UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 6

TÍTULO: Representación cartográfica de información contenida en archivos Mapfile utilizando la herramienta QGis.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: Elisa Cabrera Batista.
TUTOR: Msc. David Silva Barrera.

Ciudad de la Habana 2011.
Año 53 de la Revolución

"....En sus viajes por los pequeños planetas de su galaxia se encontró con un geógrafo que anotaba, en un gran libro de registro: montañas, ríos y estrellas.
El Principito quiso registrar a su flor (aquella que había dejado en su planeta), pero el geógrafo le dijo:
- No registramos flores, porque no se puede tomar como referencia a las cosas efímeras -.
Y el geógrafo le explicó al Principito que efímero quiere decir amenazado de pronta desaparición.
Cuando el principito escuchó esto, se entristeció mucho. Se había dado cuenta de que su rosa era efímera...."

"....Verdades que continúan siendo ciertas a través del tiempo y de las circunstancias...
...Conceptos que no son relativos a determinados momentos, sino a todos y cada uno de los instantes que, sumados, solemos llamar nuestra vida...."

De: Jorge Bucay en "Cuentos para Pensar"

Dedicatoria

A mis padres (Elisa y Juan), porque gracias a su cariño, guía y apoyo he llegado a realizar uno de los anhelos más grandes de mi vida, fruto de la inmensa confianza que en mi han depositado y con la cual he logrado terminar mis estudios profesionales que constituyen el legado más grande que pudiera recibir y por lo cual les viviré eternamente agradecida, especialmente dedicada a ti, mami por regalarme cada uno de tus días y ser la mejor madre del mundo, los quiero.

Agradecimientos

A mis padres: Gracias por la oportunidad de existir, por su sacrificio en algún tiempo incomprendido, por su ejemplo de superación incasable, por su comprensión y confianza, por su amor y amistad incondicional, porque sin su apoyo no hubiera sido posible la culminación de mi carrera profesional.

A mi abuelita Raquel: Sabiendo que jamás existirá una forma de agradecer una vida de lucha, sacrificio y esfuerzo constantes.

A mi familia: En reconocimiento a todo el apoyo brindado a través de mis estudios y con la promesa de seguir siempre adelante, gracias por existir.

A mi novio: Por enseñarme que las cosas no cambian, cambiamos nosotros, por desgastarse los pies siguiendo mis pasos, por soportar mis majaderías.

A mi tutor: Por ser el más exigente y obligarme continuamente a dar lo mejor de mí.

A mi Jefe de Proyecto (Yoandry): Por preocuparse por mí y estar pendiente del desarrollo de mi tesis, de todo corazón gracias.

A mi tribunal: Por darme aliento y confianza.

A todas aquellas personas que constantemente atosigué con tantas dudas, preguntas...y que siempre me atendieron: David (mi tutor), Alberto, Jorge, Figura, Pupo y muy especialmente a Inda, y Michel.

Agradecimientos

A toda la gente de mi grupo preferido, el 9105 por los momentos inolvidables que juntos pasamos.

A mis amigas las de verdad, gracias por todos los momentos compartidos: los alegres, los tristes y los de dolor, gracias por estar ahí para mí, siempre que lo he necesitado, por soportar mis pesadeces y ser fuertes en esta despedida que al graduarnos se vuelve inevitable.

Agradezco a todas aquellas personas que casi sin conocerme me pararon un día en la calle o el docente y me dijeron ¿y la tesis qué, sale o no sale?, por su preocupación muchas gracias.

DECLARACIÓN DE AUTORÍA

Declaro que soy el único autor de este trabajo y autorizo al Departamento de GeoInformática de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los ____ días del mes de _____ del año __2011_____.

Elisa Cabrera Batista **David Silva Barrera**

_____ _____

Resumen

Los sistemas de información geográfica constituyen una ciencia relativamente joven, lograr perfeccionarlos y adaptarlos a las más disímiles necesidades es una tarea de todos los sectores de la sociedad, debido al elevado número de servicios que a la misma prestan.

Con la presente investigación se pretende incorporar una nueva funcionalidad a la aplicación Quantum Gis, lo que beneficiará al polo productivo de GeoInformática permitiéndole acortar la brecha existente entre las aplicaciones SIG web y de escritorio de la plataforma GeneSIG, en el manejo de información cartográfica especificada en archivos Mapfile.

La explotación de la aplicación Quantum Gis luego de su personalización traerá como principal ventaja permitir la edición de la cartografía en una aplicación de escritorio que brinde mayor calidad y rapidez al proceso de edición de los datos geoespaciales contenidos en archivos mapfile.

Palabras claves: aplicación, compilador, representación, importar, mapfile, sistema de información geográfico, SIG, QGis.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Con el desarrollo de Internet y las redes de comunicación ha comenzado también el perfeccionamiento de *"la naciente ciencia de la información geográfica"* (Sendra, 1999), mostrando como principal exponente al Sistema de Información Geográfica (SIG).

"*Los SIG son el producto de la evolución, tanto de la Geografía como de la ciencia de la Informática. Dentro de la Geografía, los SIG representan una revolución en sí mismos cuando se incorpora, a la Cartografía, el diseño de los mapas digitales.*" (Barbosa, 2007)

"*En la cartografía se logra representar el mundo real mediante una proyección vertical sobre un plano bidimensional, simplificando las diferentes entidades para constituir lo que conocemos en la actualidad como mapa. Sin embargo con la aparición de los primeros SIG estas proyecciones del mundo real se comienzan a almacenar como datos ráster o vectoriales en grandes bases de datos georreferenciadas para luego ser consultadas, editadas y analizadas permitiendo la digitalización de los mapas.* " (Torres, y otros, 2009)

Las herramientas SIG fortalecieron inmensamente a la Cartografía al tributarle a los mapas vitalidad, calidad y precisión, la visualización en 3D y 4D[1] que proporcionan modelos exactos de la realidad, el dinamismo y la alta resolución de las imágenes cartográficas.

La contaminación, superpoblación, desastres naturales entre otros, son problemas fundamentales que enfrenta la población mundial en la actualidad y cada uno de ellos posee una dimensión geográfica crítica. La solución a estos problemas frecuentemente requiere el acceso a diferentes tipos de información que sólo puede ser relacionada por geografía y es precisamente la tecnología SIG la que permite almacenar, manipular y desplegar información usando geografía. Comprender el funcionamiento de los sistemas de información geográfica se hace necesario para entender los" por qué" de las soluciones a los diversos problemas.

[1] Se refiere a la cuarta dimensión (tiempo) además de poseer ancho, largo y profundidad se desarrolla en tiempo real.

Es precisamente la versatilidad de aplicaciones, lo que ha provocado que el manejo de la tecnología SIG haya sido asimilado por universidades, gobiernos, empresas e instituciones que la han aplicado en todos los sectores de la sociedad, hasta el punto de convertirla en una industria. El acelerado desarrollo que impulsa las crecientes oportunidades de los SIG respalda que cada día los profesionales estén más conscientes de las ventajas de pensar y trabajar desde una perspectiva geográfica.

Cuba, impulsada por la amplia gama de posibilidades que el uso de la tecnología SIG proporciona y en su afán de acrecentar el desarrollo científico-técnico e industrial del país, se ve inmersa en una estrategia para el desarrollo de aplicaciones informáticas, capaces de brindar servicios basados en la explotación de las bondades de los SIG. De acuerdo a las características de la isla, país socialista donde las obras sociales tienen prioridad, se vislumbra que los principales sectores a los que estarán destinados estos software serán:

- *Infraestructura:* Propuestos para la administración de redes de electricidad, agua, teléfono, rutas, también muy usados en trabajos de ingeniería, inventarios, planificación de redes, gestión de mantenimiento, entre otros.
- *Medio ambiente:* Estas aplicaciones serán realizadas específicamente para instituciones medioambientales, facilitando la estimación del impacto ambiental en la ejecución de proyectos, que permitan el análisis en tiempo real de la concentración de contaminantes, a fin de tomar las precauciones y medidas necesarias. Desempeñando un papel fundamental en trabajos tales como reforestación, explotaciones agrícolas, estudios de representatividad, caracterización de ecosistemas, estudios de fragmentación, deforestación, estudios de especies, etc.
- *Equipamiento social:* Calificados para la gestión de servicios de impacto social, tales como servicios sanitarios, centros escolares, hospitales, centros deportivos, culturales, lugares de concentración en casos de emergencias, centros de recreo, entre otros.

Como muestra del creciente interés del país en aplicar esta joven tecnología, se puede referenciar en la Universidad de las Ciencias Informáticas (UCI) el departamento de

GeoInformática, especializado en la gestión de proyectos que tienen como principal objetivo la creación de software destinado al manejo de información geográfica.

La UCI con el departamento de GeoInformática, pretende poner en el mercado una infraestructura que garantice la representación de datos espaciales en diferentes escenarios, en esa dirección se logró una solución para la web, que permite realizar funcionalidades básicas como agregar o quitar capas, localizar objetivos, exportar mapas completos o en fragmentos con extensión pdf, entre otras. Esta aplicación SIG para la web utiliza como motor de representación gráfica a la herramienta MapServer.

MapServer para su funcionamiento demanda un conjunto de recursos entre los que sobresale de manera excepcional el archivo MapFile, fichero de configuración donde se encuentra especificadas la estructura y composición de los datos cartográficos que serán representados.

Se pretende desarrollar una solución para plataforma de escritorio, que garantice funcionalidades que no son alcanzables de manera razonable en una aplicación SIG web, como constituye la edición de la cartografía. Se proyecta tomar de base al software Quantum Gis (QGIS), aplicación SIG de código abierto que cuenta con una amplia comunidad de desarrollo y herramienta fundamental del actual proyecto SIG-Desktop, que está encauzado precisamente en la personalización de esta aplicación. Entre las nuevas funcionalidades que requiere la personalización del QGIS se encuentra la representación de la información geoespacial descrita en archivos MapFile, para a partir de un origen de datos común lograr la integración de ambas aplicaciones.

De la situación anterior se identificó el siguiente **problema a resolver**: Quantum GIS no representa información cartográfica especificada en archivos MapFile.

La investigación se enmarca en el **objeto de estudio** sistemas de información geográfica que representen la estructura y composición de archivos MapFile definiendo como **campo de acción** el procesamiento de archivos MapFile a través de la herramienta Quantum GIS.

Para dar solución al problema planteado se define como **objetivo general** desarrollar nuevas funcionalidades que permitan a Quantum GIS importar un archivo MapFile.

Como **hipótesis** se tiene que el desarrollo de nuevas funcionalidades permitirá a Quantum GIS importar un archivo MapFile, lo que proveerá un punto de integración transparente al usuario entre las aplicaciones Web y de Escritorio de la plataforma GeneSIG.

Para el logro de este objetivo se desarrollaran las siguientes **tareas de la investigación**:

1. Caracterizar los referentes teóricos y prácticos que describen la representación de datos cartográficos especificados en archivos MapFile.
2. Realizar el análisis del negocio.
3. Desarrollar la propuesta de solución.
4. Desarrollar los Casos de Prueba que certifiquen la veracidad de los algoritmos empleados para un conjunto cerrado de archivo MapFile de entrada.

El éxito de toda investigación científica está en la solución del problema científico, en alcanzar los objetivos y en la comprobación de la hipótesis y esto depende del acierto que se tenga en la selección del método, los procedimientos y técnicas de investigación. Podemos definir el método como el camino, la vía, la estructura del Proceso de la Investigación Científica; es el sistema de procedimientos; la forma de estructuración de la actividad para transformar el objeto, para resolver el problema, para lograr los objetivos. (Zayas, 1995)

Para la solución de estas tareas se explotaron diferentes **métodos de la investigación:**

Los métodos teóricos cumplen una función epistemológica importante, ya que posibilitan la interpretación conceptual de los datos empíricos encontrados. (Zayas, 1995)

Se selecciona:

- El método Analítico-Sintético, del que Zayas plantea: "*Estas operaciones no existen independientemente una de otra: el análisis de un objeto se realiza a partir de la relación que existe entre los elementos que conforman dicho objeto como un todo; y a su vez, la síntesis se produce sobre la base de los*

resultados previos del análisis." Escogido precisamente con el objetivo de analizar y aumentar los conocimientos entorno al objeto de estudio a partir de consultar la bibliografía científica correspondiente, para después, haciendo uso de la síntesis, lograr resumir y exponer los resultados obtenidos del análisis.

- *"El estudio de la historia del objeto en toda su diversidad con sus zigzags y cualidades, ha de conducir a la comprensión de su lógica, de sus leyes de desarrollo internas y su causalidad."* (Zayas, 1995) Se utiliza el Método Histórico-Lógico**,** empleado para estudiar la trayectoria histórica y evolución de los productos de software existentes para la representación cartográfica a través de ficheros MapFile.

- ***"****La observación científica como método consiste en la percepción directa del objeto de investigación.****"*** (Zayas, 1995)Se elige el Método Empirico: Observación, con el objetivo de obtener un registro visual de toda la información referente a los procesos de análisis y representación cartográfica desarrollado en el departamento de GeoInformática.

El presente trabajo consta de 4 capítulos que están divididos en epígrafes y subepígrafes.

Capítulo 1. Fundamentación Teórica: Se analizan conceptos relacionados con la investigación, estableciendo de este modo el inicio de la misma. Se define un marco de trabajo delimitado por el objeto de estudio y la situación problemática, donde se examinan las posibles vias de solución del problema a resolver.

Capítulo 2.Tendencias y Tecnologías: Se especifican las características de las tecnologías seleccionadas para la personalización de la aplicación QGIS; así como la metodología de desarrollo de software, lenguajes de programación, arquitectura de software y herramientas auxiliares para la construcción del compilador durante la fase de implementación.

Capítulo 3. Descripción de la Solución Propuesta: En este capítulo se muestra la propuesta de solución que incluye el modelo de dominio, la identificación de los

requistios funcionales, diagramas de clase del dominio y diagramas de casos de uso del sistema, así como la descripción textual correspondiente a los mismos. Se realiza el diseño de la propuesta de solución con el objetivo de sentar las bases para la cercana fase de implementación.

Capítulo 4. Construcción de la solución propuesta: Se detalla todo el proceso de construcción de la propuesta de solución. Se realizan las pruebas necesarias para verificar los resultados alcanzados.

Capítulo 1. Fundamentación Teórica

1.1 Introducción

Este capítulo abarca todo el análisis de los temas relacionados con los Sistemas de Información Geográfica y la estructura de los archivos mapfile. Se realiza también un resumen de los elementos esenciales del proceso de compilación así como del estado del arte de las tecnologías SIG, estableciendo una base teórica para fundamentar las decisiones tomadas.

1.2 Sistemas de Información geográfica

Son muchos los autores que han querido dar su definición acerca de los denominados Sistemas de Información Geográfica o SIG:

"*Sistema de Información diseñado para trabajar con datos georreferenciados mediante coordenadas espaciales o geográficas. En otras palabras, un SIG es a la vez una base de datos con funcionalidades específicas para datos referenciados espacialmente y un conjunto de operaciones para trabajar con los datos*."(STAR Y ESTES, 1990)

"*Un sistema de hardware, software y procedimientos diseñado para realizar la captura, almacenamiento, manipulación, análisis, modelización y presentación de datos referenciados espacialmente para la resolución de problemas complejos de planificación y gestión*." NCGIA (National Centre of Geographic Information and Analysis ,1990)

"*Un sistema computarizado compuesto por hardware, software, datos y aplicaciones que es usado para registrar digitalmente, editar, modelar y analizar datos espaciales, y presentarlos en forma alfanumérica y gráfica*". (Hewlett Packard ,1993)

Luego de valorar las diversas citas, se pudo concluir definiendo a los SIG como una integración organizada de hardware, software y datos geográficos, diseñados para capturar, integrar, almacenar, editar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada.

1.3 Clasificación de los SIG según el modelo de datos

En dependencia del uso que se le quiera dar a los Sistemas de Información Geográfica pueden ser clasificados atendiendo al modelo de datos con que procesa la

información, de esta forma se pueden encontrar tanto SIG que utilizan modelos Vectoriales, Ráster ó ambos.

- *Datos Vectoriales:* son aquellos Sistemas de Información Geográfica que para la descripción de los objetos geográficos utilizan elementos compuestos por puntos, líneas y polígonos definidos por coordenadas relativas a algún sistema cartográfico. (Figura 1)

- *Datos Ráster:* Los Sistemas de Información Ráster basan su funcionalidad en una abstracción de la realidad. Cada superficie a representar se divide en filas y columnas, formando una malla o rejilla regular. Cada celda ha de ser rectangular, aunque no necesariamente cuadrada.(Figura 1)

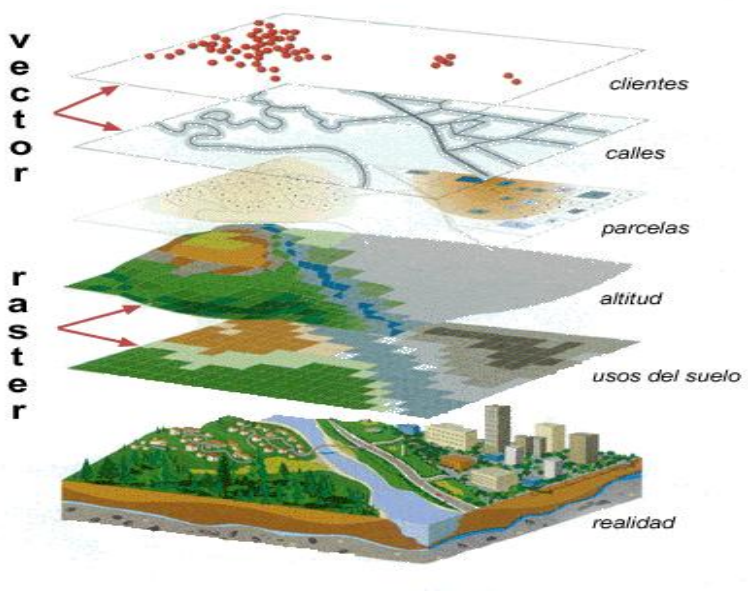


Figura 1.Clasificación de capas

1.4 Funcionamiento de los SIG

Los SIG generalmente permiten realizar seis tareas fundamentales que apoyan la comprensión de su funcionamiento:

- *Ingreso:* Cuando se va a introducir los datos estos deben ser convertidos internamente a un formato digital adecuado.

- *Manipulación:* Transformaciones que se le realizan a los datos como cambios de escala, proyección y generalización.
- *Manejo/Administración:* Para facilitar el manejo de la información almacenada se requiere de un sistema de manejo de bases de dato (SMBD).
- *Consulta:* Se deben responder las preguntas realizadas por los usuarios al sistema.
- *Análisis:* Para desarrollar el análisis se utilizan dos procesos el de proximidad y el de superposición.
- *Visualización:* Se muestra como un mapa o gráfico.

1.5 Clasificación de los SIG de acuerdo sus funcionalidades

En dependencia de los escenarios en que se ejecuten las aplicaciones informáticas estas pueden clasificarse en aplicaciones de escritorio o web. La diferencia viene dada principalmente porque una aplicación web es un conjunto de páginas web (estáticas o dinámicas), que interactúan unas con otras y con disímiles recursos de un servidor. Para su funcionamiento requieren ser cargadas por otras aplicaciones como un navegador a través del cual los usuarios realizan peticiones que son respondidas a través de consultas realizadas a la base de datos. Mientras las aplicaciones de escritorio se ejecutan recurriendo a su propia dirección de memoria (física o virtual) en una computadora, de modo local.

Los SIG constituyen aplicaciones informáticas, por tanto no están exentos de esta clasificación, teniendo en cuenta el tipo de SIG web o de escritorio con que se trabaje, la principal diferencia de acuerdo a las funcionalidades que ofrecen radica, en que los SIG para web no son capaces de editar la cartografía con la elevada calidad que puede hacerlo un SIG de escritorio.

1.6 Herramientas SIG

En la actualidad se pueden encontrar numerosos exponentes de SIG de escritorio que gozan de gran prestigio a nivel mundial, afirmación respaldada por la gran cantidad de usuarios que los utilizan producto a las cuantiosas funcionalidades que implementan y la variedad de formatos sobre los que trabajan.

Se destacan MapInfo, caracterizado por su perfecta conectividad a las bases de datos (Pitney Bowes Software Inc., 2010); GvSig, especializado en la implementación de servicios OGC (Open Geospatial Consortium) (Fundación Plone , 2010-2011); Grass,

ampliamente reconocido por su integración modular, compuesto por más de 350 módulos y herramientas para la ejecución de tareas (GRASS Development Team, 1999-2011); ERDAS Imagine conocido como el software geoespacial de creación estándar de la industria debido a su orientación centrada en el procesamiento de imágenes y la teledetección (ERDAS Inc., 2011); ARCGIS Desktop, probado como un conjunto de aplicaciones integradas de ArcMap, ArcCatalog y ArcToolbox que conjuntamente permiten desarrollar cualquier labor SIG (ArcGis Community, 2011) y Quantum Gis, considerado como uno de los SIG de escritorio de código abierto más completos. (Open Source Geospatial Foundation, 2011)

1.6.1 Software Quantum Gis

El software Quantum Gis es un Sistema de información Geográfica de Código abierto, multiplataforma, desarrollado en QT Toolkit y C++. Se publica bajo Licencia Pública General (GNU General Public License). Ofrece muchas características SIG comunes, proporcionadas por las funciones del núcleo y los complementos.

"QGIS puede ver y superponer datos vectoriales y ráster en diferentes formatos y proyecciones sin conversión a un formato interno o común." (OSGeo, 2004-2009)

Los formatos admitidos incluyen Tablas de PostgreSQL con capacidad espacial, archivos en formatos ráster e imágenes admitidas por la biblioteca GDAL (Geospatial Data Abstraction Library), datos ráster y vectoriales de GRASS de bases de datos entre otros. Es capaz de diseñar mapas y explorar datos espaciales de forma interactiva con una interfaz amigable. Permite además crear, editar, administrar y exportar mapas vectoriales en varios formatos.

"Ofrece herramientas de digitalización para formatos admitidos por OGR y capas vectoriales de GRASS, crear y editar archivos shape y capas vectoriales de GRASS, Geocodificar imágenes con el complemento Georreferenciador, Herramientas GPS para importar y exportar formato GPX y convertir otros formatos GPS a GPX, Crear capas PostGIS a partir de archivos shape con el complemento SPIT, Guardar capturas de pantalla como imágenes georreferenciados." (OSGeo, 2004-2009)

"Permite realizar análisis de datos espaciales de PostgreSQL/PostGIS y de otros formatos admitidos por OGR usando el complemento de Python fTools. QGIS actualmente ofrece herramientas de análisis vectorial, muestreo, geoprocesamiento,

geometría y administración de bases de datos. También puede usar las herramientas de GRASS integradas, que incluyen la funcionalidad completa de GRASS de más de 350 módulos. QGIS también se puede usar como cliente WMS o WFS y como servidor WMS además de permitir exportar datos a un archivo Mapfile y publicarlos en Internet usando un servidor web con UMN MapServer instalado." (OSGeo, 2004-2009)

Es precisamente esta última característica la que hace que se profundice en su composición y funcionamiento. Son muy pocos los Sistemas de Información Geográfica que centran su atención en la integración de las aplicaciones web y de escritorio, sin embargo el Quantum Gis ya tiene garantizada la exportación del archivo Mapfile considerado el corazón del reconocido servidor de mapas MapServer, que constituye el recurso básico del SIG para la Web.

1.6.2 Proyecto UMN MapServer

El proyecto UMN MapServer fue ideado inicialmente como unos scripts para la plataforma ArcGIS/ArcInfo, en la que generaban de forma dinámica impresiones de cartografía para publicar en entornos web. Es un proyecto de software libre, capaz de comunicarse con una gran variedad de formatos tanto ráster como vectoriales y servirlos mediante un módulo CGI (Common Gateway Interface). (Regents of the University of Minnesota, 2011)

MapServer es uno de los servidores de mapas más utilizados debido a que cuenta con una amplia comunidad de desarrollo y soporte. Este es definido como un ambiente de desarrollo de código libre para construir aplicaciones con datos espaciales y mostrarlas a través de Internet. Su funcionalidad básica radica en interpretar datos geográficos (mapas, imágenes y datos vectoriales) para mostrarlo en la Web.

Una aplicación de manejo, análisis y representación de información geoespacial con UMN MapServer necesita los siguientes recursos:

- El Servidor UMN MapServer.
- Un Servidor HTTP (Apache/Internet Information Server).
- Un archivo Mapfile.
- Un archivo Plantilla (Template).

  (Cubillo, 2000)

1.7 El archivo Mapfile

El Mapfile es el fichero principal de configuración del UMN MapServer, es analizado en cada interacción del usuario con el servidor, en él se definen varias características del servidor de mapas. Incluye una serie de parámetros que especifican las capas disponibles en el servicio, el estilo con que se representarán, su simbología, formato en que se generará la imagen, el sistema de referencia, etc.

La estructura del archivo Mapfile está distribuida por secciones u objetos, donde cada sección inicia con el nombre de la sección y termina con la palabra END." *El contenido de las secciones consiste en la definición de determinados parámetros del tipo atributo – valor."* (Ballari, 2001)

El orden de los parámetros no es sensitivo. Los colores son manejados mediante los tres canales R G B (rojo – verde –azul).

La siguiente tabla muestra las secciones y funciones principales de cada objeto dentro del archivo Mapfile.

No.	Objeto	CARACTERÍSTICAS
1	**MAP**	Es la sección principal del Mapfile, contiene dentro al resto de las secciones.
2	**PROJECTION**	Se encarga de definir la proyección de los mapas a representar. Es necesario especificar dos objetos PROJECTION uno en el objeto MAP para la imagen de salida y otro para cada capa en el objeto LAYER.
3	**WEB**	Define como operará la interfaz web anidando al objeto METADATA.
4	**METADATA**	Deben definirse dos objetos METADATA, uno en el objeto MAP donde contendrá los datos de servicio general y otro en cada objeto LAYER con datos específicos de cada capa de información.
5	**LAYER**	Se debe definir un objeto LAYER por cada capa de información que presente el servicio.
6	**CLASS**	Define las clases temáticas de las capas, cada capa debe tener al menos una clase.

7	**LABEL**	Permite definir una etiqueta, con la cual es posible colocar la toponimia[2] u otro tipo de anotación en el mapa, a partir de datos alfanuméricos.
8	**LEGEND**	Permite generar la simbología de forma automática.
9	**SCALEBAR**	Define como se construirá la escala gráfica.

Tabla 1.Objetos del Archivo Mapfile

Se realiza mayor énfasis en la composición del Objeto Map debido a que constituye la sección principal del Mapfile (Figura 2.) y los atributos contenidos en él, son de gran interés para la posterior generación del fichero con extensión .qgs. Algunos de estos atributos son de obligatoria inclusión mientras otros son opcionales o tienen un valor asignado por defecto, el número de objetos y parámetros está en dependencia de lo que se desee representar.

Para un mayor entendimiento de la jerarquía entre objetos referirse a la Figura 2.

[2] Estudio del origen y significación de los nombres propios de un lugar.

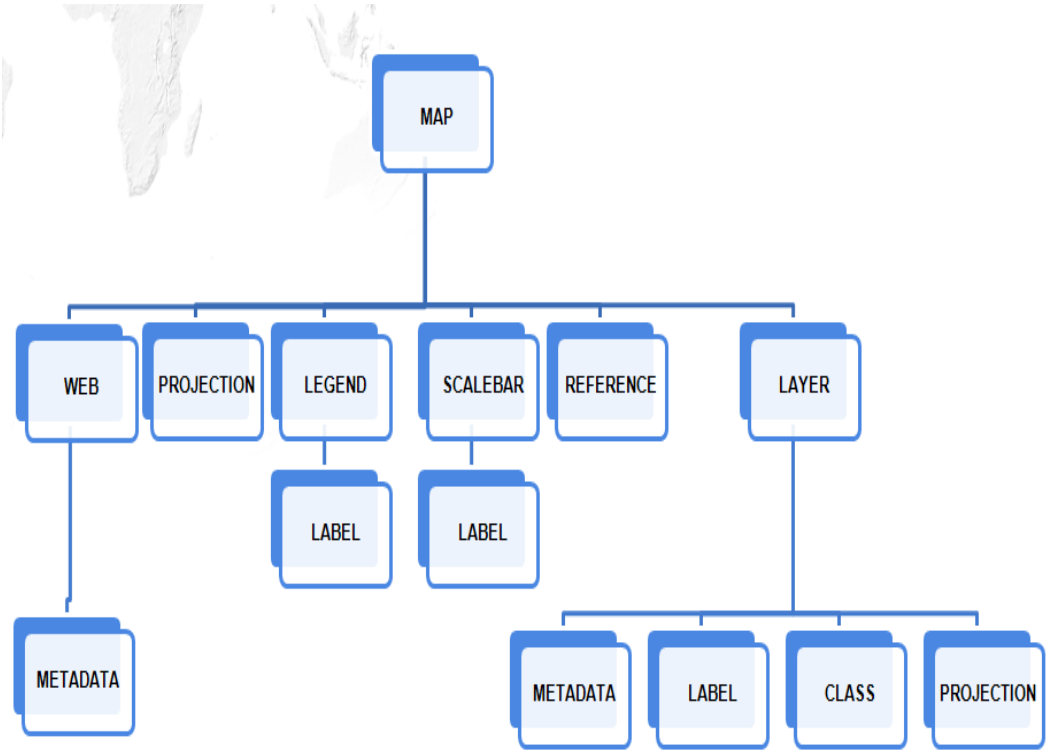


Figura 2.Jerarquía de objetos de un fichero Mapfile.

Un archivo .map está constituido por secciones y estas a su vez están compuestas por atributos de la forma llave – valor si se desea extraer cada uno de los elementos que lo componen; se requiere fragmentar los textos incluidos en él y convertirlos en frases sintácticamente válidas para, luego lograr representar la información cartográfica descrita dentro del fichero .map. Las herramientas informáticas que permiten desarrollar este tipo de acciones suelen recibir por nombre compiladores.

1.8 Proceso de Compilación

"*El desarrollo de un compilador introduce al informático en un nuevo mundo, en el que el control sobre el ordenador es absoluto y le lleva al paroxismo de la omnipotencia sobre la máquina."* (Gálvez Rojas, y otros, 2005)

Similar al modo en que las personas se comunican a través del lenguaje, las computadoras lo realizan a través de bits[3] y registros. Sin embargo, lograr el proceso

[3] Un **bit** es la unidad de información más pequeña con la que puede trabajar una computadora. Se representa, con los dígitos binarios, *cero* (0) y *uno* (1).

de comunicación hombre-máquina no es tan simple y requiere también del uso de un lenguaje, oficialmente conocido como lenguaje de programación.

Un lenguaje de programación puede ser definido como:

- Notación formal para describir algoritmos y funciones que serán ejecutados por una computadora.
- Lenguaje para comunicar instrucciones a una computadora.
- Convención para escribir descripciones que puedan ser evaluadas. (http://www.scribd.com)

Haciendo uso de los lenguajes de programación el programador es capaz de realizar programas que controlen el comportamiento lógico de la computadora, especificando sobre qué tipos de datos operará la computadora, la forma en que estos serán almacenados y transmitidos o tareas específicas como algoritmos para el tratamiento de un problema particular. (Avella, 2009)

Los programas desarrollados en los diferentes lenguajes de programación son comúnmente conocidos como programas fuente y para ser comprendidos por la computadora debe realizárseles un proceso de conversión del programa fuente al código de máquina. El programa capaz de realizar esa conversión se conoce como compilador o traductor y al proceso de conversión se le identifica como compilación.

Un traductor es un programa que toma como entrada un programa escrito en un lenguaje de programación (lenguaje fuente) y produce como salida un programa en otro lenguaje (lenguaje objeto). El traductor se escribe en un lenguaje denominado lenguaje de implementación.

Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje máquina). (Narcizo, 2004)

Un intérprete por su parte es un programa que toma el código fuente, lo analiza y a diferencia de los compiladores lo ejecuta directamente, sin generar un lenguaje objeto.

1.8.1 Definiciones asociadas al proceso de compilación

Alfabeto: Conjunto no vacío y finito de símbolos. Σ= {a, b, c,..., z}, a $\in\Sigma$

Palabra o Cadena: Secuencia finita de símbolos definida sobre un alfabeto Σ.Su longitud está dada por la cantidad de símbolos que la componen.

Lenguaje: Un lenguaje (L) sobre un alfabeto Σ es un conjunto de palabras o cadenas que obedece ciertas reglas de formación sobre dicho alfabeto.

Gramática: La gramática es un ente o modelo matemático que permite especificar un lenguaje, es decir, es el conjunto de reglas capaces de generar todas las posibilidades combinatorias de ese lenguaje, y sólo las de dicho lenguaje, ya sea éste un lenguaje formal o un lenguaje natural.

Definición: Una gramática es un cuádruplo G = {N, Σ, P, S} donde:
N: Conjunto de símbolos no terminales.
Σ: Conjunto de símbolos terminales.
P: Conjunto de Reglas de Producción.
S: Axioma o símbolo distinguido (S N) N: Conjunto de símbolos no terminales.

Autómata: Un autómata finito o máquina de estado finito es una herramienta abstracta que se utiliza para reconocer un determinado lenguaje regular. Es un modelo matemático de un sistema que recibe una cadena constituida por caracteres de cierto alfabeto y determina si esa cadena pertenece al lenguaje que el autómata reconoce.

Un autómata está definido por una 5-tupla N = (S, Σ, δ, S_0, F).
Donde:
S: Es un conjunto finito de estados.
Σ: Alfabeto del lenguaje. Conjunto de símbolos de entrada del autómata.
δ: Función de transición definida como δ: S x Σ∪ {∈ } P(S).
S_0: Es el estado inicial del autómata S_0 ∈S.
F: Es el conjunto de estados finales o de aceptación del autómata. F ∈S.

Vocabulario léxico: Conjunto de palabras que forman parte de un lenguaje específico.

Sintaxis: Conjunto de reglas necesarias para construir frases correctas en un lenguaje. (Alconchel, 2004)

1.9 Fases de un Compilador

Básicamente el proceso de compilación puede ser dividido en dos etapas una de análisis que comprende:

1. Análisis lexicológico (scanner)
2. Análisis sintáctico (parser)
3. Análisis semántico

Y otra de síntesis encargada de construir el programa objeto a partir de las estructuras generadas a partir del análisis.

1. Generación de código intermedio
2. Optimización de código intermedio
3. Generación de código ejecutable u objeto
4. Tabla de símbolos
5. Gestión de errores

La figura 3 ilustra en gran medida cómo se organiza el proceso de compilación.

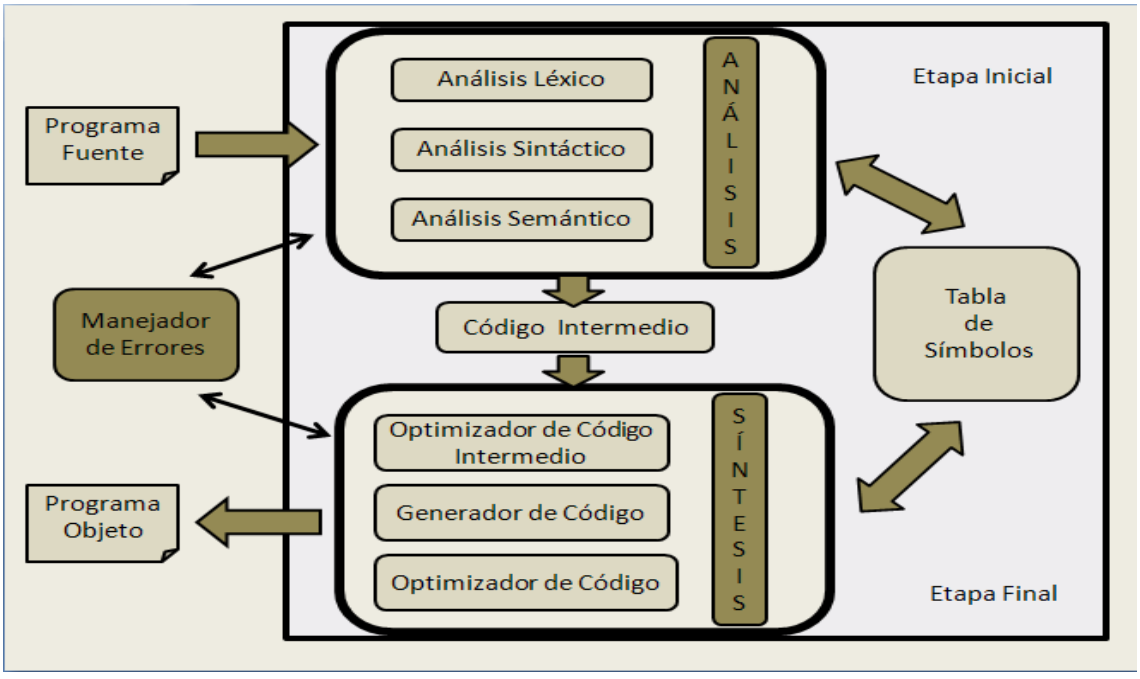


Figura 3.Proceso de Compilación

1.9.1 Análisis lexicológico (scanner)

La entrada del compilador es un programa escrito en código de programación, ese código es una secuencia de símbolos pertenecientes al alfabeto de ese lenguaje. La fase de análisis léxico es la encargada de tomar y agrupar estos símbolos en elementos sintácticos denominados tokens o lexemas, por tanto, el análisis léxico tiene lugar al nivel de los caracteres, éstos son analizados secuencialmente y comparados con patrones que representen unidades sintácticas válidas (tokens).

Los tokens son los elementos básicos sobre los que se ejecuta la traducción de un lexema, de ahí que un scanner se defina como un traductor cuya entrada es una cadena de símbolos (programa fuente) y cuya salida es una secuencia de estructuras lexicológicas o tokens.(Joven Club de Computación y Electrónica, 2009-2010)

Cuando un token es identificado, el analizador léxico entrega el mismo al analizador sintáctico acompañado de una serie de información dependiente de las necesidades del traductor.

Ejemplo 1.Funcionamiento del analizador léxico

Se tienen 3 variables declaradas como reales (real a, b, c) y la expresión: a = b – c*2;

- Se identifican a, b y c como tokens del tipo identificador (id).
- Se identifican "=","-","*" como tokens del tipo operador (Op igual, Op resta, Op mul)
- Se identifica el 2 como un token de tipo constante (const).
- Se identifica ";"como delimitador.

La salida del scanner sería:

Id1 = id2 – id3 *const;

<id, 1> <Op igual, => <id, 2> < Op resta, -><id,3> <Op mul,* > < const, 4> <delimitador, ;>

1.9.2 Análisis sintáctico (parser)

Los lenguajes de programación obedecen reglas que describen la estructura bien organizada que estos aceptan. Analizando estas reglas y sus regularidades se pueden

definir gramáticas que las generen, haciendo más simple la generación del código y la detección de los errores.

El análisis sintáctico es el proceso en el cual se examina la secuencia de tokens resultante del análisis léxico, para determinar si el orden de esa secuencia es correcto. Inmediatamente dichas secuencias son agrupadas en clases sintácticas (no terminales de la gramática), respondiendo a ciertas convenciones estructurales (reglas) de la definición sintáctica del lenguaje; de ahí que el análisis sintáctico tenga lugar en el nivel de la sentencia, haciendo este proceso más complejo que el desarrollado por el analizador léxico.

Ejemplo 2. Funcionamiento del analizador sintáctico

Dada la pequeña gramática que define a expresión.

exp − > exp - exp

exp − > exp * exp

exp − > id

exp − > const

Si se analizara la expresión a-b*2 la respuesta del analizador sintáctico sería que la expresión es sintácticamente correcta, de acuerdo a la especificación del lenguaje determinado.

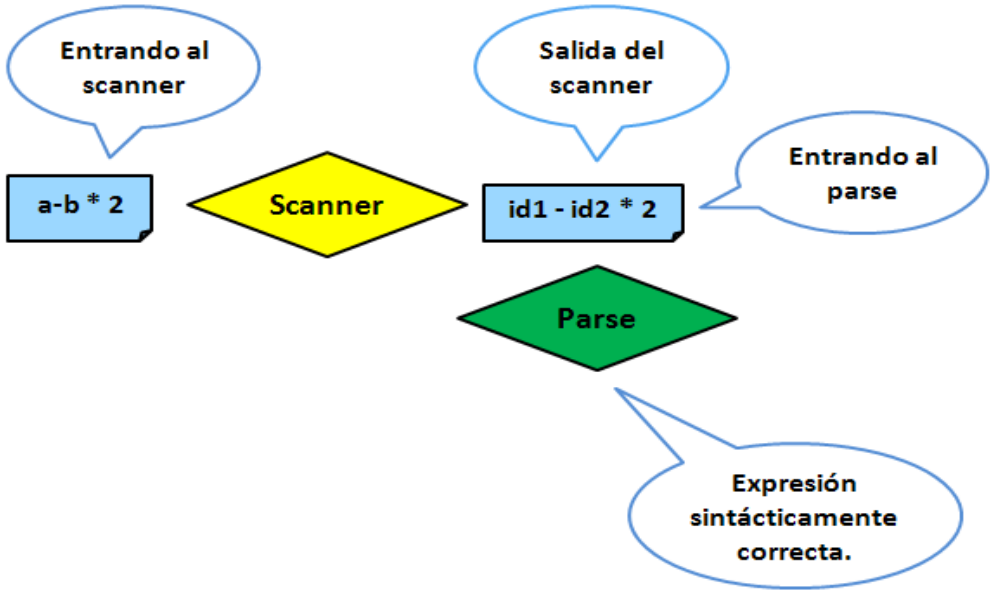


Figura 4.Salida del parser

Es necesario tener en cuenta que al realizar las operaciones de esta expresión, la multiplicación de c*2 debe realizarse antes de la operación resta. Para garantizar la precedencia de operadores se construye un árbol de sintaxis abstracta también conocido como árbol sintáctico, que permitirá conocer la estructura sintáctica de la secuencia de tokens.

Siguiendo el orden de estas operaciones se puede apreciar el proceso de creación del árbol de sintaxis abstracta.

1. id3 se multiplique con 2

2. el resultado de 1 se le resta a id2.

3. el resultado de 2 se almacene en id1.

En la figura se puede observar un recorrido en postorden del árbol.

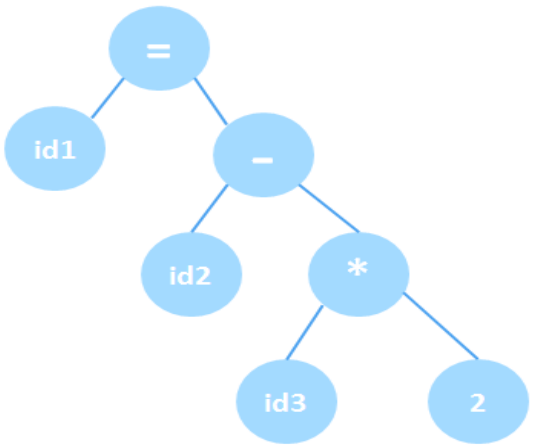


Figura 5. Árbol de Sintaxis Abstracta

1.9.3 Análisis Semántico

El analizador semántico recibe un árbol de sintaxis abstracta con la información de los tokens en la instrucción que está analizando, como resultado de la fase de análisis sintáctico. La fase del analizador semántico detecta errores relacionados con la validez del programa que vienen dadas por interdependencias entre las distintas partes de un programa. Típico de esta fase es la comprobación de tipos de datos. (Narcizo, 2004)

Ejemplo 3.Chequeo de datos

Al realizar el análisis semántico en el chequeo de tipos de datos se debe realizar una transformación en el árbol sintáctico. Se debe convertir 2 que es una constante entera a real que es el tipo de dato del resto de los identificadores.

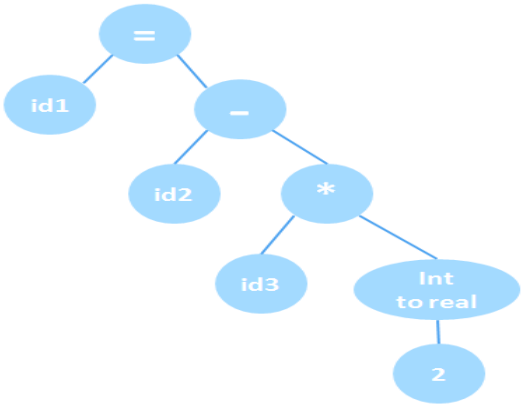


Figura 6.Árbol de Sintaxis Abstracta

1.9.4 Generación de código intermedio

El código intermedio se caracteriza por ser fácil de producir y de traducir al programa objeto. Viene siendo una representación explícita intermedia del código fuente, no es un lenguaje de programación de una máquina real sino que corresponde a una máquina abstracta. Su objetivo principal es reducir la cantidad de programas necesarios en la construcción de traductores facilitando la movilidad del mismo de una máquina a otra.

Para la traducción del programa fuente se utiliza el árbol creado en el análisis semántico logrando de esta forma la interpretación de las acciones asociadas a las sentencias. Existen numeroso métodos que auxilian este proceso se puede hacer alusión a la Notación Polaca orientada a pila, la Notación de Cuádruples y también el muy utilizado Método de Traducción Sintáctica Directa que suele ser reconocido por su facilidad para lograr la generación del código.

Ejemplo 4. Método de Traducción Sintáctica Directa

Haciendo uso del método Traducción Sintáctica Directa que funciona asociando a cada nodo (n) un código intermedio C(n) y concatena el código de los descendientes de n logrando como resultado el código de n, refiérase al árbol de la figura 6.

var1	=	IntToReal(2)	
var2	id3	*	var1
var3	id2	-	var2
id1	=	var3	

Figura 7.Método de Traducción Sintáctica Directa.

1.9.5 Optimización de código intermedio

Esta fase tiene lugar debido a la necesidad de mejorar la eficiencia del código intermedio, la misma es independiente de la máquina. Los ejemplos más probables de optimizaciones suelen ser evaluaciones de expresiones constantes, el uso de las propiedades asociativas y conmutativas de algunos operadores y la reducción de expresiones comunes.

1.9.6 Generación del código objeto

La generación del código objeto es la fase final de un compilador y es el proceso en que el código intermedio es convertido en código de máquina de una máquina real si es un compilador o a otro lenguaje si fuese un traductor. La primera acción que se realiza durante esta fase es localizar la dirección de memoria de cada una de las variables que se utilizan en el programa, luego las instrucciones intermedias son traducidas a secuencias de instrucciones en código máquina que realicen operaciones similares para posteriormente asignar a los registros las variables correspondientes.

1.9.7 Tabla de símbolos

Se define como una estructura de datos donde se encuentra almacenada toda la información referente a los identificadores del programa fuente. Los elementos de la tabla se conforman del tipo identificador – atributos, el campo identificador (lexema) es introducido por el analizador léxico que se encarga de insertar los identificadores en la medida en que son reconocidos mientras el resto de los campos puede ser introducido por cualquiera de los tres analizadores que conforman el proceso de análisis.

1.9.8 Gestión de errores

Los compiladores deben ser capaces de identificar los errores y además de mostrarles a los programadores las posibles causas del error, si los programas no tuviesen errores el trabajo de los compiladores se simplificaría en gran medida. Los errores que aparecen frecuentemente en los programas se clasifican en:

- Los errores lexicológicos son detectados por el analizador léxico cuando los caracteres restantes de la entrada no forman ningún token válido para dicho lenguaje. Los más frecuentes son (escribir mal un número, un símbolo no permitido, etc.)

- Los errores sintácticos se manifiestan durante la fase de análisis sintáctico cuando se encuentran tokens que no responden a las reglas de la gramática definidas para ese lenguaje. Un ejemplo de este tipo de error puede ser expresión aritmética con doble signo de igual o con paréntesis no balanceados.

- Durante la fase de análisis semántico el compilador trata de identificar aquellas estructuras que son sintácticamente correctas pero que de acuerdo a las

operaciones involucradas son incorrectas. Un modelo clásico de este tipo de error suele ser cuando se trata de aplicar un operador a un operando incompatible.

- Los errores lógicos o de programación vienen dados por el factor humano un ejemplo muy común suelen ser los ciclos infinitos.

Un compilador no sólo debe ser capaz de identificar los errores sino también de tratarlos y recuperarse de los mismos permitiendo la detección de errores posteriores. (Universidad de las Ciencias Informáticas, 2008-2009)

1.10 Conclusiones Parciales

En este capítulo quedaron recogidos aspectos relevantes sobre los principales conceptos asociados a la investigación, fortaleciendo los conocimientos que servirán de base para el desarrollo de la misma. Los estudios realizados hasta el momento sobre la estructura del fichero Mapfile, la aplicación SIG de escritorio Qgis y el proceso de compilación, permitieron tomar la decisión, de desarrollar un compilador dentro de la herramienta Qgis para lograr de este modo, la representación de la información contenida en el archivo Mapfile.

Capítulo 2.Tendencias actuales y tecnologías a utilizar

2.1 Introducción

El presente capítulo describe las características de las tecnologías a utilizar en la personalización de la aplicación Qgis: metodologías de desarrollo de software, herramientas de modelado así como lenguajes de programación. Manifiesta una síntesis del comportamiento de estas tecnologías a nivel mundial y las ventajas inherentes a su utilización, con el objetivo de encontrar una selección del instrumental tecnológico que permita optimizar y ganar en tiempo, en el desarrollo del producto.

2.2 Lenguaje Unificado de Modelado (UML)

Lenguaje Unificado de Modelado (UML) consiste en un "lenguaje de modelado" creado para visualizar, especificar, construir, documentar, detallar y describir métodos o procesos; en esencia, los artefactos de un sistema.

El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue.

Un rasgo distintivo del UML es la absoluta independencia que muestra del lenguaje de implementación, de tal forma que los diseños logrados usando UML se pueden realizar en cualquier lenguaje orientado a objetos.

"*UML ayuda a los usuarios a entender la realidad desde un punto de vista de la tecnología y la posibilidad de que reflexione antes de invertir y gastar grandes cantidades de dinero en proyectos que no estén seguros en su desarrollo, reduciendo el costo y el tiempo empleado en la construcción de los módulos que construirán el software.*" (Booch, y otros, 2000)

Entre las ventajas que aporta este estándar de modelado se pueden encontrar:

- Mayor rigor en la especificación.
- Admite cubrir las vistas necesarias para desarrollar y luego desplegar los sistemas.

- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y a la inversa a partir del código fuente generar los modelos.
- Ampliamente utilizado por la industria del software.
- Reemplaza a decenas de notaciones empleadas por otros lenguajes.
- Modela estructuras complejas.

2.3 Metodologías de desarrollo de software

La necesidad de crear una aplicación de software, que responda a las restricciones de los estándares de software a nivel mundial es una tarea ardua y engorrosa, que dificulta grandemente el trabajo de los desarrolladores. "*Las metodologías de desarrollo de software surgen ante la necesidad de utilizar una serie de procedimientos, técnicas, herramientas y soporte documental a la hora de desarrollar un producto software*." (Carrillo Pérez, y otros, 2008). Su objetivo principal es guiar a los desarrolladores en la creación de nuevas aplicaciones de probada calidad.

Metodologías pesadas: son aquellas orientadas al control de los procesos, estableciendo rigurosamente las actividades a desarrollar, herramientas a utilizar y notaciones que se usarán.

Metodologías ágiles: están más orientadas a la interacción con el cliente y el desarrollo incremental del software, mostrando versiones parcialmente funcionales del software al cliente en intervalos cortos de tiempo, para que pueda evaluar y sugerir cambios en el producto según se va desarrollando.

2.3.1 Proceso Unificado Racional

Dentro de las metodologías pesadas sobresale el Proceso Unificado Racional (Rational Unified Process en inglés, usualmente abreviado como RUP) considerada entre las más tradicionales, centrada en la definición detallada de los procesos y tareas a realizar, herramientas a utilizar, y requiere una extensa documentación, ya que pretende prever todo de antemano. Entre las características que distinguen a RUP sobresalen:

Dirigido por casos de uso: Los casos de uso constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo del software.
Centrado en la arquitectura: Se establece una arquitectura candidata al inicio del desarrollo del sistema que funcionará como guía y se irá perfeccionando en las restantes fases de desarrollo.

Iterativo e incremental: El desarrollo iterativo garantiza la corrección de los errores en cada iteración, brindando la posibilidad de que los elementos sean integrados continuamente, lo que garantiza un producto más robusto y de mayor calidad.

Otras características a tener en cuenta:

Desarrollo basado en componentes: El sistema se va creando en la medida en que se obtienen o se desarrollen y maduren sus componentes.

Utilización de un único lenguaje de modelado: UML es adoptado como único lenguaje de modelado para el desarrollo de todos los modelos.

Proceso Integrado: Se establece una estructura que abarca los ciclos, fases, flujos de trabajo, mitigación de riesgos, control de calidad, gestión del proyecto y control de configuración; el proceso unificado establece una estructura que integra todas estas facetas.

2.4 Arquitectura de Software

David Garlan establece que "*la Arquitectura de Software(AS) constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño"*. Numerosas son las definiciones que los expertos en el tema precisan para AS sin embargo esta investigación se regirá por la que aparece redactada en el documento de IEEE Std 1471-2000:

"*La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución*." (Reynoso, y otros, 2004)

La AS es considerada una disciplina por mérito propio, es la que establece los fundamentos para que analistas, diseñadores, programadores etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. Además, el arquitecto debe llevar a cabo expeditamente y con calidad. El diseño o selección, representación, evaluación, análisis y recuperación de la misma; tiene que garantizar el desarrollo y evolución de la aplicación basada estrictamente en ella.

Por estos motivos, la AS se convierte en un punto clave en el proceso de desarrollo del software, en el que además de buscar un diseño sólido, es preciso realizar una evaluación de las decisiones tomadas durante el diseño debido al gran impacto que pueden tener en el proceso de desarrollo posterior. La evaluación de las decisiones brinda la posibilidad de saber el grado con que se han alcanzado los atributos de calidad que se persiguen y pone al descubierto tanto los puntos débiles como las potencialidades de la arquitectura propuesta en una etapa donde se pueden hacer los cambios necesarios para mitigar estas debilidades sin un gran costo, maximizando los resultados del proceso de desarrollo del software.

De acuerdo a las características de la arquitectura de software utilizada en la construcción de Quantum Gis no se concibe la idea de realizar cambios en la misma, por lo que se continuará trabajando sobre las elegidas por los desarrolladores originales que desplegaron su trabajo apoyado en una arquitectura basada en componentes y orientada a objetos.

2.4.1 Arquitectura Orientada a Objetos.

"Los componentes de un sistema encapsulan los datos y las operaciones que deben aplicarse para manejar los datos. La comunicación y coordinación entre componentes se consigue mediante el paso de mensajes". (Pressman, 2005)

"*Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos, se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. Las interfaces están separadas de las*

implementaciones. En general la distribución de objetos es transparente."(Reynoso, y otros, 2004)

Según Boosh "*La identificación de las clases y los objetos es la parte más difícil dentro de diseño orientado a objetos",* si la identificación es realizada de forma innovadora y cuidadosa sus resultados serán satisfactorios al proyectar una guía para el proceso de implemetación de clases y atributos.

2.4.2 Arquitectura Basada en Componentes.

"Un componente es una unidad de composición de aplicaciones software, que posee un conjunto de interfaces y un conjunto de requisitos, y que ha de poder ser desarrollado, adquirido, incorporado al sistema y compuesto con otros componentes de forma independiente, en tiempo y espacio" (Fuentes, et al., 2003)

"Un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. Los componentes en el sentido estilístico son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución."(Reynoso, y otros, 2004)

El Desarrollo de Software Basado en Componentes (DSBC), aboga por el principio de reutilización del software, donde la descomposición de la aplicación en componentes funcionales respalda que los componentes sean implementados de forma que se logre su manipulación sobre otros sistemas.

2.5 Herramientas Case

"*Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas en términos de tiempo y de dinero.*" (Case, 2011)

La tecnología CASE supone la automatización del desarrollo del software, contribuyendo a mejorar la calidad y la productividad en el desarrollo de sistemas en función de:

- Permitir la aplicación práctica de metodologías estructuradas, las cuales al ser realizadas con una herramienta permitan agilizar el trabajo.
- Facilitar la realización de prototipos y el desarrollo conjunto de aplicaciones.
- Simplificar el mantenimiento de los programas.
- Mejorar y estandarizar la documentación.
- Aumentar la portabilidad de las aplicaciones.
- Facilitar la reutilización de componentes de software.
- Permitir un desarrollo y un refinamiento visual de las aplicaciones, mediante la utilización de gráficos.

2.5.1 Visual Paradigm

Visual Paradigm es una herramienta CASE orientada a UML que proporciona apoyo al modelado visual, ofrece soporte al ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Sus características principales:

- Entorno de creación de diagramas para UML.
- Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
  (www.visual-paradigm.com,2011)

2.6 Lenguaje de Programación

Un lenguaje de programación es la técnica que permite al programador establecer las instrucciones que debe realizar el computador, a través de reglas sintácticas y semánticas que conforman un programa. Los lenguajes de programación son la vía de

comunicación entre el hombre y la máquina, definen sobre qué tipos de datos trabajar y cómo responder a cada una de las variadas de circunstancias.

El C++ es un lenguaje versátil, potente y general. Proviene del lenguaje C original, del cual mantiene sus ventajas en cuanto a riqueza de operadores y expresiones; ha eliminado algunas de las dificultades y limitaciones del mismo. De ahí que goce de gran prestigio entre los programadores profesionales como herramienta fundamental para el desarrollo de software.

El C++ es un lenguaje imperativo, orientado a objetos se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Mantiene una considerable potencia para programación a bajo nivel aunque se le han añadido elementos que le permiten también un estilo de programación con alto nivel de abstracción. Se le han incorporado nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones inline, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería.

2.7 Entorno Integrado de Desarrollo

Un Entorno Integrado de Desarrollo (IDE) es un entorno de programación o programa informático compuesto por un conjunto de herramientas que utilizan los programadores para generar código. Las herramientas que oficialmente componen un IDE son: un editor de código, un compilador, un depurador, un constructor de interfaz gráfica y eventualmente un sistema de control de versiones. Tiene como objetivo acortar la brecha entre el usuario y el lenguaje de programación proporcionando un marco de trabajo amigable. Un IDE puede estar realizado para soportar un sólo lenguaje de programación o varios.

2.7.1 Qt Creator

Qt Creator es un Entorno Integrado de Desarrollo o IDE bastante completo que permite el desarrollo de aplicaciones en entornos MS Windows, Mac OS y Linux. Se caracteriza por ser abierto, gratuito y muy eficiente. Proporciona herramientas para el diseño y desarrollo de aplicaciones complejas, entre sus principales características se distinguen:

- Utiliza el lenguaje de programación orientado a objetos C++.

- Se basa en Qt, una librería multiplataforma y gratuita para la creación de interfaces gráficos, programación web, multihilo, bases de datos, etc.
- Permite realizar programación visual y programación dirigida por eventos.
- Características avanzadas de IDE: sintaxis coloreada, complexión automática de código, ayuda sensible al contexto, inspector de objetos, diseñador visual, compilador y depurador integrado, etc.
- Completamente orientado a objetos.
  (Nokia Corporation, 2008-2011)

2.8 Herramientas para el análisis léxico y sintáctico

Para hacer más fácil la creación de los compiladores se han desarrollado programas que ayudan en gran medida a la construcción de los mismos, atendiendo principalmente las áreas más engorrosas y difíciles de implementar (scanner, parser). Entre las herramientas que avalan esta afirmación se encuentran: Flex, generador de analizadores léxicos y Bison, generador de analizadores sintácticos.

2.8.1 Flex, generador de analizadores sintácticos

Flex es una herramienta para generar escáneres: programas que reconocen patrones léxicos en un texto. Flex lee los ficheros de entrada dados, o la entrada estándar si no se le ha indicado ningún nombre de fichero, con la descripción de un escáner a generar. La descripción se encuentra en forma de parejas de expresiones regulares y código C, denominadas reglas. Flex genera como salida un fichero fuente en C, `lex.yy.c', que define una rutina `yylex ()'. Este fichero se compila y se enlaza con la librería `-lfl' para producir un ejecutable. Cuando se arranca el fichero ejecutable, este analiza su entrada en busca de casos de las expresiones regulares. Siempre que encuentra uno, ejecuta el código C correspondiente. (Paxson, 1995)

Qgis, contiene incorporadas las herramientas Flex/Bison por lo que no será necesario preocuparse por generar el código en C a partir del fichero de entrada, este proceso será inherente a la ejecución del sistema. Sin embargo, sí se requiere, para el correcto funcionamiento de Flex dentro de la aplicación, que dicho fichero sea elaborado de forma minuciosa previendo de este modo, los posibles errores en la definición de las expresiones regulares y reglas que lo componen.

Los desarrolladores de Qgis realizaron pequeñas transformaciones, tipificando los ficheros generados por Flex a sus propios nombres y extensiones. Un fichero de entrada para Flex fuera del Qgis tendría una extensión .l sin embargo dentro deberá ser .ll se, generará un fichero con extensión .cpp y cuyo nombre tendrá la sintaxis flex_nombre_original del fichero de entrada; por ejemplo para un fichero capas.ll se generaría flex_capas.cpp.

Un fichero de entrada para Flex está estructurado por secciones, ver Figura8.

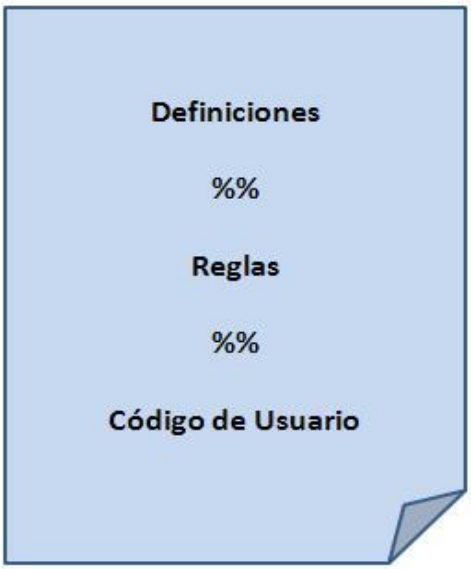


Figura 8.Estructura del fichero de entrada de Flex

1. **Sección de Definiciones:** Dentro de esta sección se encuentran las declaraciones de definiciones de nombres sencillos utilizados para abreviar la especificación del escáner.

   Ejemplo 5:

   DIGITO [0-9]

   NUMERO {DIGITO}+| {DIGITO}+\. {DIGITO}+

   De este modo cuando se requiera manipular la expresión regular que lo describe solo se necesita hacer referencia a la palabra DIGITO o NUMERO. Es fácil notar que dentro de NUMERO se está referenciando a la expresión regular DIGITO declarada anteriormente, aunque si la acción apareciera de este modo: NUMERO [0-9]+| [0-9] +\. [0-9]+ se obtendría el mismo resultado.

2. **Seccion de Reglas:** Las reglas siguen una estructura Patrón - - - Acción, donde cada patrón en una regla debe tener una acción asociada que suele ser cualquier sentencia en C.

   Ejemplo 6:

   NUMERO {yylval.double = atof (yytext); return (NUMERO) ;}

   La sección de reglas siempre comienza después del símbolo repetido %% y termina con la aparición del mismo. La acción debe encontrarse a continuación del patrón y siempre en la misma línea. Se utilizan variables predefinidas dentro de Flex como es el caso de yylval y yytext. En este ejemplo cuando el scanner identifique un número el valor será guardado en una variable y se retornara NUMERO indicando que la estructura léxica encontrada fue del tipo token NUMERO.

3. **Sección de Código Usuario:** Es una sección opcional pero que en caso de ser utilizada se copia de forma íntegra en el fichero `lex.yy.c' donde es muy estilada para rutinas de complemento que llaman al escáner o son llamadas por este.

```
DIGITO      [0-9]
NUMERO    {DIGITO}+| {DIGITO}+\. {DIGITO}+

%%

NUMERO          {yylval.double = atof(yytext); return (NUMERO);}

%%

Main()
{
  yylex();
}
```

Figura 9.Ejemplo de fichero de entrada de Flex

2.8.2 Bison, generador de analizadores sintácticos

Bison es un generador de analizadores sintácticos de propósito general que convierte una descripción para una gramática independiente del contexto en un programa en C que analiza esa gramática. Es un desarrollo realizado por GNU bajo licencia GPU, usándose junto a Flex, esta herramienta permite construir compiladores de lenguajes muy potentes. La forma general de un fichero de entrada de Bison es:

Figura 10.Estructura de un fichero de Bison

1. **Declaraciones en C:** En esta sección se realizan declaraciones de las funciones y macros que serán utilizadas en las acciones de las reglas de la gramática. Generalmente se utiliza "#include" para obtener las declaraciones de un archivo de cabecera.

   Ejemplo 7:

   %{

   #include <QList>

   #include <cstdlib>

   #include "qgsproject.h"

   %}

De este modo se puede acceder a las funcionalidades que las clases incluidas proporcionan.

2. **Declaraciones de Bison:** Es en esta sección dónde se permite declarar los símbolos terminales y no terminales de la gramática. También se especifican los tipos de dato de los símbolos terminales y no terminales utilizando para ello la directiva "%union".Se precisa además el símbolo de arranque de la gramática a través de "start".

   Ejemplo 8:

   %union {float f_type; char c_type ;}

   %start mapfile

   %token <f_type> NUMERO

   %type <c_type> seccion

   En el momento en que el analizador este examinando el token NUMERO sabrá que el tipo de dato que acepta es un float y ocurrirá lo mismo con el no terminal "seccion" con su respectivo tipo de dato. Al encontrar la directiva "%start "el analizador identificará, que este no terminal, representa el comienzo de la gramática.

3. **Reglas Gramaticales:** Dentro de esta sección es donde está estipulado que se construya la gramática, las gramáticas de Bison se construyen siguiendo una estructura de la forma:

   bosque**:** arboles | arbol

   ;

   De modo que bosque es el símbolo no terminal que describe esta regla y "arboles o arbol "son los diversos símbolos terminales y no terminales que están reunidos por esta regla.

4. **Código C adicional:** En esta seccion se copia exactamente igual la salida del fichero del analizador léxico, unido a la seccion de declaraciones de C

realizada al comienzo del fichero de Bison. Esta es la seccion apropiada para incluir cualquier código que no deba ir antes de la llamada al método yyparse.

```
%{
#include <QList>
void yyerror(const char* msg);
%}

%union {float f_type; char c_type ;}

%start selva

%token <f_type> NUMERO
%type <c_type> arboleda

%%
bosque: arboles | arbol
        ;
%%
void yyerror(const char* msg){ //gParserErrorMsg = msg;}
```

Figura 11.Estructura de un fichero de entrada de Bison

La fuente (fichero de entrada) suele ser un fichero con extensión .y donde se describe una gramática, debido a las modificaciones realizadas en el Qgis la extensión será .yy. El código que se genera determina si el fichero de entrada utilizado pertenece o no al lenguaje generado por esa gramática y el mismo puede ser accedido en las clases generadas .cpp y .hpp.

Dentro de Bison la sección de declaraciones está encargada de declarar los símbolos terminales y no terminales, también la precedencia de operadores y los tipos de datos de los valores semánticos.

2.9 Conclusiones Parciales

La selección de estas tecnologías garantiza en gran medida una guía durante todo el proceso de desarrollo en la personalización de Quantum Gis, se decide implementarla en el lenguaje C++ utilizando como EDI a Qt Creator y siguiendo una AS basada en componentes y orientada a objetos como la del programa original. Bajo la tutela de la metodología RUP, donde los diagramas serán modelados utilizando el instrumento Visual Paradigm de CASE y las herramientas Flex y Bison auxiliarán la construcción de las fases de análisis léxico y sintáctico del compilador.

Capítulo 3.Descripción de la Solución Propuesta

3.1 Introducción

En el presente capítulo se describen las características de la solución propuesta, para una mejor comprensión del contexto en que se desarrolla el sistema, a través de la realización del modelo de dominio, la identificación de los requisitos funcionales y no funcionales, elaboración del diagrama de casos de uso del sistema así como la descripción textual correspondiente a los mismos. Las actividades mencionadas garantizan el inicio de la fase de diseño del sistema.

3.2 Modelo del Negocio

El modelado del negocio logra crear una visión más profunda de la organización a la que se le realiza la automatización, permitiendo definir los procesos, roles y responsabilidades en los modelos de casos de uso del negocio. Permite comprender la estructura y la dinámica de la organización en la cual se va a implantar el sistema, además de los problemas actuales de dicha organización, e identificar así las mejoras potenciales. Realizar un modelado del negocio tiene como ventaja que se puedan derivar los requerimientos del sistema que va a soportar la organización.

Sin embargo, la investigación no está enmarcada en las necesidades de un cliente específico ni tiene por objetivo la creación de una aplicación independiente en la que deban identificarse procesos a automatizar sino, en personalizar la aplicación Quantum Gis a través de la incorporación de una nueva funcionalidad. No se concibe la necesidad de definir un Modelo del Negocio por lo que se procederá a trabajar en el desarrollo del Modelo del Dominio.

Se utiliza el término Modelo de Dominio para distinguirlo del Modelo de Negocios ya que este último es mucho más abarcador. El Modelo de Dominio centra su atención en una parte del Modelo del Negocio la relacionada con el ámbito del proyecto.

3.3 Modelo del Dominio

El Modelo de Dominio (MD) es una representación visual estática del entorno real del proyecto expresado a través de un diagrama con los objetos existentes relacionados con el proyecto que se va a acometer y las relaciones evidentes entre ellos. El MD se centra en una parte del negocio, la relacionada con el ámbito del proyecto; tiene por

objetivo principal ayudar a comprender los conceptos que utilizan los usuarios y los conceptos con que trabajan y que serán con los que deberá trabajar la aplicación. El MD representa una actividad clásica del análisis orientado a objetos.

3.3.1 Definiciones, Acrónimos y Abreviaturas del Modelo de Dominio

- *Archivo MapFile:* El MapFile es un componente muy importante de UMN MapServer, es un archivo con extensión ".map" en formato texto, que contiene todas las definiciones y configuraciones iniciales necesarias para la ejecución de un servidor de mapas UMN MapServer (Salinas, 2007).

- *Archivo .qgs:* Fichero con extensión .qgs escrito en lenguaje de etiquetas muy similar al XML dentro del cual se especifican las características de un proyecto perteneciente al software Quantum Gis y sobre el cual se realizan todas las operaciones de esta aplicación.
- *Objeto:* También conocido como sección, es el elemento que utiliza el MapFile para estructurar su contenido de acuerdo a una función determinada.
- *Cartografía:* La cartografía constituye un conjunto de operaciones que permiten a partir de observaciones y mediciones, la representación de una parte o la totalidad de la Tierra.

- *Mapas:* Representación geográfica de una parte de la superficie terrestre, en la que se da información relativa a una ciencia determinada.
- *Datos:* Información amplia o concreta respectiva al tema que se maneje.
- *Cliente:* Persona que utiliza los servicios del sistema.
- *Capas:* Elementos que superpuestos conforman un mapa, definen por separado la información a representar del mismo.
- *Clase:* Objeto contenido dentro de las capas define la temática que dicha capa representará dentro del mapa.
- *Proyección:* Define la proyección con que se representará la imagen de salida del mapa.
- *Escala:* Línea recta dividida en partes iguales que representan unidades de medida, sirve para dibujar proporcionadamente las distancias y dimensiones en un mapa, para luego calcular las medidas reales con respecto a lo dibujado, objeto que especifica la escala del mapa.
- *Leyenda:* Texto o símbolo que acompaña a un mapa y explica su contenido,

objeto que define la leyenda del mapa.

Las relaciones existentes entre los conceptos asociados al ámbito del sistema se reflejan en el correspondiente Diagrama de Clases del Modelo del Dominio.

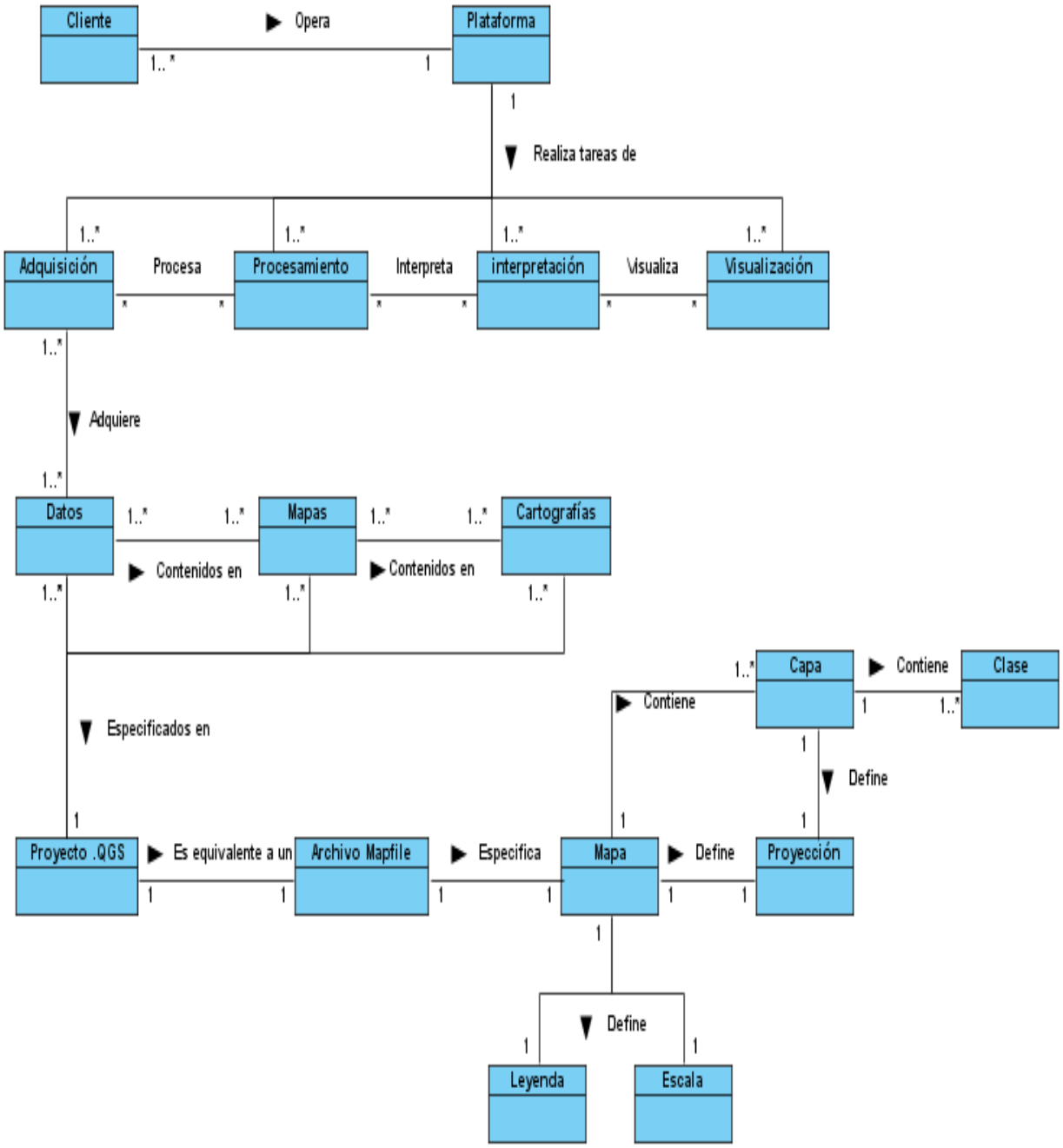


Figura 12.Diagrama de Clases del Modelo de Dominio

3.3.2 Descripción del Modelo de Dominio

El cliente es el usuario que interactúa con la plataforma, esta responde a las necesidades del mismo realizando las operaciones básicas de un SIG: visualización, interpretación, procesamiento y adquisición de los datos contenidos en mapas y cartografías; para realizar estas operaciones la plataforma trabaja sobre un fichero con extensión .qgs también conocido como archivo de proyecto de Qgis. La información especificada en este fichero es equivalente a la comprendida en un Mapfile que estructura su contenido respondiendo a objetos mapa, capa, clases, leyenda, proyección y escala.

3.4 Requisitos Funcionales

El sistema debe ser capaz de:

RF1: Cargar un Fichero Mapfile: El sistema debe permitir que el usuario pueda cargar un fichero con extensión .map.

3.5 Requisitos no Funcionales

Los requisitos no funcionales son aquellas cualidades o propiedades que el software debe poseer. Son estas propiedades las que hacen que el software sea confiable, eficiente, ágil y atractivo a los ojos del usuario.

RNF_Usabilidad: El sistema podrá ser utilizado por usuarios con conocimientos básicos en el manejo de la aplicación Quantum Gis.

RNF_Apariencia o interfaz externa: La nueva interfaz que se incluirá en la aplicación Qgis, encargada de importar el archivo mapfile, deberá ser amigable, intuitiva y de fácil comprensión para el usuario, facilitando en todo momento la interacción de este con el sistema y siguiendo la misma línea de diseño del Qgis.

3.6 Descripción del modelo propuesto

Cumplimentando los objetivos trazados al inicio de esta investigación y respondiendo a los requerimientos planteados, la personalización del sistema que se propone tiene como única funcionalidad importar un archivo Mapfile.

3.6 Descripción de los actores del sistema

Actores del Sistema	Descripción
Usuario	Persona facultada para proporcionar el fichero Mapfile al sistema.

3.7 Casos de Uso del Sistema.

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario. Por lo tanto, establece un acuerdo entre clientes y desarrolladores sobre las condiciones y posibilidades (requisitos) que debe cumplir el sistema.

Un diagrama de casos de uso del sistema representa gráficamente a los procesos y su interacción con los actores.

CUS1 Cargar Fichero MapFile.

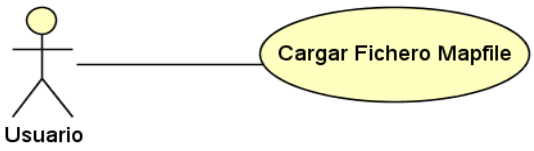


Figura 13.Diagrama de Casos de uso del Sistema

3.8 Descripción textual del caso de uso Cargar Fichero Mapfile

Caso de Uso:	Cargar Fichero MapFile
Actores:	Usuario
Resumen:	El caso de uso inicia cuando el usuario selecciona, la opción “Cargar fichero MapFile”. Esta funcionalidad permite importar un fichero MapFile original de MapServer en la aplicación

	Quantum Gis.
Precondiciones:	Debe existir el fichero MapFile.
Referencias	
Prioridad	Secundario

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1- El actor selecciona la opción de “Cargar fichero MapFile “dentro del menú “Archivo.”	2- El sistema muestra una ventana permitiendo la selección del fichero que se desea abrir.
3-El actor elige el fichero que desea abrir y selecciona la opción de “Abrir” o “Cancelar”, según lo que es deseado.	4- En caso de que la opción escogida por el usuario haya sido “Abrir”, el sistema accede a la ubicación del fichero MapFile y muestra las capas especificadas en él, dentro del lienzo de trabajo. 4.1-En caso contrario el sistema cierra la ventana.

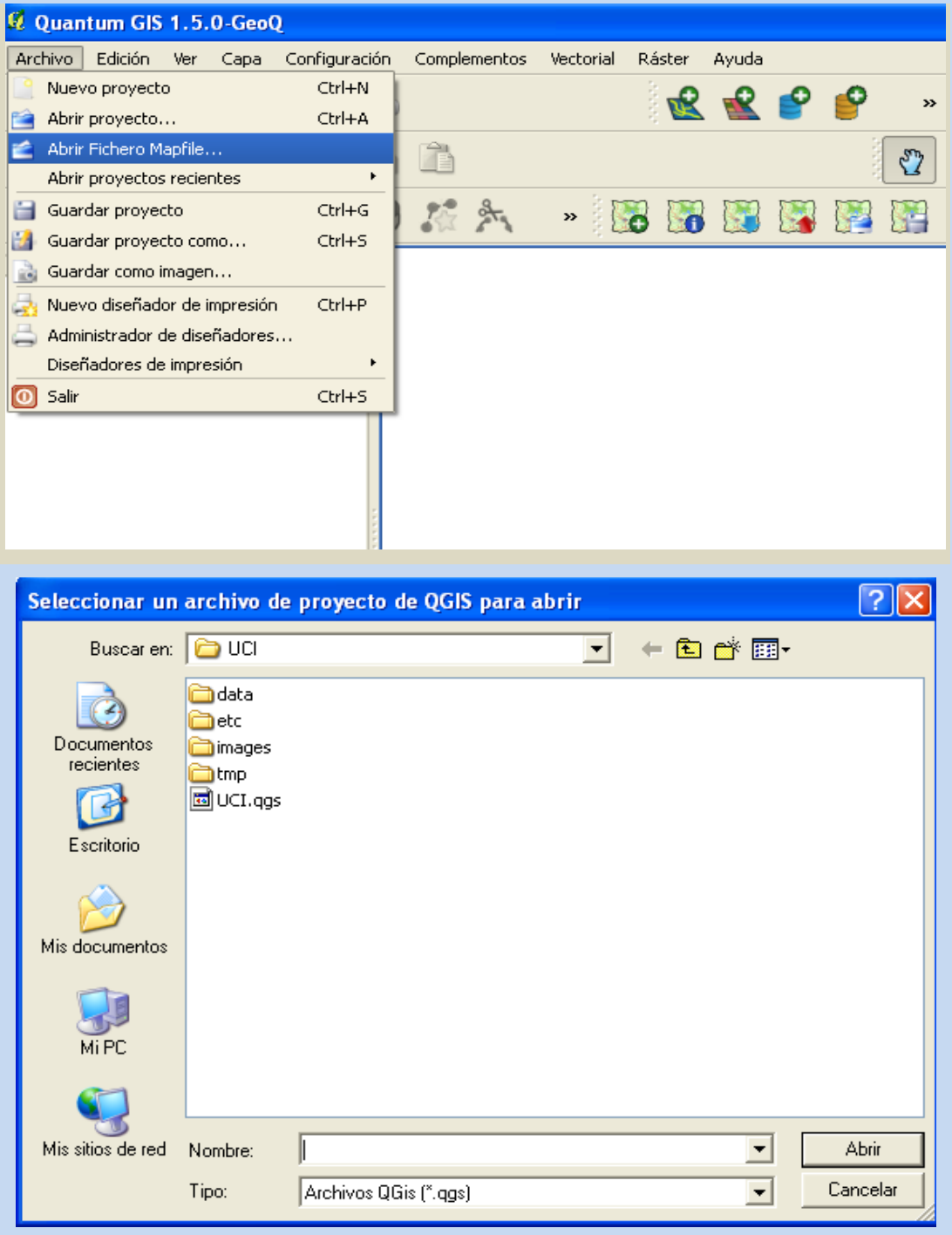
Prototipo de Interfaz	
	
Poscondiciones	

Tabla 2.Descripción del CUS Cargar Fichero Mapfile

3.9 Conclusiones Parciales

El proceso ingenieril desarrollado hasta el momento ofrece una amplia panorámica de cómo va desarrollándose la propuesta de solución para la personalización de Quantum Gis, documentando los artefactos que confirman que se encuentran sentadas las bases para el comienzo de la implementación de los requerimientos funcionales y no funcionales.

Capitulo 4. Construcción de la Solución Propuesta

4.1 Introducción

En este capítulo se pretende documentar el proceso de implementación que materializará la propuesta de solución mostrada en el modelo de clases del diseño. En el mismo se realiza un análisis de la conformación de los ficheros de entrada de las herramientas Flex y Bison así como el código generado a partir estos, formulándose además el examen de validación de la solución propuesta.

4.2 Modelo de Diseño

El modelo de diseño pretende crear un plano del modelo de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases. Procura lograr la descomposición de los trabajos de implementación en partes manejables a través de una comprensión de los aspectos relacionados con los requisitos funcionales y no funcionales unidos a las restricciones de los lenguajes de programación, sistemas operativos y tecnologías de distribución.

4.2.1 Diagrama de clases del diseño

Durante el diseño, el diagrama de clase se elabora para tener en cuenta los detalles concretos de la implementación del sistema, describiendo la realización física de los casos de uso que este engloba. Este tipo de diagrama presenta una estructura estática que muestra las relaciones entre las clases del sistema y las relaciones existentes entre ellas.

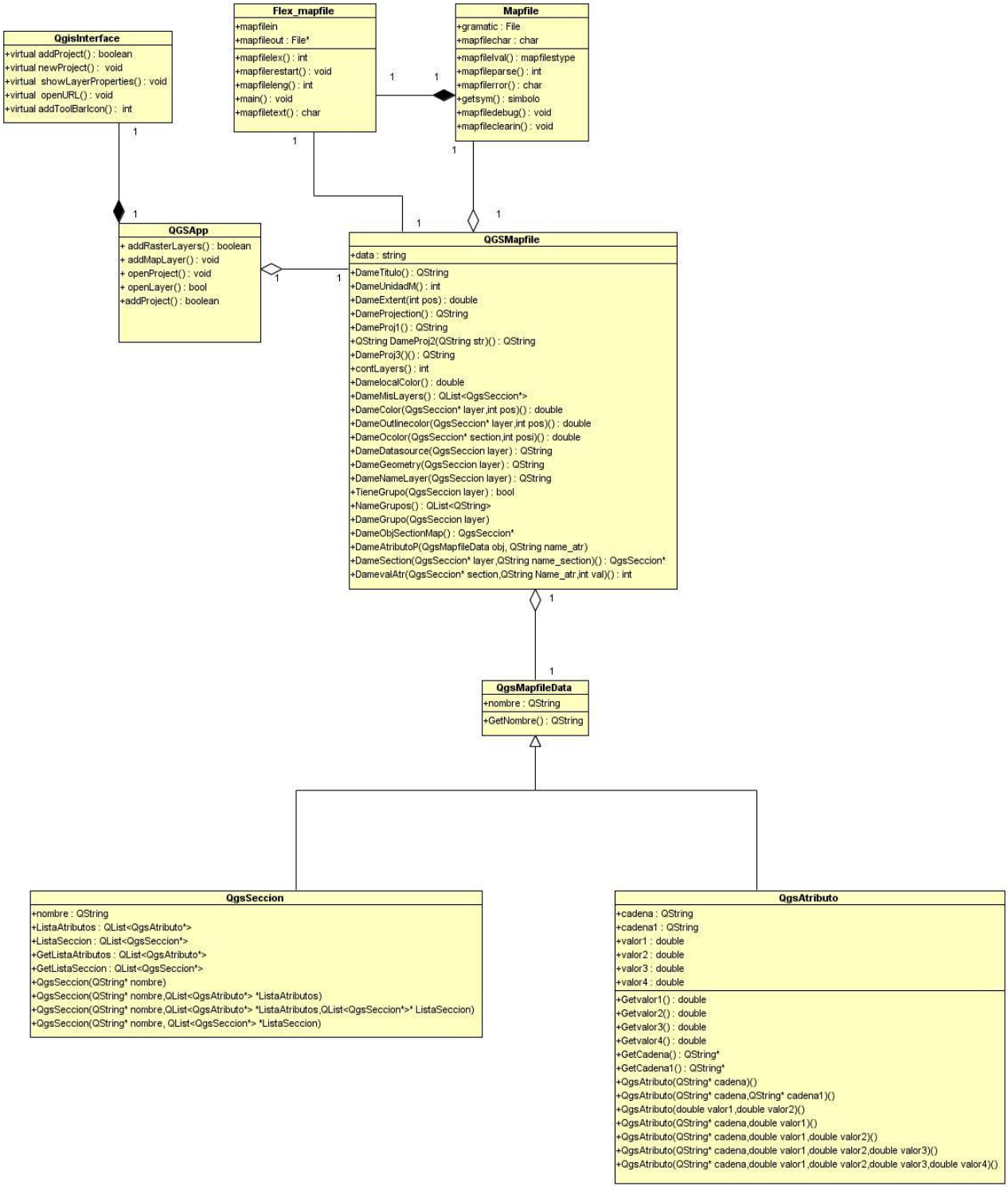


Figura 14.Diagramas de clases del diseño (Cargar Fichero MapFile)

4.2.2 Descripción del Diagrama de clases del diseño

La figura 16 muestra el diagrama de clases del diseño, en él aparecen las clases implicadas en la realización del caso de uso Cargar Fichero Mapfile.Se identifica la clase interfaz QgisInterface que posee una relación de composición con la clase controladora QgsApp en la cual se realizan todas las operaciones de la aplicación. Esta a su vez tiene una relación de agregación con la clase QgsMapfile, clase que auxilia el proceso de compilación a través de las clases escáner y parser. Se utiliza la clase QgsMapfileData de la que heredan las clases QgsAtributo y QgsSeccion ambas representan la estructura que debe conservar un objeto QgsMapfileData.

La clase QgsMapfile cuenta con los métodos necesarios para lograr enlazar los datos compilados con el constructor de la aplicación mostrando en pantalla la imagen especificada dentro del MapFile, como fruto de la cumplimentación del caso de uso en cuestión.

4.3 Generalidades de la implementación

Durante la fase de construcción se materializan en forma de código aquellos modelos desarrollados durante las fases de análisis y diseño siguiendo la guía de los patrones y arquitectura escogidos.

4.4 Diagrama de Componentes

Los diagramas de componentes modelan la vista estática del sistema muestran la organización y dependencias lógicas existentes entre sus componentes. Un componente generalmente está compuesto por clases. Ver figura 15.

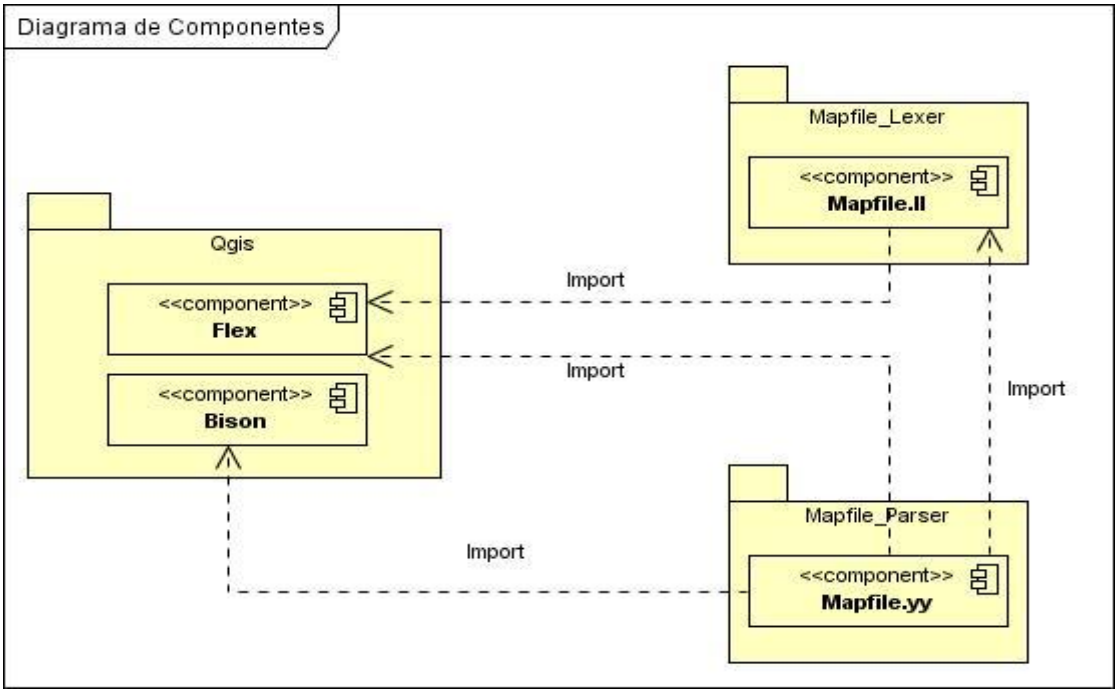


Figura 15.Diagrama de Componentes

4.5 Modelo de Despliegue

La nueva funcionalidad incluida dentro de la aplicación Qgis, no modifica a la misma con nuevos elementos que alteren su despliegue, de ahí que se decida mostrar el diagrama de despliegue oriundo de la aplicación.

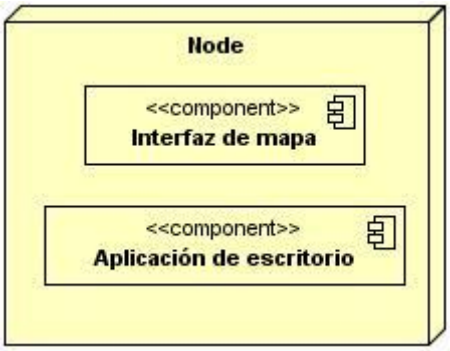


Figura 16.Diagrama de Despliegue de Qgis

4.6 Descripción de la Arquitectura

Según Garla y Perry la arquitectura no es más que... "*la estructura de los componentes de un programa o sistema, sus interrelaciones, y los principios y reglas que gobiernan su diseño y evolución en el tiempo*". (Reynoso, y otros, 2004)

Desde el surgimiento de la arquitectura de software en la práctica del diseño y la implementación se comenzaron a identificar un conjunto de regularidades que se repetían una y otra vez como respuesta a similares demandas, de ahí el surgimiento del nombre estilos arquitectónicos.

De acuerdo a la arquitectura candidata definida en el capítulo anterior y según lo planteado por la metodología RUP, la arquitectura tratará de mantenerse fiel a la seleccionada en la fase de inicio del desarrollo del software, conservando el estilo arquitectónico utilizado por los desarrolladores originales: llamada y retorno, específicamente la arquitectura orientada a objetos y la arquitectura basada en componentes.

4.7 Confección de los ficheros de entrada Flex/Bison

En el capítulo dos se hizo alusión a las herramientas Flex y Bison, mencionándose la necesidad de proporcionarle a las mismas, ficheros de entrada, donde a través de reglas, patrones, expresiones regulares y código en C se especifican las características de los analizadores léxico y sintáctico generados.

4.7.1 Declaraciones de Flex

Siguiendo el esqueleto modelo para la creación del fichero de Flex de nombre Mapfile.ll, se incluyeron dentro de la sección definiciones, las expresiones regulares que permiten identificar los tokens así como las opciones utilizadas que marcarán las características del compilador, se incluyeron además las cabeceras de aquellas clases que serán accedidas durante la generación del código. Ver Anexo I.

Continua a esta sección fueron definidas las reglas, las mismas aparecen dispuestas en la forma patrón---- acción, donde el patrón aparece sin sangría y la regla a continuación de este, en la misma línea. Dentro de las reglas se especificaron las acciones que realizará el analizador al identificar los diferentes tokens. Ver Anexo II.

Para culminar la construcción del fichero de Flex se incluyó el código en C adicional, el mismo es opcional y aparece sin transformaciones dentro del código en C generado al

ejecutar la aplicación. Se hizo uso de la función mapfile_scan_string (yy_scan_string ()) que orienta el scanner a analizar una cadena de texto que termine con un espacio en blanco. Ver Anexo III.

4.7.2 Declaraciones de Bison

Con el objetivo de garantizar la construcción del analizador sintáctico haciendo uso de la herramienta Bison, se confeccionó un fichero de entrada de nombre Mapfile.yy; dentro del mismo se incluyeron las declaraciones de funciones, macros y clases, que necesitan ser accedidas durante la ejecución del programa. Ver Anexo IV.

Los tokens que se le definen a Bison deben coincidir con los precisados en el fichero de entrada Mapfile.ll de Flex, los mismos deben identificarse atendiendo al tipo de dato tratando siempre de no crear inconsistencias de "tipo", para ello se utiliza la directiva %unión { }, donde se definen los tipos de datos tanto de terminales como no terminales de la gramática. Ver Anexo V.

Los terminales de la gramática utilizan la directiva token <tipo de dato> nombre, mientras los no terminales hacen uso del type<tipo de dato>nombre del no terminal. Ver Anexo VI.

Habiendo declarados los tipos de dato de la gramática, se procedió a definir la misma, declarando el axioma distinguido %start Mapfile, de esta forma cuando se encuentre el no terminal Mapfile comenzará el análisis de la gramática. Ver Anexos VII, VIII, IX, X, XI.

Luego de la confección de los ficheros de entrada de Flex y Bison se procedió a implantarlos dentro del código de la aplicación Qgis, donde en el CMakeLists de la misma se definieron las funciones encargadas de generar los ficheros .h, .hpp y .cpp correspondientes. Haciendo uso de comandos naturales de Bison. Ver tabla 3.

Comandos	Descripción
'–v'	Permite a Flex escribir en stderr[4] un sumario de las estadísticas respecto al analizador que genera.

[4] Contiene posibles errores que el programa muestra por pantalla al usuario.

<table>
<tr><td></td><td></td></tr>
<tr><td>‘–d’</td><td>Permite que el analizador generado se ejecute en modo de depuración.</td></tr>
<tr><td>‘–p(prefijo)’</td><td>Se utiliza en compiladores interactivos, permite enlazar múltiples programas Flex en el mismo ejecutable. Su esencia es la de modificar el nombre de las variables y funciones de Flex, sustituyendo la conocida “yy” por los caracteres que se le definan; en el caso de esta aplicación se utilizó “mapfile”.

Sintaxis del comando: -p mapfile mapfile.ll

<table>
<tr><th>Antes</th><th>Después</th></tr>
<tr><td>yylex()</td><td>mapfilelex()</td></tr>
<tr><td>yytext</td><td>mapfiletext</td></tr>
</table>
</td></tr>
</table>

Tabla 3 .Opciones utilizadas de Flex

4.8 Construcción del Árbol de Sintaxis Abstracta (AST)

En la medida en que se reconoce la gramática se hace necesario comenzar la construcción del árbol de sintaxis abstracta (AST).Para definir la arquitectura del mismo se utilizó el TDA Lista, así como una estructura de clases que responden a los objetos que lo conformarán. Ver Tabla 4.

Clase	Atributos	Descripción
QgsMapfileData	QString* nombre	Establecerá la jerarquía entre clases, todas sus hijas heredaran el atributo nombre que permitirá identificar posteriormente a qué sección o atributo se hace referencia dentro de

		la lista.
QgsAtributo	QString* nombre double val1,val2 double val3,val4	Contendrá un nombre y un valor en caso de ser un atributo simple de lo contrario podrá tener múltiples valores.
QgsSeccion	QString* nombre QList<QgsAtributo*>* ListaAtributo QList<QgsSeccion*> *ListaSeccion	Dispondrá de un nombre, una lista de atributos y una lista de secciones, debido a que las secciones están constituidas por atributos y pueden contener además a otras secciones como es el caso de MAP, LAYER, CLASS etc...

Tabla 4.Descripción de clases

Dentro de las producciones de la gramática se crearon atendiendo al tipo de dato y los objetos reconocidos durante el análisis sintáctico, objetos de tipo QgsSeccion y QgsAtributo, logrando conformar un único objeto contenedor de la seccion MAP, a partir del cual se pretende realizar la selección de los elementos necesarios para confeccionar un proyecto qgs que pueda ser interpretado por la aplicación QGis.

4.9 Código del Quantum Gis

Quantum Gis utiliza el lenguaje de etiquetas xml para estructurar los elementos que conforman un proyecto qgs, por lo que fue necesario realizar un análisis de las clases involucradas en el proceso de lectura y escritura XML dentro de la aplicación. El mismo tuvo lugar con el objetivo de reutilizar código y lograr una comprensión mínima de la lógica utilizada en el desarrollo de este software.

4.10 Pruebas del sistema propuesto

Los seres humanos tienen una habilidad innata de provocar errores y sus invenciones no están exentas de estos, de ahí que se requiera realizar varios test o pruebas que garanticen la calidad de los productos de software desarrollados por los mismos.

Los errores dentro del proceso de desarrollo del software pueden venir dados desde su creación durante las fases de inicio y análisis hasta su explotación durante el despliegue. La identificación errónea de requisitos o un mal diseño de clases pueden provocar faltas graves en el buen funcionamiento de una aplicación. De ahí que se realicen a las aplicaciones de software numerosas pruebas con el objetivo de descubrir fallas no detectadas hasta ese momento.

Sin embargo son disímiles los tipos de pruebas existentes y la selección del prototipo de prueba apropiado a determinado negocio debe responder a las características del mismo y a lo que se desee probar.

- Pruebas de Integración: Comprueban la compatibilidad y funcionalidad de los interfaces entre los distintos elementos que componen un sistema, pueden ser módulos, aplicaciones individuales, aplicaciones cliente/servidor, etc.
- Pruebas de Validación: Son realizadas sobre un software completamente integrado para evaluar el cumplimiento con los requisitos especificados.
- Prueba de Caja Blanca: Se basa en el diseño de casos de prueba que usen la estructura de control del diseño procedimental para derivarlos, en otras palabras se analiza la estructura lógica del programa.
- Prueba de Caja Negra: Este tipo de prueba se centra principalmente en los requisitos funcionales del software reflejados en su interfaz sin tener en cuenta el funcionamiento interno de la aplicación, no considera la codificación dentro de los parámetros a evaluar. Se basa en que las entradas sean aceptadas de forma adecuada y se reciba una salida correcta demostrando que cada función es completamente operativa.
- Prueba de Aceptación: Es la prueba final basada en las especificaciones del usuario o basada en el uso del programa por el usuario final. Su objetivo principal es demostrarle el cumplimiento del requisito de software al usuario. Puede estar asociado tanto a requisitos funcionales como no funcionales y cada requisito puede tener una o más pruebas de aceptación asociada.

"La prueba no puede asegurar la ausencia de defectos, sólo puede demostrar que existen defectos en el software" (Pressman, 2000)

El tipo de prueba y el método utilizado por ésta puede variar, pero su objetivo fundamental es el mismo, encontrar errores o defectos en el funcionamiento del software.

Realizar un proceso de pruebas que garantice la aptitud del software para satisfacer las necesidades del usuario requiere técnicas que guíen el desarrollo mismo, establecer una estrategia es un elemento clave, de modo que se pretende realizar casos de pruebas que auxilien la detección de fallas en el sistema.

La aplicación cuenta con un 70% de código generado por programas auxiliares lo que dificulta en gran medida su análisis lógico de ahí que se descarte el uso del tipo de prueba Caja Blanca. La misma no es un módulo ya que no constituye en esencia una aplicación sino una funcionalidad dentro de un software probado, lo que elimina las posibilidades de pruebas de Integración y Validación, y no cuenta con un cliente que realice las pruebas de Aceptación, por lo antes expuesto se selecciona como método de prueba la técnica funcional de Caja Negra.

4.10.1Técnica de Caja Negra

"Las pruebas de caja negra, también denominada prueba de comportamiento, se centran en los requisitos funcionales del software…" (Pressman, 2000)

Mediante las técnicas de prueba de caja negra se obtiene un conjunto de casos de prueba que satisfacen los siguientes criterios:

- Casos de prueba que reducen, en un coeficiente que es mayor que uno, el número de casos de prueba adicionales que se deben diseñar para alcanzar una prueba razonable.
- Casos de prueba que nos dicen algo sobre la presencia o ausencia de clases de errores en lugar de errores asociados solamente con la prueba que estamos realizando.

Para desarrollar la prueba de caja negra existen varias técnicas, entre ellas están: (Pressman, 2000)

- Técnica de la Partición de Equivalencia: esta técnica divide el campo de entrada en clases de datos que tienden a ejercitar determinadas funciones del software.
- Técnica del Análisis de Valores Límites: esta Técnica prueba la habilidad del programa para manejar datos que se encuentran en los límites aceptables.
- Técnica de Grafos de Causa-Efecto: es una técnica que permite al encargado de la prueba validar complejos conjuntos de acciones y condiciones.

El método de partición equivalente es considerado uno de los más prácticos para la identificación de errores, el mismo divide el campo de entrada de un programa en clases de datos, de los que se pueden derivar casos de prueba.
"El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada…una clase de equivalencia representa un conjunto de estados validos o no validos para condiciones de entrada. Típicamente, una condici6n de entrada es un valor numérico específico, un rango de valores, un conjunto de valores relacionados o una condición lógica." (Pressman, 2000)
Para definir las clases de equivalencia se siguen un conjunto de directrices:

- Si una condición de entrada especifica un rango, se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada requiere se define una clase de equivalencia válida y dos no válidas.
- Si una condición de entrada especifica un miembro de un conjunto, se define una clase de equivalencia válida y una no válida.
- Si una condición de entrada es lógica, se define una clase de equivalencia válida y una no válida.

Al aplicar estas directrices en la obtención de clases de equivalencia se ejecutan casos de prueba para cada elemento de datos del campo de entrada, tratando siempre de ejercitar la mayor cantidad de atributos de cada clase de equivalencia a la vez.

Se muestra el caso de prueba para el caso de uso Cargar fichero Mapfile utilizando la técnica de partición equivalente.

Caso de uso: Cargar fichero Mapfile

Descripción general del caso de uso

El caso de uso comienza cuando el actor desea abrir un archivo mapfile y termina cuando el sistema representa la información contenida en el mismo.

Nombre de la Sección	SC: Abrir fichero mapfile	
Escenario de la sección	**Descripción de la funcionalidad**	**Flujo Central**
EC 1.1 Seleccionar fichero mapfile	Al acceder a la opción “Abrir archivo mapfile” del menú “Archivo “se muestra la ventana “Seleccionar archivo Mapfile”. La misma permite la selección del fichero mapfile. Complementariamente se tienen los botones “Abrir” y “Cancelar”.	1. Ventana Principal. 2. Menú Archivo. 3. Abrir fichero mapfile. 4. Seleccionar archivo mapfile.
EC 1.2 Abrir en Abrir fichero mapfile	Al seleccionar el botón “Abrir” dentro de la ventana “Seleccionar fichero mapfile” el sistema cargará de forma automática el archivo mapfile seleccionado anteriormente y mostrará sobre la interfaz principal su contenido cartográfico. En caso de error el sistema lanzará una excepción informando sobre el mismo.	1. Ventana Principal. 2. Menú Archivo. 3. Abrir fichero mapfile. 4. Seleccionar archivo mapfile. 5. Botón Abrir
EC 1.3 Cancelar en Abrir fichero mapfile.	Al seleccionar el botón “Cancelar” el sistema debe cerrar la ventana “Seleccionar archivo mapfile” cancelando la operación.	1. Ventana Principal. 2. Menú Archivo. 3. Abrir fichero mapfile.

		4. Seleccionar archivo mapfile. 5. Botón Cancelar.

Tabla 5. Secciones a probar en el caso de uso Cargar fichero mapfile

Descripción de las variables

No.	Nombre del campo	Clasificación	Requerido	Descripción
1	Nombre	Campo de Texto	Si	Especifica el nombre del fichero que se desea abrir.

Tabla 6.Descripción de las variables para el caso de uso Cargar fichero mapfile

Matriz de datos (SC 1 Cargar fichero Mapfile)

Escenario	Nombre	Respuesta del sistema	Resultado de la prueba
EC 1.1	**V**	Al acceder a la opción "Cargar fichero Mapfile" el sistema muestra la ventana que permite la selección del mismo.	

Tabla 7.Matriz de Datos (SC 1 cargar fichero mapfile)

4.11 Conclusiones Parciales

En este capítulo se expusieron y detallaron los argumentos necesarios para comprender la materialización del caso de uso Cargar Fichero Mapfile, que tuvo lugar durante la fase más engorrosa del desarrollo de software, implementación y prueba. Sé aseguró de este modo el cumplimiento del objetivo primario de esta investigación al lograr que la herramienta Qgis representara la información cartográfica contenida en ficheros mapfile.

Conclusiones

Concluido el proceso de incorporación de nuevas funcionalidades a la aplicación SIG de escritorio Qgis, objetivo primario de la presente investigación, se arribó a los siguientes resultados:

Permanecen reflejados en la documentación los conceptos fundamentales que permiten entender la lógica y funcionamiento del entorno de trabajo donde se desarrolló la misma.

El instrumental tecnológico seleccionado para la personalización de la aplicación Qgis propició la optimización del capital humano y la correcta materialización del objetivo de la investigación.

Los requerimientos funcionales y no funcionales identificados durante el levantamiento de requisitos fueron implementados en su totalidad y debidamente probados.

Alcanza mayor relevancia el resultado de la presente investigación si se considera que no existen soluciones semejantes a la desarrollada.

Se logró documentar todo el proceso de desarrollo de la misma para posteriores estudios y modificaciones.

Recomendaciones

Se recomienda perfeccionar la gramática del compilador, de modo que permita la lectura de una variada gama de ficheros Mapfile.

Referencias Bibliográficas

ERDAS Inc. Erdas,The Earth to Business Company [Online] // Erdas,The Earth to Business Company. - ERDAS Inc., 2011. - 1 4, 2011. - http://www.erdas.com/Homepage.aspx.

Alconchel Miguel Ángel Aguilar Chomsky la Gramática Generativa [Journal]. - 2004. - 7 : Vol. 3.

ArcGis Community ArcGis Online [Online] // ArcGis Online. - Esri Community, 2011. - 1 5, 2011. - http://www.arcgis.com/home/.

Avella Joel David Rojas Lenguaje natural y lenguaje formal [Report]. - 2009.

Ballari Daniela Instalación de MapServer como WMS, WFS y WCS [Report]. - Universidad Politécnica de Madrid : [s.n.], 2001.

Barbosa Dr. José Seguinot Pasado, presente y futuro de los SIG [Report]. - Puerto Rico : [s.n.], 2007.

Booch G, Rumbaugh J and Jacobson I El Proceso Unificado de Desarrollo de Software [Book]. - 2000.

Carrillo Pérez Isaías, Pérez González Rodrigo and Rodríguez Martín David Aureliano Metodología de Desarrollo de Software [Report]. - 2008.

Case CASE Tools [Online] // CASE Tools. - 2011. - 2 4, 2011. - http://case-tools.org/.

Case.Herramientas Case www.elprisma.com [Online] // www.elprisma.com. - enero 3, 2011. - http://www.elprisma.com/apuntes/curso.asp?id=13324...

César Ignacio García Osorio Introducción al proceso de compilación [Online] // http://www.scribd.com/doc/48961961/CompiladoresClase2. - 5 5, 2011. - http://www.scribd.com/doc/48961961/CompiladoresClase2.

Cubillo Darío Rodríguez Proyecto Final del Máster en Tecnologías de la Información Geográfica [Report]. - Universidad Autònoma de Barcelona : [s.n.], 2000.

Díaz M.Luisa González Introducción a la construcción de compiladores [Report]. - Valladolid : [s.n.].

Díaz M.Luisa González Introducción a la construcción de compiladores [Report]. - Valladolid : [s.n.], 2000.

Entorno Virtual de Aprendizaje(EVA-UCI) Teoría de Lenguajes y Compilación. [Conference] // http://eva.uci.cu/mod/resource/view.php?id=23514. - La Habana : [s.n.], 2008-2009.

Firesmith Donald Diccionario de tecnología de Objetos [Book]. - 1993.

Fuentes Lidia, Troya Jose M and Vallecillo Antonio Desarrollo de Software Basado en Componentes [Report]. - Málaga : [s.n.], 2003.

Fundación Plone Portal GvSig [Online] // Portal GvSig. - Fundación Plone , 2010-2011. - 1 3, 2011. - http://www.gvsig.org/web/.

Gálvez Rojas Sergio and Mora Mata Miguel Ángel Java a Tope:Traductores y compiladores con Lex/Yacc,JFlex/Cup y Java CC [Book]. - Málaga : [s.n.], 2005.

GRASS Development Team GRASS GIS [Online] // GRASS GIS. - GRASS Development Team, 1999-2011. - 1 4, 2011. - http://grass.osgeo.org/.

Hernández Enrique Orallo El UML [Report].

Herramientas Case Visual Paradigm [Online] // Visual Paradigm. - 2011. - 2 3, 2011. - http://www.visual-paradigm.com/.

Jalón Javier García de Aprenda C++ como si estuviera en primero [Book]. - Navarra : [s.n.].

Joven Club de Computación y Electrónica Debate en el Ciberespacio [Online] // Debate en el Ciberespacio. - 2009-2010. - 5 5, 2011. - http://foro.jovenclub.cu/index.php?PHPSESSID=69606834cf5c1b334688a49d8594f714&topic=7792.msg48450#msg48450.

Martínez Alejandro and Martínez Raúl Guía a Rational Unified Process [Book]. - Castilla : [s.n.].

Narcizo Flor Introducción al Proceso de Compilación [Report]. - Los Andes : [s.n.], 2004.

Nokia Corporation Qt Creator [Online] // Qt Creator. - 2008-2011. - 3 8, 2011. - http://qt.nokia.com/products/developer-tools.

Open Source Geospatial Foundation Quantum Gis [Online] // Quantum Gis. - Open Source Geospatial Foundation(OSGeo), 2011. - 1 5, 2011. - http://www.qgis.org/.

OSGeo Guia de Usuario QGis [Report]. - 2004-2009.

Paxson Vern Flex, versión 2.5, Un generador de analizadores léxicos rápidos [Book]. - 1995. - Vol. Edición 2.5.

Pitney Bowes Software Inc. MapInfo [Online] // MapInfo. - 2010. - 1 3, 2011. - http://www.pbinsight.com/welcome/mapinfo/.

Pressman Roger S Ingeniería de Software un Enfoque Práctico [Book]. - 2005. - Vol. 6ta Edición.

Ramsey Paul Manual de PostGis [Report].

Regents of the University of Minnesota MapServer [Online] // MapServer. - Regents of the University of Minnesota, 2011. - 1 5, 2011. - http://www.mapserver.org/.

Reynoso Carlos and Kiccillof Nicolás Estilos y Patrones en la Estrategia de Arquitectura de Microsoft [Book]. - 2004.

RUMBAUHG J JACOBSON I El Lenguaje Unificado de Modelado. Manual de Referencia [Book]. - Madrid : Pearson Educación, 2000.

Sendra Bosque Los Sistemas de información Geográfica [Report]. - 1999.

Torres Jordi [et al.] Edición y visualización de información vectorial en aplicaciones SIG [Report]. - San Sebatián : [s.n.], 2009.

Universidad de las Ciencias Informáticas Introducción a las técnicas de Compilación [Conference] // Introducción a las técnicas de Compilación. - Ciudad de la Habana : [s.n.], 2008-2009.

Zayas Dr. Cs. Carlos Alvarez de Metodología de la Investigación Científica [Book]. - Santiago de Cuba : [s.n.], 1995.

Referencia de Figuras

(1)Tomado del sitio web (http://www.aulati.net/?tag=google-maps,2011)

Bibliografía Consultada

ERDAS Inc. Erdas,The Earth to Business Company [En línea] // Erdas,The Earth to Business Company. - ERDAS Inc., 2011. - 4 de 1 de 2011. - http://www.erdas.com/Homepage.aspx.

Alconchel Miguel Ángel Aguilar Chomsky la Gramática Generativa [Publicación periódica]. - 2004. - 7 : Vol. 3.

ArcGis Community ArcGis Online [En línea] // ArcGis Online. - Esri Community, 2011. - 5 de 1 de 2011. - http://www.arcgis.com/home/.

Avella Joel David Rojas Lenguaje natural y lenguaje formal [Informe]. - 2009.

Balcázar José L. Procesadores de Lenguaje [Informe]. - Cantabria : Universidad de Cantabria, 2010.

Ballari Daniela Instalación de MapServer como WMS, WFS y WCS [Informe]. - Universidad Politécnica de Madrid : [s.n.], 2001.

Barbosa Dr. José Seguinot Pasado, presente y futuro de los SIG [Informe]. - Puerto Rico : [s.n.], 2007.

Booch G, Rumbaugh J y Jacobson I El Proceso Unificado de Desarrollo de Software [Libro]. - 2000.

Carrillo Pérez Isaías, Pérez González Rodrigo y Rodríguez Martín David Aureliano Metodología de Desarrollo de Software [Informe]. - 2008.

Case CASE Tools [En línea] // CASE Tools. - 2011. - 4 de 2 de 2011. - http://case-tools.org/.

Case.Herramientas Case www.elprisma.com [En línea] // www.elprisma.com. - 3 de enero de 2011. - http://www.elprisma.com/apuntes/curso.asp?id=13324...

César Ignacio García Osorio Introducción al proceso de compilación [En línea] // http://www.scribd.com/doc/48961961/CompiladoresClase2. - 5 de 5 de 2011. - http://www.scribd.com/doc/48961961/CompiladoresClase2.

Cubillo Darío Rodríguez Proyecto Final del Máster en Tecnologías de la Información Geográfica [Informe]. - Universidad Autònoma de Barcelona : [s.n.], 2000.

Díaz M.Luisa González Introducción a la construcción de compiladores [Informe]. - Valladolid : [s.n.].

Díaz M.Luisa González Introducción a la construcción de compiladores [Informe]. - Valladolid : [s.n.], 2000.

Domínguez Alberto Entorno gráfico para la ejecución y depuración de código intermedio [Informe]. - Valencia : [s.n.], 2010.

Entorno Virtual de Aprendizaje(EVA-UCI) Teoría de Lenguajes y Compilación. [Conferencia] // http://eva.uci.cu/mod/resource/view.php?id=23514. - La Habana : [s.n.], 2008-2009.

Firesmith Donald Diccionario de tecnología de Objetos [Libro]. - 1993.

Fuentes Lidia, Troya Jose M y Vallecillo Antonio Desarrollo de Software Basado en Componentes [Informe]. - Málaga : [s.n.], 2003.

Fundación Plone Portal GvSig [En línea] // Portal GvSig. - Fundación Plone , 2010-2011. - 3 de 1 de 2011. - http://www.gvsig.org/web/.

Gálvez Rojas Sergio y Mora Mata Miguel Ángel Java a Tope:Traductores y compiladores con Lex/Yacc,JFlex/Cup y Java CC [Libro]. - Málaga : [s.n.], 2005.

González Pedro Expresiones regulares en STklos [Informe]. - 2005.

GRASS Development Team GRASS GIS [En línea] // GRASS GIS. - GRASS Development Team, 1999-2011. - 4 de 1 de 2011. - http://grass.osgeo.org/.

Habing Thomas G. XML Tutorial [Informe]. - 2004 : [s.n.].

Hernández Enrique Orallo El UML [Informe].

Herramientas Case Visual Paradigm [En línea] // Visual Paradigm. - 2011. - 3 de 2 de 2011. - http://www.visual-paradigm.com/.

Jalón Javier García de Aprenda C++ como si estuviera en primero [Libro]. - Navarra : [s.n.].

Joven Club de Computación y Electrónica Debate en el Ciberespacio [En línea] // Debate en el Ciberespacio. - 2009-2010. - 5 de 5 de 2011. - http://foro.jovenclub.cu/index.php?PHPSESSID=69606834cf5c1b334688a49d8594f714&topic=7792.msg48450#msg48450.

Martínez Alejandro y Martínez Raúl Guía a Rational Unified Process [Libro]. - Castilla : [s.n.].

Narcizo Flor Introducción al Proceso de Compilación [Informe]. - Los Andes : [s.n.], 2004.

Nokia Corporation Qt Creator [En línea] // Qt Creator. - 2008-2011. - 8 de 3 de 2011. - http://qt.nokia.com/products/developer-tools.

Open Source Geospatial Foundation Quantum Gis [En línea] // Quantum Gis. - Open Source Geospatial Foundation(OSGeo), 2011. - 5 de 1 de 2011. - http://www.qgis.org/.

OSGeo Guia de Usuario QGis [Informe]. - 2004-2009.

Paxson Vern Flex, versión 2.5, Un generador de analizadores léxicos rápidos [Libro]. - 1995. - Vol. Edición 2.5.

Pitney Bowes Software Inc. MapInfo [En línea] // MapInfo. - 2010. - 3 de 1 de 2011. - http://www.pbinsight.com/welcome/mapinfo/.

Prado Elena Raja Casi todas las pruebas del software [Informe]. - 2007.

Pressman Roger S Ingeniería de Software un Enfoque Práctico [Libro]. - 2000. - Vol. 6ta Edición.

Ramsey Paul Manual de PostGis [Informe].

Regents of the University of Minnesota MapServer [En línea] // MapServer. - Regents of the University of Minnesota, 2011. - 5 de 1 de 2011. - http://www.mapserver.org/.

Reynoso Carlos y Kiccillof Nicolás Estilos y Patrones en la Estrategia de Arquitectura de Microsoft [Libro]. - 2004.

Rienzi Bruno Estándares y Tecnologías de base para la construcción de un Sistema de Información Geográfica Empresarial [Informe]. - 2010.

RUMBAUHG J JACOBSON I El Lenguaje Unificado de Modelado. Manual de Referencia [Libro]. - Madrid : Pearson Educación, 2000.

Sendra Bosque Los Sistemas de información Geográfica [Informe]. - 1999.

Sepúlveda Carlos Gustavo Infante Aplicaciones Geográficas Enriquecidas para Internet utilizando componentes [Informe]. - 2008.

Torres Jordi [y otros] Edición y visualización de información vectorial en aplicaciones SIG [Informe]. - San Sebatián : [s.n.], 2009.

Universidad de las Ciencias Informáticas Introducción a las técnicas de Compilación [Conferencia] // Introducción a las técnicas de Compilación. - Ciudad de la Habana : [s.n.], 2008-2009.

Zayas Dr. Cs. Carlos Alvarez de Metodología de la Investigación Científica [Libro]. - Santiago de Cuba : [s.n.], 1995.

Glosario de Términos

-C-

Cartografía: La cartografía constituye un conjunto de operaciones que permiten a partir de observaciones y mediciones, la representación de una parte o la totalidad de la superficie terrestre.

Compilador: Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior.

-E-

Edición de la cartografía: Conjunto de transformaciones de que es objeto la cartografía con el objetivo de resaltar e identificar aspectos relevantes sobre el contenido que la misma posee.

-M-

Mapfile: Fichero de configuración de MapServer, donde se especifican la estructura y composición de los datos cartográficos que serán representados.

-P-

Proceso de compilación: Proceso que engloba las fases del desarrollo de un compilador.

-S-

Sistema de información geográfico: Herramienta informática que integra hardware, software y datos geográficos que permitiendo capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada.

Anexos

Anexo I. Sección de Definiciones

```
%option noyywrap
%option case-insensitive
%option never-interactive
%option 8bit
%x coment

%{
#include <QString>
#include <QtGui>
#include <stdlib.h>
#include "qgsmapfiledata.h"
#include "qgsseccion.h"
#include "qgsatributo.h"
#include "qlist.h"
#include "qgsmapfile.h"
#include "mapfile.hpp"
%}

DIGITO          [0-9]
blanco          [ \t\r\n]+
EXPRESION_CS    \'[(A-Za-z0-9)*\."/""="_""-""?"]*\'
EXPRESION_CD    \"[(A-Za-z0-9)*"="_":""|""/""-""?"\'\.]*\"
ID              [a-z A-Z][A-Za-z0-9]*
NUMEROD         [-+]?{DIGITO}+\.{DIGITO}+
NUMEROE         [-+]?{DIGITO}+
```

Anexo II.Sección de Reglas

```
%%
"#"             {       BEGIN (coment);}
<coment>[^\n]*          {}
<coment>\n              {BEGIN (INITIAL);}
MAP                     { return (MAP);}
END                     {return (END);}
OUTPUTFORMAT            {return (OUTPUTFORMAT);}
LABEL                   { return (LABEL);}
REFERENCE               { return (REFERENCE);}
SCALEBAR                {return (SCALEBAR);}
PROJECTION              {return (PROJECTION);}
LEGEND                  {return (LEGEND);}
WEB                     {return (WEB);}
METADATA                {return (METADATA);}
CLASS                   {return (CLASS);}
STYLE                   {return (STYLE);}
LAYER                   {return (LAYER );}
COLOR                   {return (COLOR);}
SIZE                    {return (SIZE);}
EXTENT                  {return (EXTENT);}
IMAGECOLOR              {return (IMAGECOLOR);}
OUTLINECOLOR            {return (OUTLINECOLOR);}
KEYSIZE                 {return (KEYSIZE);}
{NUMEROE}               { mapfilelval.d_type = atof(mapfiletext);return (NUMEROE);}
{NUMEROD}               { mapfilelval.d_type = atof(mapfiletext);  return (NUMEROD);}
{ID}                    { mapfilelval.char_type =strdup(mapfiletext); return (ID);}
{EXPRESION_CS}          { mapfilelval.char_type =strdup(mapfiletext); return (EXPRESION_CS);}
{EXPRESION_CD}          { mapfilelval.char_type =strdup(mapfiletext);;return (EXPRESION_CD);}
{blanco}                { }
%%
```

Anexo III. Sección de Código Usuario

```
%%

/*----------------------------------Código de Usuario--------------------------------*/

void set_mapfile_input_buffer(const char* buffer)
{
  mapfile_scan_string(buffer);
}
```

Anexo IV. Declaraciones en C de Bison

```
%{
#include "qgsseccion.h"
#include "qgsmapfiledata.h"
#include "qgsatributo.h"
#include "qstringlist.h"
#include "qlist.h"
#include <QString>
#include <QtGui>
#include <qdebug.h>
#include <qglobal.h>
#include <QList>
#include <cstdlib>
#include "qgsproject.h"

QgsSeccion* parseMapFile( const QString& str, QString& mparserErrorMsg );
extern int mapfilelex();
extern char* mapfiletext;
extern void set_mapfile_input_buffer(const char* buffer);
QString mParserErrorMsg;
void mapfileerror(const char* msg);

QList<QgsSeccion*> ls;
QList<QgsAtributo *> LL;

%}
```

Anexo V. Declaraciones de Bison

```
%union { double d_type; char* char_type; QgsSeccion* section;      QgsAtributo* atributo;
QList<QgsAtributo*>* listatributo;  QList<QgsSeccion*>* listsection; }
%start mapfile
%token <char_type> COLOR
%token <char_type> SIZE
%token <char_type> EXTENT
%token <char_type> IMAGECOLOR
%token <char_type> OUTLINECOLOR
%token <char_type> KEYSIZE
%token <char_type> MAP
%token <char_type> END
%token <char_type> OUTPUTFORMAT
%token <char_type> LABEL
%token <char_type> SCALEBAR
%token <char_type> REFERENCE
%token <char_type> PROJECTION
%token <char_type> LEGEND
%token <char_type> WEB
%token <char_type> METADATA
%token <char_type> CLASS
%token <char_type> STYLE
%token <char_type> LAYER
%token <d_type> NUMEROD
%token <d_type> NUMEROE
%token <char_type> ID
%token <char_type> EXPRESION_CS
%token <char_type> EXPRESION_CD
%token <char_type> blanco
```

Anexo VI. Declaraciones de Bison

```
%type <section> seccion
%type <section> outputformat
%type <section> scalebar
%type <section> reference
%type <section> projection
%type <section> legend
%type <section> web
%type <section> class
%type <section> layer
%type <d_type> valor_num
%type <char_type> expresion
%type <atributo> color_atributo
%type <atributo> size_atributo
%type <atributo> extent_atributo
%type <atributo> imagecolor_atributo
%type <atributo> outlinecolor_atributo
%type <atributo> keysize_atributo
%type <listsection> lista_seccion_scalebar
%type <section> seccion_scalebar
%type <listsection> lista_seccion_legend
%type <section> seccion_legend
%type <listsection> lista_seccion_web
%type <section> seccion_web
%type <section> label
%type <section> style
%type <section> metadata
%type <listatributo> lista_meta
%type <atributo> meta
%type <section> seccion_class
%type <listsection> lista_seccion_class
%type <listsection> lista_seccion_layer
%type <section> seccion_layer
%type <listsection> lista_seccion
%type <section> mapfile
%type <listatributo> lista_atributo
%type <atributo> atributo
%type <atributo> multiple
%type <atributo> simple
```

Anexo VII. Reglas Gramaticales

```
mapfile:
 MAP lista_atributo lista_seccion END    { QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
 |MAP END                                {}
lista_seccion:
lista_seccion seccion     {$1->append($2; ); $$ = $1;}
 |seccion                 {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>(); lista->append($1); $$ = lista;}
;
seccion
 :outputformat          {ls.append($1); $$ = $1;}
 |scalebar              {ls.append($1); $$ = $1;}
 |reference             {ls.append($1); $$ = $1;}
 |projection            { ls.append($1); $$ = $1;}
 |legend                {ls.append($1); $$ = $1;}
 |web                   { ls.append($1); $$ = $1;}
 |class                 {ls.append($1); $$ = $1;}
 |layer                 {ls.append($1); $$ = $1;}
;
outputformat
 :OUTPUTFORMAT  lista_atributo END       {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
 |OUTPUTFORMAT   END                     {}
;
scalebar
 :SCALEBAR lista_atributo lista_seccion_scalebar lista_atributo END    { int i=0; for (i = 0; i<$4->count(); i++) $2->append($4-
>at(i)); QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
 |SCALEBAR lista_atributo lista_seccion_scalebar END              { QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
 |SCALEBAR  END                          {}
;
lista_seccion_scalebar
 :lista_seccion_scalebar seccion_scalebar          {$1->append($2); $$ = $1;}
 |seccion_scalebar                        {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>();lista->append($1); $$ = lista;}
;
seccion_scalebar
       :label                    {$$ = $1;}
;
```

Anexo VIII. Reglas Gramaticales

```
label
:LABEL lista_atributo END            {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
|LABEL END                           { }
;
reference
:REFERENCE lista_atributo END          {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
|REFERENCE END                       { }
;
projection
:PROJECTION lista_atributo END         {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
|PROJECTION END                      { }
;
legend
:LEGEND lista_atributo lista_seccion_legend END   {QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
|LEGEND END                          { }
;
lista_seccion_legend
:lista_seccion_legend seccion_legend     {$1->append($2);  $$ = $1;}
|seccion_legend                  {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>();lista->append($1);  $$ = lista;}
;
seccion_legend
:label                       {$$ = $1;}
;
class
:CLASS lista_atributo lista_seccion_class END       {QgsSeccion* sec = new QgsSeccion($1,$2,$3);  $$ = sec;}
|CLASS END                           { }
;
lista_seccion_class
:lista_seccion_class seccion_class       {$1->append($2);  $$ = $1;}
|seccion_class                       {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>(); lista->append($1);  $$ = lista;}
;
seccion_class
:style                       {$$ = $1;}
|label                       {$$ = $1;}
;
```

Anexo IX. Reglas Gramaticales

```
style
 :STYLE lista_atributo END          {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
 |STYLE   END                       { }
 ;
layer
 :LAYER lista_atributo lista_seccion_layer END        {QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
 |LAYER END                         { }
 ;
lista_seccion_layer
 :lista_seccion_layer  seccion_layer       {$1->append($2);$$ = $1;}
 |seccion_layer                    {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>();lista->append($1); $$ = lista;}
 ;
seccion_layer
 :metadata                  {$$=$1;}
 |class                     {$$=$1;}
 |projection                {$$=$1;}
 ;
web
 :WEB lista_atributo lista_seccion_web END   { QgsSeccion* sec = new QgsSeccion($1,$2,$3); $$ = sec;}
 |WEB END                           { }
 ;
 lista_seccion_web
  :lista_seccion_web seccion_web          {$1->append($2);$$ = $1;}
  |seccion_web                  {QList<QgsSeccion*>* lista=new QList<QgsSeccion*>();lista->append($1); $$ = lista;}
 ;
 seccion_web
 :metadata                  {$$=$1;}
 ;
 metadata
  :METADATA lista_meta END            {QgsSeccion* sec = new QgsSeccion($1,$2); $$ = sec;}
 |METADATA END                 { }
 ;
lista_meta :lista_meta meta                      {$1->append($2); $$ = $1;}
 |meta                         {QList<QgsAtributo*>* Lo =new QList<QgsAtributo*>();Lo->append($1); $$ = Lo;}
 ;
 meta
 :ID ID                           {$$= atr;}
 |ID expresion                    {$$=atr;}
 |expresion ID                    {QgsAtributo* atr= new QgsAtributo($1,$2); $$=atr;}
 |expresion expresion             {QgsAtributo* atr= new QgsAtributo($1,$2); $$=atr;};
```

Anexo X. Regla Gramaticales

```
Atributo
:simple                    {$$ = $1;}
|multiple                  {$$ = $1;}
;
simple:
ID ID                      {QgsAtributo* atr= new QgsAtributo($1,$2); $$=atr;}
|ID valor_num              {QgsAtributo* atr= new QgsAtributo($1,$2); $$=atr;}
|ID expresion              {QgsAtributo* atr= new QgsAtributo($1,$2); $$=atr;}
|expresion                 {QgsAtributo* atr= new QgsAtributo($1); $$=atr;}
|SIZE ID                   {QgsAtributo* atr= new QgsAtributo($1); $$=atr;}
;
multiple:color_atributo          {$$ = $1;}
|size_atributo                   {$$ = $1;}
|extent_atributo                 {$$ = $1;}
|imagecolor_atributo             {$$ = $1;}
|outlinecolor_atributo           {$$ = $1;}
|keysize_atributo                {$$ = $1;}
;
color_atributo
:COLOR valor_num valor_num valor_num          {QgsAtributo* atr = new QgsAtributo($1,$2,$3,$4);$$ = atr;}
;size_atributo:SIZE valor_num valor_num        {QgsAtributo* atr = new QgsAtributo($1,$2,$3);$$ = atr;}
;
extent_atributo
:EXTENT valor_num valor_num valor_num valor_num   {QgsAtributo* atr = new QgsAtributo($1,$2,$3,$4,$5);$$ = atr;}
;
imagecolor_atributo
:IMAGECOLOR valor_num valor_num valor_num     {QgsAtributo* atr = new QgsAtributo($1,$2,$3,$4);$$ = atr;}
;
outlinecolor_atributo
:OUTLINECOLOR valor_num valor_num valor_num   {QgsAtributo* atr = new QgsAtributo($1,$2,$3,$4);$$ = atr;}
;
keysize_atributo
:KEYSIZE valor_num valor_num                  {QgsAtributo* atr = new QgsAtributo($1,$2,$3);$$ = atr;}
;
valor_num
:NUMEROD                   {$$=$1;}
|NUMEROE                   {$$=$1;}
expresion:
EXPRESION_CS               {$$=$1;}
|EXPRESION_CD              {$$=$1;}
```

Anexo XI.Codigo en C Adicional

```
%%
QgsSeccion* parseMapFile( const QString& str, QString& parserErrorMsg )
{
Q_ASSERT( ls.count() == 0);
 set_mapfile_input_buffer(str.toUtf8().constData());

int res = mapfileparse();
if(res == 0)
{
Q_ASSERT( ls.count() == 1);
 return ls.takeAt(0);
 }
 else
QString("ERROR...!!!! ")
   return NULL ;
 }

return NULL;
}

void mapfileerror(const char* msg)
{
      mParserErrorMsg = msg;
}
```