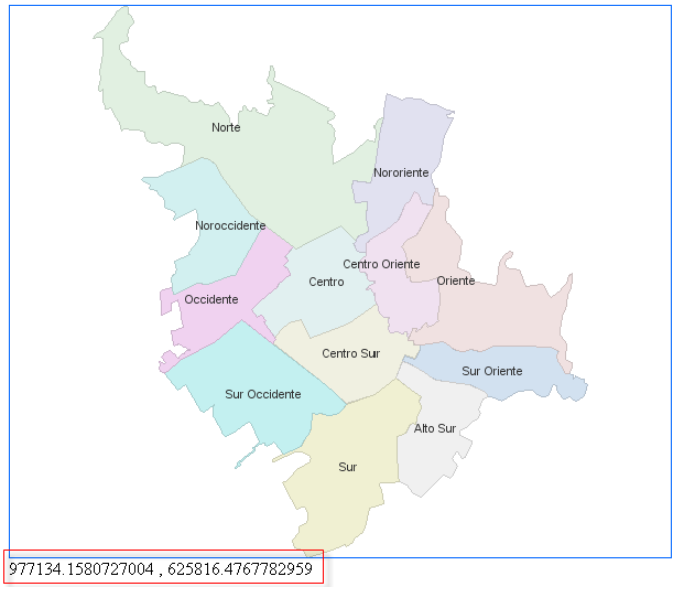
Para mostrar el contenido es necesario crear un etiqueta con el identificador “log” como se muestra en la siguiente figura.

Figura 300. Div de visualización de coordenadas.

```
<div id="log"></div>
```

Al ejecutar la aplicación, El escuchador registrado hace que se complete la información en la parte inferior de la pagina, permitiendo conocer la posición del puntero del mouse, como se muestra en la figura.

Figura 301. Indicadores de posición del mapa.



Procesar información GetFeatureInfo.

El servidor Atlas entrega información GetFeatureInfo en formato HTML, GML y texto plano, para este ejemplo se utiliza una nueva ventana para recibir respuestas en formato HTML.

Para esto se debe agregar el código seleccionado al MapPanel como se muestra en la figura.

Figura 302. Código configuración FeatureInfo.

```
<script language="javascript">
    var map = null;
    function init(){
        map = new atlas.MapPanel("map","lib");
        map.setURL("http://192.168.10.111:8080/service");

        map.setFeatureInfoTarget("window","FeatureInfo","height=350");
        map.setCurrentFeatureInfoFormat("text/html");

        map.addListener("onloaddata", function(){
            var project = map.getProjects()[0];
            map.setCurrentProject(project);
            var layers = project.getLayers();
            map.setLayers(layers);
            map.updateView();
        });
    }
</script>
```

Para lanzar la petición GetFeatureInfo debe agregarse un nuevo botón cuya función será establecer el modo MODEFEATUREINFO como se muestra en la siguiente figura.

Figura 303. Código para seleccionar el modo FeatureInfo

```
<div style="margin:10px;">
    <input type="button" value="SIN MODO" onclick="map.setMode('NOMODESELECTED');" />
    <input type="button" value="MODO ZOOMPLUS" onclick="map.setMode('MODEZOOMPLUS');" />
    <input type="button" value="MODO ZOOMMINUS" onclick="map.setMode('MODEZOOMMINUS');" />
    <input type="button" value="MODO PAN" onclick="map.setMode('MODEPAN');" />
    <input type="button" value="MODO ZOOMBOX" onclick="map.setMode('MODEZOOMBOX');" />

    <input type="button" value="MODO FEATURE INFO" onclick="map.setMode('MODEFEATUREINFO');" />

</div>
```

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

Figura 304. Área de interés de una consulta GetFeatureInfo.

Al finalizar el arrastre con el mouse se debe mostrar una nueva ventana con el resultado de la consulta en formato HTML. Como se muestra en la siguiente figura.

Figura 305. Resultado HTML de la petición GetFeatureInfo.

LAYER BARRIOS	
nombre	**comuna**
Arco Iris	7
Torres de Pubenza	7
La Aurora	6
Javerianito	6
San Ignacio	6
Los Hexagonos	6
La Castellana	7

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 haciendo clic en el mapa, para ello se usa el modo MODEGETCOORD del MapPanel y un escuchador que registra el evento clic.

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

Figura 306. Escuchador del evento clic del MapPanel

```
map.addListener("onclick", function(){
    var coor = atlas.CoordinateUtils.translateIJtoXY(  map.getVisibleBbox(),
                                                       map.getWidth(),
                                                       map.getHeight(),
                                                       map.getCurrentMousePosition().getX(),
                                                       map.getCurrentMousePosition().getY(),
                                                       map.getCurrentProject().getEpsgCrs());

    var marker = new atlas.Marker(coor,
                                  "", "images/home.gif",
                                  "images/home.gif", "images/home.gif",
                                   20, 21);
    map.addMarker(marker);
});
```

Para agregar un nuevo marcador debe agregarse un nuevo botón cuya función será establecer el modo MODEGETCOOR como se muestra en la siguiente figura.

Figura 307. Código para seleccionar el modo MODEGETCOOR

```
<div style="margin:10px;">
    <input type="button" value="SIN MODO" onclick="map.setMode('NOMODESELECTED');" />
    <input type="button" value="MODO ZOOMPLUS" onclick="map.setMode('MODEZOOMPLUS');" />
    <input type="button" value="MODO ZOOMMINUS" onclick="map.setMode('MODEZOOMMINUS');" />
    <input type="button" value="MODO PAN" onclick="map.setMode('MODEPAN');" />
    <input type="button" value="MODO ZOOMBOX" onclick="map.setMode('MODEZOOMBOX');" />
    <input type="button" value="MODO FEATURE INFO" onclick="map.setMode('MODEFEATUREINFO');" />

    <input type="button" value="MODO GETCOORD" onclick="map.setMode('MODEGETCOORD');" />

</div>
```

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

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.

Figura 308. Escuchador para la petición de geocoding.

```
map.addListener("onresponsegeocoder", function(){
    var resg = map.getResponseGeocoder();
    if(resg.getCoordinates() != null){
        if(resg.getCoordinates().length > 0){
            var res = resg.getCoordinates()[0];
            var point = new Point(res.getX(), res.getY());

            map.setCenter(point);
            map.setZoomLevel(-10);
            map.updateView();

            var marker = new atlas.Marker(point, "", "images/home.gif",
                                          "images/home.gif", "images/home.gif", 20, 21);
            map.addMarker(marker);

        }
    }else{
        alert("El Geocoder no logro resultados para la direccion " + resg.getRequest());
    }
});
```

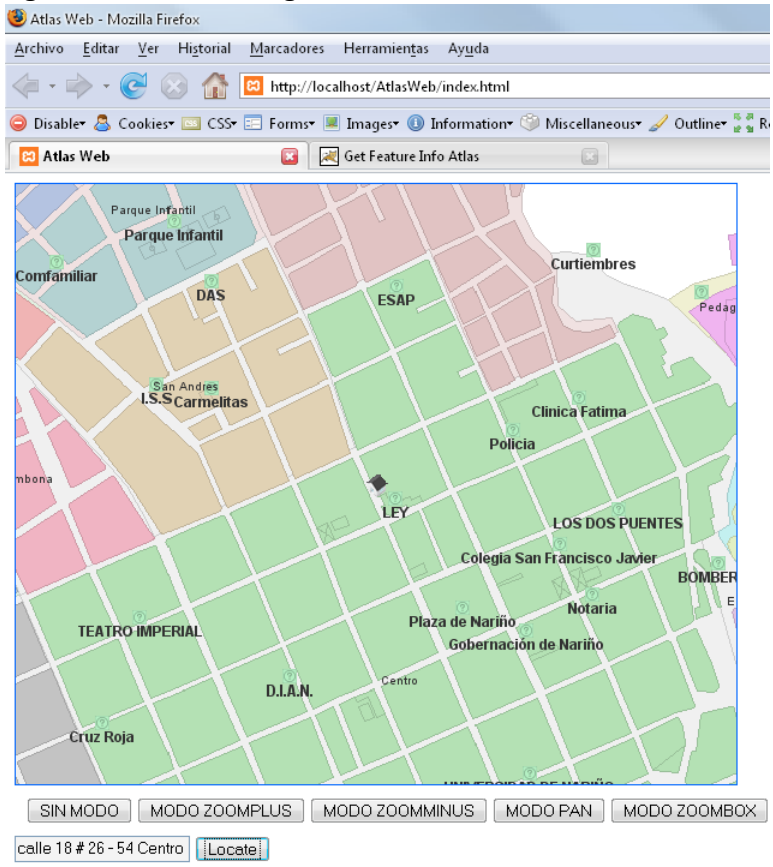
Para acceder a la opción de geocoder, se agrega una caja de texto para escribir la dirección y un botón para enviar la petición al servidor, con el código que se muestra en la figura.

Figura 309. Código para agregar botón del geocoder.

```
<input id="address" type="text" value="calle 18 # 26 - 54 Centro" />
<input type="button" value="Locate"
 onclick="map.sendGeocoderRequest(document.getElementById('address').value);" />
```

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

Figura 310. Dirección geocodifica.



A9. MARCAS REGISTRADAS.

- Geoserver es una marca registrada por The Open Planning Project.
- Deegree es una marca registrada por GIS Research Group of the Department of Geography of University of Bonn.
- MapServer es una marca registrada por Regents of the University of Minnesota.
- ARCIMS ™ es una marca registrada por ESRI Corp.
- OpenLayers es una marca registrada por MetaCarta Inc.
- GOOGLE MAP ™ es una marca registrada por GOOGLE.
- GOOGLE Mobile™ es una marca registrada por GOOGLE.
- NetBeans™ es una marca registrada por Sun MicroSystems.
- Eclipse es una marca registrada por Eclipse Foundation.
- Quantum GIS ™ es una marca registrada por Quantum GIS.
- Cad2Shape es una marca registrada por Guthrie.
- JAVA ™ es una marca registrada por Sun Microsystems.
- Java topology suite es una marca registrada por Vivid Solutions, Inc.
- Geotools es una marca registrada por Codehaus Foundation.
- Hibernate es una marca registrada por Red Hat Middleware.
- Jdom es una marca registrada por Jdom.
- KXML es una marca registrada por Stefan Haustein.
- PostgreSQL es una marca registrada por PostgreSQL Global Development Group.
- Postgis es una marca registrada por Refractions Research.
- Tomcat™. es una marca registrada por Apache Software Foundation.
- Unix® es un marca registrada de The Open Group.
- Windows™ es un marca registrada por Microsoft Corp.

REFERENCIAS

[1] OGP Surveying & Positioning Committee, “EPSG Geodetic Parameter Dataset”, en http://www.epsg.org/Geodetic.html, Consultado el 1 de Noviembre de 2007.

[2] Environmental Systems Research Institute, Inc. (July, 1998). "ESRI Shapefile technical description". Consultado el 2007-07-04.

[3] Open Geospatial Consortium, Inc. "OpenGIS Implementation Specification #02-023r4: OpenGIS® Geography Markup Language (GML) Implementation Specification, version 3.1.1", January 2005.

[4] Open Geospatial Consortium, Inc. (Octubre 2006). “OpenGIS® Implementation Specification for Geographic information - Simple feature access - Part 1: Common architecture”. Consultado el 1 de Noviembre de 2007.

[5] Kraak, Menno Jan (2001): Settings and needs for Web cartography, in: Kraak and Allan Brown (eds), Web Cartography, Francis and Taylor, New York.

[6] Open Geospatial Consortium, Inc. (Marzo 2006). “OpenGIS® Web Map Server Implementation Specification”. Consultado el 1 de Noviembre de 2007.

[7] Ort, E. Brydon, S. Basler, M. “Mashup Styles, Part 1: Server-Side Mashups”, en http://java.sun.com/developer/technicalArticles/J2EE/mashup_1/, consultado el 8 de Mayo de 2007.

[8] Wikipedia, "Geocodificación", en http://es.wikipedia.org/wiki/Geocodificación, consultado el 22 de Julio de 2007.

[9] Clodoveu, R. Torres, F. Albuquerque K, "A Flexible Addressing System for Approximate Geocoding", en http://www.geoinfo.info/geoinfo2003/papers/geoinfo2003-25.pdf, consultado el 6 de Octubre de 2007

[10] Environmental Systems Research Institute, "An overview of linear referencing", en http://Webhelp.esri.com/arcgisdesktop/9.2/index.cfm?TopicName=An_overview_of_linear_referencing, consultado el 19 de Julio de 2007

[11] GeoServer, "GeoServer Home ", en http://geoserver.org/display/GEOS/GeoServer+Home, consultado el 14 de Octubre de 2007

[12] lat/lon, "deegree - Free Software for Spatial Data Infrastructures ", en http://www.deegree.org/, consultado el 13 de Septiembre de 2007

[13] University of Minnesota , "Welcome to MapServer", en http://mapserver.gis.umn.edu/, consultado el 28 de Septiembre de 2007

[14] Environmental Systems Research Institute, "ArcIMS", en http://www.esri.com/software/arcgis/arcims/index.html, consultado el 6 de Septiembre de 2007

[15] MetaCarta, "OpenLayers", en http://www.openlayers.org/, consultado el 14 de Julio de 2007

[16] Google, "Acerca de Google Maps ", en http://maps.google.es/support/bin/topic.py?topic=10778, consultado el 27 de Agosto de 2007

[17] Google, "Accede a Google en cualquier momento y lugar", en http://www.google.com/mobile/, consultado el 28 de Julio de 2007

[18] Google, "Geocoding", en http://code.google.com/apis/maps/documentation/services.html#Geocoding, consultado el 28 de Septiembre de 2007

[19] Sun Microsystems, "NetBeans IDE 6.0 Features", en http://www.netbeans.org/features/, consultado el 25 de Agosto de 2007

[20] The Eclipse Foundation, "Eclipse - an open development platform", en http://www.eclipse.org/, consultado el 20 de Agosto de 2007

[21] Quantum GIS, "Quantum GIS", en http://www.qgis.org/, consultado el 7 de Agosto de 2007

[22] Guthrie , "Cad2Shape 3.0", en http://www.guthcad.com.au/cad2shape.htm, consultado el 24 de Agosto de 2007

[23] Wikipedia, "Plataforma Java", en http://es.wikipedia.org/wiki/Plataforma_Java, consultado el 22 de Octubre de 2007

[24] Sun Microsystems, "Java SE at a Glance", en http://java.sun.com/javase/, consultado el 14 de Septiembre de 2007

[25] Sun Microsystems, "Java ME at a Glance", en http://java.sun.com/javame/index.jsp, consultado el 18 de Agosto de 2007

[26] Sun Microsystems, "Java EE at a Glance", en http://java.sun.com/javaee/, consultado el 13 de Septiembre de 2007

[27] Vivid Solutions, "JTS Topology Suite ", en http://www.vividsolutions.com/jts/jtshome.htm, consultado el 17 de Septiembre de 2007

[28] GeoTools, "The Open Source Java GIS Toolkit", en http://geotools.codehaus.org/, consultado el 18 de Septiembre de 2007

[29] Red Hat Middleware, "Hibernate Core for Java", en http://www.hibernate.org/344.html, consultado el 10 de Septiembre de 2007

[30] PostgreSQL Global Development Group , "About", en http://www.postgresql.org/about/, consultado el 28 de Octubre de 2007

[31] Refractions Research , "What is PostGIS?", en http://www.postgis.org/, consultado el 7 de Octubre de 2007

[32] Wells, D, "Extreme Programming: A gentle introduction. ", en http://www.extremeprogramming.org/, consultado el 21 de Septiembre de 2007

[33] Ambysoft Inc, "Agile Modeling (AM) Home Page", en http://www.agilemodeling.com/, consultado el 4 de Octubre de 2007