Universidad de las Ciencias Informáticas

Trabajo para Optar por el Título

De

Ingeniero en Ciencias Informáticas

Título:

“Herramienta de compilación para la manipulación de Mapfile”

Autor:

Michel Antonio Fajardo Mera

Tutor:

Msc. Yusnier Valle Martínez

Ciudad de la Habana, 1 de junio del 2011

"Dedicarse a lo fácil, cuando se tienen bríos para intentar lo difícil, es despojar de dignidad al talento"

José Martí

Declaración de autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Universidad de las Ciencias Informáticas hacer uso del mismo en su beneficio. Para que así conste firmamos la presente a los ____ días del mes de _____ del año _____.

_____

Firma del Autor

Michel Antonio Fajardo Mera

_____

Firma del Tutor

Msc. Yusnier Valle Martínez

Dedicatoria

Dedico este trabajo de diploma a mis padres, a mis abuelos, a mis tíos, a mi familia en general. Por todo el cariño y toda una vida de constante dedicación y sacrificios, por darme todo lo que en esta vida he necesitado.

Agradecimientos

A mis padres Maira y José Antonio por darme tanto amor y cariño, por estar en cada momento sosteniendo mi vida, a los que lograron hacer de mí una persona de bien y profesional, por estar siempre a mi lado guiándome incondicionalmente por el camino de la verdad y el amor, a los que ni dedicándoles mi vida entera recompensaré jamás.
A mi abuela, abuelos, tías, tíos, primos, primas, en fin a toda mi familia y vecinos por creer en mí.
A mis amigos por estar siempre a mi lado.
A mis amigos de la Universidad y a todos mis compañeros de aula y de cuarto por darme fuerzas en esta batalla.
Al tutor por estar hay en cada momento que lo necesite, por su apoyo, por sus exigencias, él es el alma de esta tesis.
A nuestro tribunal por sus reconstructivas críticas.
A todas las personas que de una forma u otra hicieron posible este sueño.

Resumen

En la actualidad han cobrado gran interés los Sistemas de Información Geográfica por las múltiples aplicaciones que tienen este tipo de plataformas. En el centro GEYSED de la facultad 6 se ha creado GeneSIG, sistema que constituye una base para la personalización de soluciones SIG[1] a la medida. Con el fin de lograr la configuración y gestión de los mapas que se manipulan en dicha plataforma, se ha creado LiberMap, una aplicación que brinda servicios de catálogo y que tiene como objetivo principal la edición dinámica del fichero Mapfile. Para apoyar la importación de los mapas de MapServer al catálogo, se ha implementado una herramienta de compilación de Mapfiles que permitirá la manipulación de los archivos.

El compilador está desarrollado en el lenguaje Python, auxiliándose en las bibliotecas de Ply, las cuales son una traducción de las tradicionales herramientas de construcción de compiladores: LEX y YACC para el lenguaje antes citado. El desarrollo del software está basado en la metodología descrita por el Proceso Unificado de Desarrollo de Software (RUP), utilizando las técnicas de modelación establecidas por el Lenguaje Unificado de Modelado (UML).

Palabras Claves

Mapfile, Compilador, Fases de Compilación, Lexer, Parser, Análisis Lexicográfico, Análisis Sintáctico, Análisis Semántico, Árbol de Sintaxis Abstracta.

[1] SIG acrónimo de Sistema de Información Geográfica

Índice

Índice de Figuras

Índice de Tablas

Introducción

En el naciente siglo XXI, las nuevas Tecnologías de la Información y la Comunicación han tomado gran auge e importancia en su aplicación dentro de la sociedad en que vivimos. Hoy día ha cobrado interés a nivel mundial la aplicación de los Sistemas de Información Geográfica en muchas de las entidades que nos rodean. Los mismos pueden ser utilizados en diversas esferas, dígase para investigaciones científicas, la gestión de los recursos, gestión de activos, la arqueología, la evaluación del impacto ambiental, la planificación urbana, la cartografía, la sociología, la geografía histórica, el marketing entre otras.

Con el progreso de los SIG y el incremento de las necesidades de su uso, el número de estos, desarrollados sobre plataformas o tecnologías libres ha ido en aumento. El término de "software libre" ocupa hoy un lugar destacado en la búsqueda de una forma para alcanzar la soberanía o libertad tecnológica para numerosos países en desarrollo, gracias a las ventajas, características y flexibilidades que brinda este tipo de software.

Cuba no escapa de esta realidad. Actualmente tiene como objetivo para lograr el futuro desarrollo económico de la sociedad, el uso y explotación de las nuevas tecnologías de la ciencia y la computación. La Universidad de las Ciencias Informáticas juega un papel importante en el cumplimiento de esta tarea, la cual cuenta con centros de desarrollo que tienen como propósito la implementación y perfeccionamiento de Sistemas de Información Geográfica mediante el uso de herramientas libres. Con tal motivo y como vía para encaminarse en el conocimiento e impulso de los SIG en la UCI[2], en la esfera de producción de software, se han creado en la institución una serie de proyectos que buscan la concepción de nuevas herramientas que brinden soporte a estos sistemas.

El catálogo de mapas LiberMap, que ha sido desarrollado por el Departamento de GeoInformática, perteneciente al centro de desarrollo GeoInformática y Señales Digitales (GEYSED) de la Facultad 6, es un ejemplo de las aplicaciones que se han ido desarrollando en la institución. Es un producto con autonomía suficiente como para funcionar de manera independiente a un SIG, que cuenta con los protocolos necesarios para la comunicación con otros sistemas y ofrece servicios que permiten a los usuarios finales, la personalización de los datos y las funcionalidades. Este sistema, permite además

[2] UCI acrónimo de Universidad de Ciencias Informáticas

editar dinámicamente los ficheros de configuración Mapfile, imprescindibles en el funcionamiento de MapServer, servidor de mapas libre que se destaca por su eficiencia y calidad. LiberMap fue creado para complementar GeneSIG con la finalidad de contar con herramientas cubanas que faciliten no sólo el manejo de la información georeferenciada, sino que además tributen a la informatización de las esferas socioeconómicas del país que hoy se sustentan en software propietario para desarrollar estas tareas, y en peores casos, para suplir el trabajo manual de entidades que exhiben un deterioro en la productividad y desempeño, todo esto por la falta de sistemas que automaticen sus labores y apoyen la toma de decisiones, muchas veces de gran impacto en la sociedad **(1)**.

MapServer es un proyecto de software libre puesto a disposición de la comunidad. Ha sido desarrollado con las contribuciones realizadas por otros proyectos, donde el fichero Mapfile es el núcleo de esta herramienta, el mismo se comporta como un archivo de configuración, sirviendo de base para el desarrollo de diferentes funcionalidades en proyectos de software libre.

Precisamente la función principal de LiberMap es la edición dinámica del fichero .map, ya que parte de los usuarios que deben configurar el Mapfile lo hacen manualmente, lo que requiere de conocimientos informáticos, y el trabajo a la hora de modificarlo o configurarlo se hace más difícil debido al extenso volumen de información que se encuentra dentro del archivo.

Actualmente LiberMap importa el archivo Mapfile a través del acceso a la API[3] de MapServer que proveen las librerías Mapscript, cargando estos archivos con PHP, la cual no brinda todas las estructuras que posee el Mapfile.

A partir de la situación planteada se ha identificado el siguiente problema de investigación:

¿Cómo importar los ficheros de configuración de mapas (Mapfile) de MapServer en el catálogo de mapas LiberMap?

Para darle solución al problema planteado se definió como objeto de estudio: Técnicas y herramientas de desarrollo de compiladores.

Planteándose como objetivo general: Desarrollar una herramienta de compilación para los ficheros de configuración de mapas de MapServer.

[3] API acrónimo de Interfaz de Programación de Aplicaciones (*Application Programming Interface*)

El campo de acción de la investigación lo constituye: Desarrollo de compiladores en software libre para la especificación del fichero .map de MapServer.

Por lo que se infiere la idea a defender siguiente: El desarrollo de una herramienta de compilación permitirá la importación de los archivos Mapfile para la manipulación de los mismos, a través del catálogo de mapas LiberMap.

Para darle cumplimiento al objetivo general se definieron diferentes tareas de investigación que se relacionan a continuación:

- Realización de un estudio de los referentes teórico-prácticos que preceden la realización de este trabajo de diploma.
- Selección de las herramientas para el desarrollo del compilador.
- Realización del análisis, diseño e implementación del compilador.
- Realización de pruebas a la herramienta de compilación desarrollada.
- Elaboración de la documentación técnica del proceso de desarrollo del compilador.

Para darle cumplimiento a las tareas especificadas se emplearán los siguientes métodos científicos:

Métodos Teóricos

- **Analítico-Sintético:** Se usa cuando se investiga las características de los ficheros .map y las herramientas para la interpretación de los mismos, se realiza una división de las variadas formas de compilación de los lenguajes, para facilitar su estudio. Luego se establece el tipo de herramienta de compilación a utilizar, dado el estudio de las diferentes partes que lo conforman, posibilitando una mejor comprensión de sus particulares.
- **Análisis Histórico-Lógico:** Se emplea durante el análisis de la evolución de los compiladores, así como las herramientas que sirven para la interpretación de los ficheros ".map", donde se realiza un estudio de las diferentes aplicaciones que han ido surgiendo para la interpretación y edición del Mapfile, poniendo de manifiesto la lógica interna de su desarrollo.

- **Inductivo-Deductivo:** Se utiliza en la solución del problema donde a partir del estudio realizado se arriban a proposiciones generales y se infieren casos particulares por un razonamiento lógico que pueden ser verificados en la práctica.

Para facilitar su comprensión, el documento está estructurado en 4 capítulos:

Capítulo 1: Fundamentación Teórica: Se establecen conceptos que sustentan la investigación. Se realiza un análisis de los mismos, así como una descripción más detallada del objeto de estudio y de la situación problemática, logrando un mejor entendimiento de lo que se va a tratar en la solución del problema.

Capítulo 2: Tendencias y Tecnologías: Se presenta una fundamentación de las tecnologías actuales que facilitan el diseño de compiladores en software libres. Se justifican la utilización de todo el instrumental tecnológico utilizado en el desarrollo de la herramienta de compilación, así como las metodologías y lenguajes utilizados.

Capítulo 3: Descripción de la Solución Propuesta: En este capítulo se aborda todo lo referente al modelo de dominio, en el cual se analizan todas las entidades y conceptos presentes en el entorno donde se aplica el compilador. Se especifica detalladamente los requisitos funcionales y los no funcionales que cumple dicho sistema y se describen los actores y los casos de usos del sistema.

Capítulo 4: Construcción de la solución propuesta: Se visualiza el diseño del compilador que se realizó. En él se encuentran los principios de diseño, una explicación de cómo se realiza la compilación del fichero Mapfile, así como los estándares de programación utilizados. Además se realiza la validación de la solución propuesta.

Capítulo 1: Fundamentación Teórica

En el siguiente capítulo se establecen conceptos relacionados con el dominio del problema que sustentan la investigación, así como una descripción más detallada de la situación problemática, logrando un mejor entendimiento de lo que se va a tratar en la solución del problema.

1.1 Conceptos asociados al problema

1.1.1 Sistema de Información Geográfica

Un Sistema de Información Geográfica es una integración organizada de hardware, software y datos geográficos diseñado para capturar, almacenar, manipular, analizar y desplegar en todas sus formas la información geográficamente referenciada con el fin de resolver problemas complejos de planificación y gestión. También puede definirse como un modelo de una parte de la realidad referido a un sistema de coordenadas terrestres y construidas para satisfacer necesidades concretas de información. Es cualquier sistema de información capaz de integrar, almacenar, editar, analizar, compartir y mostrar la información geográficamente referenciada. En un sentido más genérico, los SIG son herramientas que permiten a los usuarios crear consultas interactivas, analizar la información espacial, editar datos, mapas y presentar los resultados de todas estas operaciones. **(2)**

Otro de los conceptos asociados a los SIG es que se pueden definir como una tecnología computacional compuesta por equipo (hardware), programas (software) y datos, empleados para capturar, editar, representar y lo más importante analizar información geográfica **(3)**.

1.1.2 ¿Qué es MapServer?

MapServer es un entorno de desarrollo en código abierto para la creación de aplicaciones SIG con el fin de visualizar, consultar y analizar información geográfica a través de la red, mediante la tecnología Web. Este puede ser utilizado como una aplicación CGI[4] o a través del acceso a la API de MapServer que proveen las librerías Mapscript. **(4)**

[4]CGI acrónimo de *Common Gateway Interface:* método para la transmisión de información hacia un compilador instalado en el servidor.

1.1.3 Mapfile

El Mapfile define los recursos que serán utilizados en la aplicación CGI. Contiene información acerca de cómo se debe dibujar el mapa, la leyenda y el resultado de realizar una consulta. Por tanto define parámetros de los datos, el despliegue y las consultas que serán usados en una aplicación con MapServer, se puede hablar del Mapfile como un archivo de configuración y que normalmente tienen la extensión ".map". **(5)**

El archivo .map consta de varias secciones, cada sección se inicia con el nombre de la sección y termina con la etiqueta END: *Figura 1*. El contenido de las secciones consiste en la definición de determinados parámetros del tipo "atributo-valor".

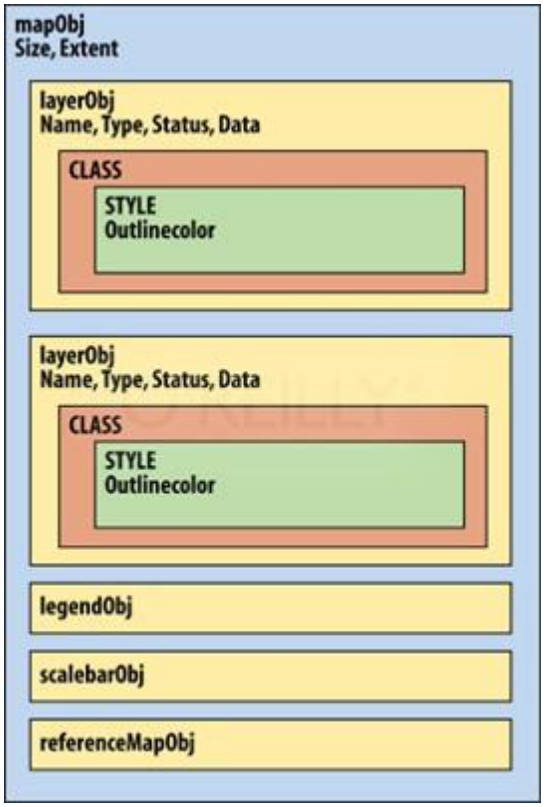


Figura 1: Estructura por objetos del Mapfile

Propiedades del fichero Mapfile organizado por objetos:

- MAP: Es el objeto principal y determina las propiedades del mapa en general.
  Propiedades del mapa contenidas dentro de la etiqueta: NAME, SIZE, STATUS, EXTENT, UNITS, IMAGECOLOR, IMAGETYPE, SHAPEPATH, FONSET, OFFSITE.

Contiene los objetos: LAYER, LEGEND, SCALEBAR, REFERENCE, WEB, PROJECTION, METADATA.

- LAYER: Este objeto determina las propiedades para una fuente de datos, se pueden crear tantos objetos LAYER como sean necesarios aunque el límite es de 100 por default.
  Propiedades contenidas dentro de la etiqueta: NAME, GROUP, TYPE, TYPERASTER, DATA, CONNECTIONTYPE, CLASSITEM, LABELITEM, HEADER, FOOTER, TRANSPARENCY, TOLERANCE, TILEINDEX, PROCESSING.

  Contiene objetos como: PROJECTION, METADATA, CLASS

- CLASS: Este objeto determina un conjunto de propiedades específicas para un objeto LAYER.

  Propiedades contenidas dentro de la etiqueta: NAME, COLOR, OUTLINECOLOR, EXPRESSION.

  Contiene objetos como: STYLE
- STYLE: Determina un conjunto de propiedades específicas para un objeto CLASS.

  Propiedades contenidas dentro de la etiqueta: LABEL, ANGLE, BACKGROUNDCOLOR, BACKGROUNDSHADOWCOLOR, BACKGROUNDSHADOWSIZE, COLOR, FONT, FORCE, MAXSIZE, MINSIZE, MINDISTANCE, OFFSET, OUTLINECOLOR, PARTIAL, POSITION, SHADOWCOLOR, SHADOWSIZE, SIZE, TYPE, IMAGECOLOR, KEYSIZE, KEYSPACING, STATUS, SYMBOLSET, BUFFER, MINDISTANCE, PARTIALS.

- LABEL: El objeto LABEL puede ser usado bajo otros objetos (Ejemplo: El objeto SCALEBAR).
- PROJECTION: Para definir la proyección de los mapas es necesario especificar dos objetos PROJECTION, uno en el objeto MAP para la generación de la imagen de salida y otro para cada capa en el objeto LAYER. Cada capa puede tener un sistema de referencia diferente y el servidor de mapas se encargará de proyectarla al sistema especificado, teniendo en cuenta que MapServer utiliza la librería PROJ4 "*Geographic Projection Library*" para tal fin, el sistema de referencia y proyección pueden ser definido de dos maneras, una es especificando los parámetros de la proyección y otra utilizando la codificación del *European Petroleum Survey.*

Ejemplo de definición de UTM zona 15, NAD83:

```
PROJECTION
"proj=utm"
"ellps=GRS80"
"zone=15"
"north"
"no_defs"
END
```

Para definir coordenadas Geográficas:

```
PROJECTION
 "proj = latlong"
END
```

Utilizando la codificación del *European Petroleum Survey Group* (EPSGP):

```
PROJECTION
 "init = epsg: 23030"
END
```

- SCALEBAR: Esta sección define cómo se construirá la escala gráfica. Comienza con la palabra SCALEBAR y termina con END.

  Propiedades contenidas dentro de la etiqueta: STYLE, STATUS, SIZE, COLOR, UNITS, INTERVALS, TRANSPARENT, POSITION, BACKGROUNDCOLOR, IMAGECOLOR, OUTLINECOLOR.

- REFERENCE: Define cómo será creado el mapa de referencia. Este es un mapa que comprende la extensión total de la zona que incluirá el servicio de WMS, sobre él se representará una marca en la zona que se visualiza actualmente, actualizándose interactivamente. También es posible realizar un clic en un determinado sector del mapa de referencia y MapServer generará el mapa de dicha zona. En las consultas puede generar se un

mapa de referencia, resaltándose en el mismo el punto (x, y), la zona geográfica o la entidad consultada.

Propiedades contenidas dentro de la etiqueta: IMAGE, EXTENT, SIZE, STATUS MARKER, MARKERSIZE, MINBOXSIZE, COLOR, OUTLINECOLOR.

- WEB: Este objeto define cómo operará la interface Web.

  Propiedades contenidas dentro de la etiqueta: HEADER, TEMPLATE, FOOTER, MINSCALE, MAXSCALE, IMAGEPATH, IMAGEURL, EMPTY URL.

  Contiene objetos como: METADATA

- METADATA: Deberá ser incluido tanto en el objeto MAP, como en cada LAYER. En el primer caso contendrá metadatos en general del servicio, y en el segundo caso, metadatos específicos para cada capa de información. Luego el servidor WMS/WFS se basará en estos metadatos para confeccionar el archivo de capacidades.

  -wms_srs: Lista de espacio delimitado de proyección de códigos EPSG soportados por el servidor remoto, normalmente la recibe desde las capacidades de salida del servidor. Este valor debe ser en mayúsculas para evitar problemas en caso de que exista una plataforma sensible y se utiliza para fijar el parámetro SRS WMS URL.

  -wms_name: Lista de capas separadas por comas que se descargan del servidor remoto.

  WMS. Este valor se utiliza para configurar los parámetros de las capas y de las URL de las QUERY_LAYERS WMS.

  -wms_server_version: La versión del protocolo WMS apoyado por el servidor remoto.

  WMS y que será utilizado para la emisión de solicitudes GetMap.

  -wms_format: Formato de imagen para su uso en las solicitudes GetMap. Si se proporciona wms_formatlist entonces wms_format es opcional y MapServer cogerá el primer formato soportado en wms_formatlist para su uso en las solicitudes GetMap. Si tanto wms_format y wms_formatlist se proporcionan, wms_format tiene prioridad. Los Servidores WMS sólo soportan formatos que forman parte de las bibliotecas GD / GDAL.

- QUERYMAP: Define un mecanismo para asignar los resultados de una consulta.

  Propiedades contenidas dentro de la etiqueta: COLOR, STATUS, SIZE, STYLE.

1.1.4 Biblioteca Mapscript

La biblioteca Mapscript de PHP constituye la vía de comunicación de las aplicaciones SIG con el servidor de mapas MapServer y por otra parte rompen en cierta medida ésta rigidez de la representación de mapas a través de los ficheros ".map" permitiendo modificar el Mapfile en tiempo de ejecución, a los cuales se les conoce como Mapfile dinámicos, facilitando de esta forma la creación de aplicaciones con un grado de personalización mayor, eventualmente no alcanzado con aplicaciones del MapServer en modo CGI. **(6)**

1.2 Proceso de Compilación

Desde el surgimiento de la computadora estas han operado sobre bits (ceros y unos) y registros, por lo existe una marcada diferencia entre el lenguaje del hombre y la máquina, imposibilitando así la comunicación entre estos. Esta dificultad dio origen a los lenguajes de programación, los cuales son el vehículo de comunicación entre el hombre y la máquina.

La función principal de los lenguajes de programación es permitir a los programadores especificar de manera precisa: sobre qué datos una computadora debe operar, cómo deben ser almacenados y transmitidos estos, y qué acciones debe tomar bajo una variada gama de circunstancias. A estas especificaciones es a lo que se llama programa.

No solo basta tener estos lenguajes y los programas desarrollados en ellos para lograr una completa comunicación entre el hombre y la computadora, sino también se necesita un mediador que haga contrastar la realización de la misma. Un programa fuente escrito en un lenguaje de programación necesita pasar por un proceso de conversión al lenguaje de máquina que entiende la computadora, a ese proceso de conversión del programa fuente al lenguaje o código de máquina se le llama proceso de compilación o simplemente compilación. Y al programa capaz de realizar ese proceso se le llama compilador o traductor.

¿Qué es un Traductor?

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 **(7)**. El término de traductor engloba tanto a los compiladores como a los intérpretes.

¿Qué es un 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 (típicamente lenguaje máquina) **(7).**

¿Qué es un Intérprete?

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**. (7)**

Ventajas de Compilar frente a interpretar:

- Se compila una vez, se ejecuta $\boldsymbol{n}$ veces.
- En bucles, la compilación genera código equivalente al bucle, pero interpretándolo se traduce tantas veces una línea como veces se repite el bucle.
- El compilador tiene una visión global del programa, por lo que la información de mensajes de error es más detallada.

Ventajas del intérprete frente al compilador:

- Un intérprete necesita menos memoria que un compilador. En principio eran más abundantes dado que los ordenadores tenían poca memoria.
- Permiten una mayor interactividad con el código en tiempo de desarrollo.

1.2.1 Definiciones

Gramática

Es un juego de reglas formalizadas con precisión matemática que genera, sin necesidad de ninguna otra información que no esté representada en el sistema, las oraciones gramaticales del lenguaje que describe o caracteriza y asigna a cada oración una descripción estructural o análisis gramatical. **(8)**

Matemáticamente se define como un conjunto que consta de cuatro fases:

-El alfabeto, $\sum$
-Los productos no terminales, $\boldsymbol{N}$
-Las producciones, $\boldsymbol{P}$
-Los símbolos objetivos o metas (Terminales) $\boldsymbol{S}$

Los tres primeros de estos símbolos representan conjuntos matemáticos y el cuarto identifica a un elemento particular del segundo conjunto: $(\sum, \mathbf{N}, \mathbf{P}, \mathbf{S})$

Alfabeto ($\sum$)

El alfabeto o conjunto de terminales (sin posibilidad de mayor modificación) es un conjunto finito que consiste en todos los caracteres o símbolos de entrada que pueden ordenarse para formar oraciones en un lenguaje.

No terminales (N)

Es un conjunto finito que no está incluido en el alfabeto, pero que cuyos símbolos pueden interpretarse como los que representan al conjunto de cadenas que son subconjunto de $\sum^*$.

Un símbolo no terminal, el símbolo objetivo, representa exactamente a todas las cadenas del lenguaje. Se usa aquí al alfabeto en mayúscula para representar a los símbolos no terminales. En lenguajes reales, los identificadores que sugieren los significados asociados a los no terminales son preferidos como símbolos no terminales (else, if, while). El conjunto de terminales y no terminales forman el vocabulario de la gramática.

Producciones

La producción, $\boldsymbol{P}$, de una gramática es el conjunto de reglas de reescritura, cada una de ellas representada por dos cadenas de símbolos separadas por una flecha. Los símbolos de cada lado de la flecha pueden construirse de terminales y no terminales sujetos a ciertas restricciones de acuerdo a la gramática. Los lenguajes más complejos requieren de menos restricciones para definirlos.

Terminales

Los símbolos terminales pueden agruparse en cadenas de caracteres de manera arbitraria de acuerdo a las reglas de la gramática en cuestión.

Tipos de Gramáticas. Clasificación de Noam Chomsky

- Si cada producción en $\boldsymbol{P}$ es de la forma $\boldsymbol{A \longrightarrow xB}$ ó $\boldsymbol{A \rightarrow x}$ con $\boldsymbol{A, B \in N}$ y $\boldsymbol{x \epsilon \sum^*}$ o de la forma $\boldsymbol{A \rightarrow Bx}$ ó $\boldsymbol{A \rightarrow x}$ con $\boldsymbol{A, B \in N}$ y $\boldsymbol{x \in \sum^*}$ entonces la gramática $\boldsymbol{G}$ se denomina Regular Derecha o Regular a la izquierda respectivamente, o de forma general Gramática Regular.
- Si cada producción en $\boldsymbol{P}$ es de la forma $\boldsymbol{A \rightarrow \alpha}$ donde $\boldsymbol{A \in N}$ y $\boldsymbol{\alpha \in (N \cup \sum) * N}$ entonces la gramática $\boldsymbol{G}$ se denomina de Libre Contexto.
- Si cada producción en $\boldsymbol{P}$ es de la forma $\boldsymbol{\alpha \rightarrow \beta}$ donde $\boldsymbol{\langle \alpha | \leq | \beta \rangle}$ y $\boldsymbol{\alpha}$ contiene al menos un no terminal, entonces la gramática $\boldsymbol{G}$ se denomina Dependiente del Contexto.
- Si una gramática $\boldsymbol{G}$ no cumple las restricciones anteriores se denomina Gramática sin Restricciones.

Autómata

Un autómata consiste en un mecanismo de control con un número de estados finitos y un mecanismo de entrada que puede avanzarse o retrocederse, como una cinta continua, para su lectura y posiblemente escritura. Un alfabeto finito define los símbolos que se puede encontrar dentro de la cinta, que es el mismo alfabeto que se encuentra en la definición del lenguaje. El contenido inicial de la cinta que es una cadena de caracteres que están dentro del alfabeto, es la entrada del autómata. Comenzando por un estado que se denomina estado inicial, el autómata pasa por sus estados internos, basado en lo que encuentra en la cinta y las reglas definidas de antemano. Cada uno de estos pasos se le denomina transición. En cualquier momento el autómata puede detenerse llegando a dos estados posibles:

- La aceptación de la entrada como una secuencia válida de su gramática.
- El bloqueo que consiste en un estado en el cual no cuenta con la regla necesaria para seguir analizando la entrada y esta es rechazada. **(8)**

Léxico (vocabulario): Conjunto de palabras que forman parte de un lenguaje específico.

Sintaxis: Conjunto de reglas necesarias para construir frases correctas en un lenguaje.

Semántica: Significado de frases generadas por la sintaxis y el léxico.

Token o lexema: Unidad básica de un compilador, que puede representar un símbolo o conjunto de símbolos con un significado semántico.

Cadena: Es una secuencia de símbolos de cierto alfabeto colocados uno a continuación del otro.

Concatenación de Cadenas: Si **x** y **y** son cadenas sobre ∑, entonces **xy** es una cadena sobre ∑, que se denomina concatenación de **x** y **y** .

Reverso de una Cadena: El reverso de una cadena **x** se denota **xR**, y es la cadena que se obtiene escribiendo los símbolos de **x** en sentido inverso.

Lenguaje: Un lenguaje sobre ∑ es un conjunto de cadenas sobre dicho alfabeto que obedece una cierta regla de formación.

Clausura de un Alfabeto: El conjunto ∑* denota todas las cadenas sobre ∑, incluyendo la cadena vacía, este conjunto se denomina Clausura de ∑.

El conjunto ∑+ se denomina Clausura Positiva de ∑ y se define como: ∑+ = ∑*- e

1.3 Fases de un compilador

Un compilador opera en fases, cada una de las cuales transforma al programa fuente en una representación útil para la próxima fase, tal como se muestra en la figura siguiente:

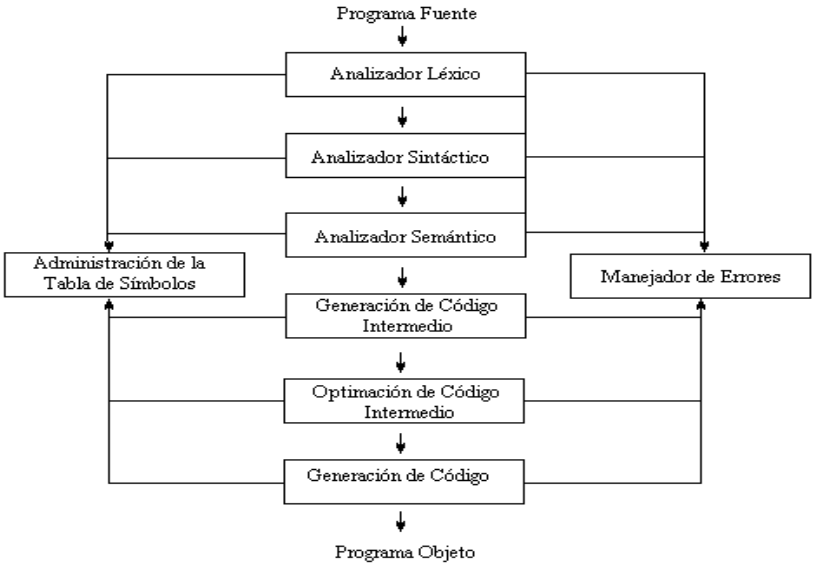


Figura 2: Fases de un Compilador

Las tres primeras partes forman la mayor porción de análisis de un compilador. Como se muestra en la figura el Administrador de la Tabla de Símbolos y el Manejador de Errores están estrechamente relacionados con las diferentes fases del compilador, los cuales interactúan con los mismos a lo largo del proceso de compilación.

1.3.1 Tabla de Símbolos

La tabla de símbolos tiene como función principal guardar en un área de memoria los identificadores utilizados en las expresiones y declaraciones junto con sus atributos. Dentro de estos puede encontrarse el tipo de número, el espaciado asignado, su alcance (donde es válido), y en el caso de procedimientos o rutinas las variables de argumento, el método de acceder a los argumentos (por valor o referencia) el tipo de valor resultante en caso de una función etc. **(9)**

Una tabla de símbolo se forma básicamente de una estructura de datos donde cada registro representa un identificador y sus campos los atributos del identificador .El analizador léxico sitúa los identificadores en la tabla pero usualmente no puede saber sus atributos.

Las fases siguientes del compilador agregan información a los identificadores en la tabla de símbolos usando esta información de forma variada .Por ejemplo, se puede agregar de qué tipo de identificador se trata y asegurarse que el programa fuente los use de forma válida a ese lenguaje.

1.3.2 Manejador de Errores

En las fases de compilación es muy usual que se encuentren errores, los cuales deben manejarse de forma tal que se pueda seguir a la próxima fase sin tener que interrumpir el proceso, para poder identificar otros errores en las fases posteriores.

La mayoría de los errores son detectados en las fases de análisis sintáctico y semántico:

- El análisis sintáctico detecta cuando la cadena de token viola las reglas de estructuras (sintaxis) especificadas en el lenguaje.
- El análisis semántico detecta las construcciones que aunque la semántica esté bien, no respetan las reglas de construcción del lenguaje.
- En el análisis léxico se detectan los errores de cadenas que no forman token válidos en el lenguaje.

1.3.3 Análisis Léxico

El compilador recibe un código de un programa escrito en un lenguaje de programación donde el análisis léxico toma los caracteres de la cadena y los agrupa en fichas en los que cada uno de ellos representa una secuencia de caracteres cohesivo de forma lógica, tales como un identificador, una palabra clave (if, while, do) una marca de puntuación o un operador de varios caracteres. La secuencia de caracteres que forma una ficha se llama lexema o token. No solo se genera una ficha sino que se introduce el lexema a la tabla de símbolos.

Como el analizador léxico es la parte del compilador que lee el texto fuente, también puede realizar ciertas funciones secundarias en la interfaz de usuario, como eliminar del programa fuente comentarios y espacios en blanco en forma de caracteres de espacios caracteres TAB y de línea nueva. Otra función es relacionar los mensajes de error del compilador con el programa fuente. Por ejemplo el analizador léxico puede tener localizado el número de caracteres de nueva línea detectados, de modo que se pueda asociar un número de línea con un mensaje de error.

En la *Figura 3* se muestra el funcionamiento del análisis léxico o scanner.

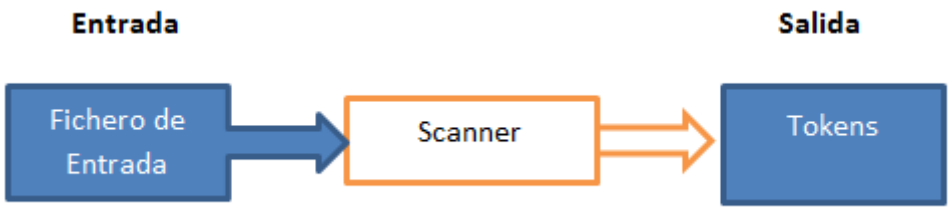


Figura 3: Objetivo del análisis léxico

En la *Tabla 1* se muestran ejemplos donde se pueden ver las diferencias entre los conceptos: token, lexema y patrón.

Tabla 1: Ejemplos de token, lexema y patrón

Token	Lexema	Descripción del Patrón
tk_const	"const"	const
tk_if	"if"	If
tk_les_iq	"<="	<=
tk_id	"edad_paciente"	letra(letra||digito)*
tk_real_num	"2.01"	digito*.digito*
tk_literal_string	"Hola mundo"	"(carácter alfabeto)*"

1.3.4 Análisis Sintáctico

En el análisis sintáctico se comprueba si la secuencia de tokens que representa al texto de entrada, en base a una gramática dada es correcta. En caso de que el programa de entrada sea válido, suministra el árbol sintáctico que lo reconoce en base a una representación computacional. Este árbol es el punto de partida de la fase posterior de la etapa de análisis: el analizador semántico. Ver *Figura 4*

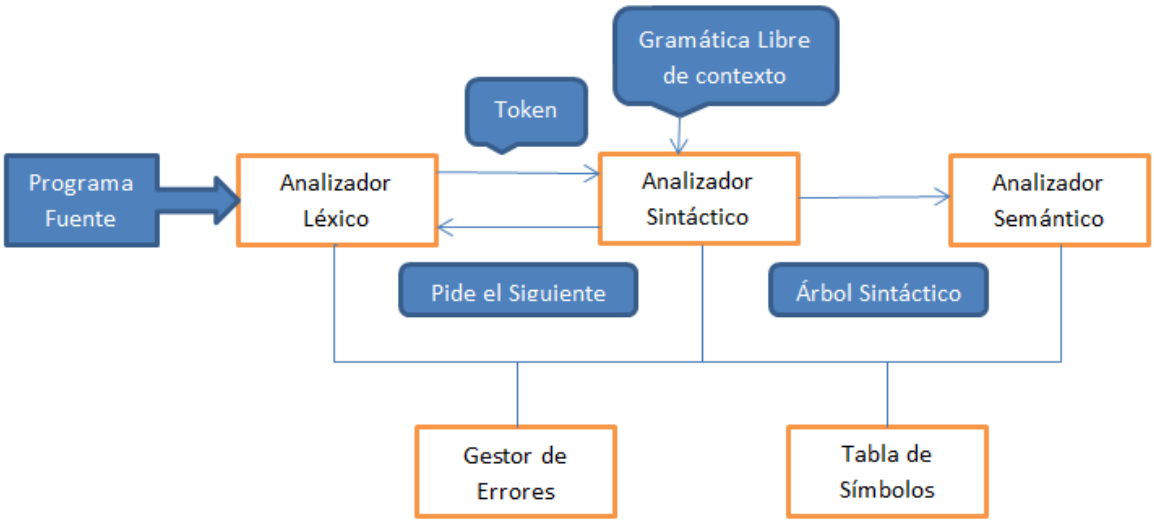


Figura 4: Interacciones del analizador sintáctico

El analizador sintáctico impone una estructura jerárquica en la cadena de token que se representa por un árbol en forma esquemática o una lista ligada en forma interna en una computadora.

Otras funciones del análisis sintáctico:

- Controla el flujo de tokens reconocidos por parte del analizador léxico.
- Informa de la naturaleza de los errores sintácticos que encuentra e intenta recuperarse de ellos para continuar la compilación.
- Incorpora acciones semánticas (excepto el analizador léxico): desde el análisis semántico hasta la generación de código.

Árbol de Sintaxis Abstracta (AST)

Los árboles de sintaxis abstracta son el lenguaje común que hablan las distintas fases del proceso de compilación. Cada fase realiza algún procesamiento usando el árbol, lo reescribe o simplemente crea otra estructura de datos antes de pasar el árbol a la siguiente fase. Para realizar sus funciones las fases recorren el árbol y realizan determinadas acciones cuando encuentran algún patrón conocido **(10)**. En la siguiente figura se muestra la representación de un subárbol para la expresión: 3+4*5.

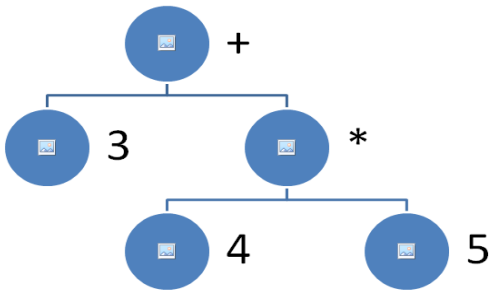


Figura 5: Representación de una expresión

Contienen todos los tokens que aparecen en la cadena de entrada. Sin embargo no todos estos tokens son necesarios en las fases posteriores del proceso de compilación. El objetivo es diseñar una representación intermedia que salga del analizador sintáctico que contenga solo las estructuras útiles a posteriores fases.

1.3.5 Análisis Semántico

La función principal del análisis semántico es comprobar la coherencia de las operaciones. Se recoge información que resulta de utilidad para fases posteriores, cuando esta información se almacena en el árbol de sintaxis abstracta, este se denomina AST "decorado". En este nuevo árbol se almacena la información necesaria para en un recorrido posterior del AST poder chequear la compatibilidad de tipos en las operaciones entre las variables. La comprobación de tipos constituye en esta fase la función primordial, aparejado a esta se encuentra el proceso de interacción con la tabla de símbolos

Esta fase es la encargada de detectar la validez semántica de las sentencias aceptadas por el analizador sintáctico. Tiene como entrada el árbol de sintaxis generado por el analizador sintáctico, y su salida se suele utilizar como entrada para la generación de código. La estructura de datos empleada para intercambiar información entre las dos fases mencionadas es un árbol de sintaxis abstracta decorado.

Este árbol posee información adicional al árbol generado por el analizador sintáctico, como por ejemplo la información relativa al tipo de cada una de las expresiones del programa. El empleo de dicha información es útil para llevar a cabo el proceso de generación de código (a bajo nivel, el tipo de una expresión es necesario, por ejemplo, para saber el número de bytes que ocupa su valor).

1.3.6 Generación de Código Intermedio

En la generación de código intermedio se genera una representación explícita del programa fuente en un código intermedio específico del compilador, transformando un árbol de análisis sintáctico (semántico) en una representación de un lenguaje intermedio, que suele ser código suficientemente sencillo para poder luego generar código máquina. Este lenguaje permite construir en mucho menos tiempo un compilador para otra máquina y también permite construir compiladores para otros lenguajes fuente generando códigos para la misma máquina. Consta de dos características importantes:

- Debe ser fácil de producir
- Debe ser fácil de traducir al código objeto.

Si el código objeto será el ensamblador es entonces importante tener un lenguaje intermedio que use un esquema de "código de tres direcciones" donde cada localidad de memoria funciona como un registro.

Algunos compiladores generan una representación intermedia explícita del programa fuente, también llamada forma interna. Existen varios tipos de forma interna, los cuales son considerados como un programa para una máquina abstracta.

Tipos de Forma Interna

- Formas Intermedias de Alto Nivel: son aquellas que se suelen emplear en las primeras fases de análisis.
- Formas Intermedias de Nivel Medio: son válidas para representar un conjunto amplio de lenguajes fuente, no siendo dependientes de uno en concreto. Válidas para representar un conjunto extenso de arquitecturas de hardware.
- Formas Intermedias de Bajo Nivel: permiten traducir a distintos micros de una misma arquitectura, creando una dependencia a ésta.
- Formas Intermedias Multinivel: Aquéllas que conjugan varias de las características anteriores.

1.3.7 Optimización del Código

Consiste en perfeccionar el código intermedio para obtener un programa que ejecute más rápido. Debido a su importancia, gran parte del tiempo del compilador es usado en esta fase.

1.3.8 Generador de Código

Es la fase final del compilador y consiste en generar el programa objeto relocalizable, usualmente ensamblador o lenguaje de máquina. Aquí, cada instrucción del código mejorado es traducida a su equivalente en el lenguaje objeto. Un aspecto crucial es la asignación de variables a los registros.

1.4 Herramientas de apoyo en la confección de compiladores

El uso y perfeccionamiento de los compiladores ha traído consigo el desarrollo de herramientas que aportan a la realización de los mismos, estas se han ido especializando en las diferentes fases del proceso de compilación, brindándole a los desarrolladores una serie de facilidades a la hora de diseñar e implementar un compilador.

En el mundo existen diversas herramientas de apoyo de este tipo, desarrolladas en diferentes lenguajes de programación, las cuales responden a los intereses de los múltiples sistemas operativos. Entre las herramientas más utilizadas se pueden encontrar el Flex, Yacc, Lex, Bison entre otras. Ver *Tabla2*

Tabla 2: Herramientas para la construcción de compiladores

Herramienta	Descripción
Bison	Generador de Analizadores Sintácticos Ascendentes tipo YACC
COCO/R	Generador de Analizadores Léxicos y Sintácticos Descendentes Recursivos
Flex	Generador de Analizadores Léxicos tipo Lex
Lex	Generador de Analizadores Léxicos
SDGLL1	Sistema Detector de Gramáticas LL(1)
TS 2006	Tipo abstracto de datos Tabla de Símbolos de uso sencillo (beta 0.4)

TS	Tipo abstracto de datos Tabla de Símbolos
TS-OO	Tipo abstracto de datos Tabla de Símbolos
YACC	Generador de Analizadores Sintácticos Ascendentes LR(1)

Bison: Es un generador de analizadores sintácticos de propósito general que convierte una descripción gramatical para una gramática independiente del contexto en un programa en C que analice esa gramática. Es utilizado en un amplio rango de analizadores de lenguajes, desde aquellos usados en simples calculadoras de escritorio hasta complejos lenguajes de programación.

Lex: Es un generador de analizador léxico, que sirve para generar los token para la siguiente fase. La principal característica de Lex es que va a permitir asociar acciones descritas en C, a la localización de las Expresiones Regulares que se hayan definido. Para ello Lex se apoya en una plantilla que recibe como parámetro, y que se debe diseñar con cuidado.

Internamente Lex va a actuar como un autómata que localizará las expresiones regulares que se le describan, y una vez reconocida la cadena representada por dicha expresión regular, ejecutará el código asociado a esa regla.

Yacc: Es un programa para generar analizadores sintácticos. Las siglas del nombre significan *Yet Another Compiler Compiler*, es decir, "Otro generador de compiladores más". Genera un analizador sintáctico (la parte de un compilador que comprueba que la estructura del código fuente se ajusta a la especificación sintáctica del lenguaje) basado en una gramática analítica. Yacc genera el código para el analizador sintáctico en el Lenguaje de programación C.

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.

Estas herramientas de apoyo han sido reescritas para otros lenguajes, incluyendo Ratfor, EFL, ML, Ada, Java, Python, y Limbo. De esta forma se ha logrado una mayor utilización de las mismas en diferentes compiladores desarrollados sobre tecnologías libres.

Teniendo en cuenta las características de las aplicaciones antes mencionadas, se ha escogido para la realización del compilador las herramientas Yacc y Lex. En muchos de los compiladores desarrollados

en el mundo suelen ser utilizados juntos. Yacc utiliza una gramática formal para analizar un flujo de entradas, algo que Lex no puede hacer con expresiones regulares simples (Lex se limita a los autómatas de estados finitos simples). Sin embargo, Yacc no puede leer en un flujo de entradas simple, requiere una serie de símbolos. Lex se utiliza a menudo para proporcionar a Yacc estos símbolos.

1.5 Conclusiones parciales

En la elaboración de este capítulo se abordaron conceptos generales que sirvieron de guía para lograr un mejor entendimiento de esta investigación. La realización de una caracterización de los procesos de compilación, permitió precisar qué tipo aplicación (compilador) usar. Se lo logró definir las herramientas con las que se diseñará e implementará la aplicación, permitiendo demostrar la importancia del compilador a la hora de importar el fichero Mapfile. El estudio y análisis de las distintas herramientas de apoyo a la construcción de traductores, trajo como resultado concretar cuáles serían las que se utilizarían en el desarrollo del trabajo de diploma.

Capítulo 2: Tendencias y Tecnologías

En este capítulo se presenta una fundamentación de las tecnologías actuales que facilitan el diseño de compiladores en software libre. En él se justifican la utilización de todo el instrumental tecnológico utilizado en el desarrollo de la herramienta de compilación, así como las metodologías y lenguajes utilizados.

2.1 ¿Por qué se utiliza Python como lenguaje de programación?

Python es un lenguaje de programación interpretado, orientado a objetos y de sintaxis sencilla. Se ha implementado sobre muchos sistemas operativos (incluyendo Linux, Windows, VMS, AmigaOS...) y para los entornos más variados (incluyendo .NET de Microsoft y Java, de Sun). **(11)**

Python presenta una serie de ventajas que lo hacen muy atractivo, tanto para su uso profesional como para el aprendizaje de la programación. Entre las más interesantes características se tienen:

- Es un lenguaje muy expresivo, es decir, los programas Python son muy compactos: un programa Python suele ser bastante más corto que su equivalente en lenguajes como C. (Python llega a ser considerado por muchos un lenguaje de programación de muy alto nivel).
- Es muy legible. La sintaxis de Python es muy elegante y permite la escritura de programas cuya lectura resulta más fácil que si utilizáramos otros lenguajes de programación.
- Ofrece un entorno interactivo que facilita la realización de pruebas y ayuda a despejar dudas acerca de ciertas características del lenguaje. El entorno de ejecución de Python detecta muchos de los errores de programación que escapan al control de los compiladores y proporciona información muy rica para detectarlos y corregirlos.
- Puede usarse como lenguaje imperativo procedimental o como lenguaje orientado a objetos.
- Posee un rico juego de estructuras de datos de alto nivel que se pueden manipular de modo sencillo, como matrices flexibles y diccionarios, que llevarían días de programación en C.
- Permite dividir un programa en módulos reutilizables desde otros programas. Viene con una gran colección de módulos estándar que pueden ser utilizados como base para nuevos programas (o como ejemplos para empezar a aprender Python). También hay módulos

incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a lGU (interfaz gráfica con el usuario) como Tk.

- Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa de la base hacia arriba.

2.2 Ply como analizador léxico–sintáctico.

Se escoge Ply porque es la implementación para Python de Lex y Yacc, dos herramientas para apoyar el desarrollo de compiladores (generan lexer y parser dependiendo de ciertos parámetros: expresiones regulares para Lex, y gramática para Yacc).

Ply consta de dos módulos separados; lex.py y yacc.py, que se encuentran en un paquete denominado ply. El objetivo principal de Ply es permanecer fiel a la forma tradicional de las herramientas de trabajo Lex / Yacc. Esto incluye el apoyo a LALR, así como analizar el suministro de una amplia entrada de validación, reporte de errores, y diagnósticos.

El módulo lex.py se utiliza para convertir la entrada de texto en una colección de fichas especificado por un conjunto de reglas de expresiones regulares. Yacc.py se utiliza para reconocer la sintaxis de lenguaje que se ha especificado en la forma de una gramática libre de contexto. Las dos herramientas están diseñadas para trabajar en conjunto.

Algunas características:

- Es totalmente implementado en Python.
- Utiliza LR, es razonablemente eficiente y muy adecuado para el concepto más amplio de gramáticas.
- Sencillo de utilizar y proporciona muy extensa comprobación de errores.

- Proporciona la mayor parte de la norma Lex / Yacc incluyendo características que presentan el apoyo a las producciones vacías, reglas de prioridad, recuperación de errores y el apoyo a las gramáticas ambiguas.

2.3 Porqué Wing IDE como entorno de desarrollo para la programación de la Herramienta de Compilación?

Wing IDE es un entorno integrado de desarrollo especialmente diseñado para el lenguaje de programación Python. Permite el desarrollo rápido de aplicaciones de plataforma cruzada para escritorio, web y empresariales. Wing IDE se enfoca en incrementar la productividad y la calidad del código, especialmente en proyectos complejos con requerimientos cambiantes.

Entre las principales características de Wing IDE se encuentran:

- Auto completado de código.
- Asistente de codificación.
- Depurador avanzado con soporte para Zope y Plone.
- Búsquedas en múltiples archivos.
- Búsquedas en todo el disco.
- Búsquedas con expresiones regulares.
- Atajos de tecla de Visual Studio, VI y Emacs.
- Soporte de CVS, Subversión y Perforce.
- Sintaxis coloreada.

2.4 El Lenguaje Unificado de Modelado (UML) como soporte a la programación orientada a objetos.

El Lenguaje Unificado de Modelado (UML) es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. Captura decisiones y conocimiento sobre los sistemas que se deben construir. Se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas. Está pensado para usarse con todos los métodos de desarrollo, etapas del ciclo de vida, dominios de aplicación y medios.

El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. Está pensado para ser utilizado en herramientas interactivas de modelado visual que tengan generadores de código así como generadores de informes. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos. **(12)**

UML también contiene construcciones organizativas para agrupar los modelos en paquetes, lo que permite a los equipos de software dividir grandes sistemas en piezas de trabajo, para entender y controlar las dependencias entre paquetes, y para gestionar las versiones de las unidades del modelo, en un entorno de desarrollo complejo. Contiene construcciones para representar decisiones de implementación y para elementos de tiempo de ejecución en componentes. **(12)**

UML permite modelar sistemas de información, y su objetivo es lograr modelos que, además de describir con cierto grado de formalismo tales sistemas, puedan ser entendidos por los clientes o usuarios de aquello que se modela. Para ello, es muy importante que el idioma en el que estén las palabras y textos que aparezcan en tales modelos sea el propio de estas personas.

Sin embargo, desde el punto de vista puramente tecnológico, UML tiene una gran cantidad de propiedades que han sido las que, realmente, han contribuido a hacer de UML el estándar de guía de la industria del software. Algunas de las propiedades de UML como lenguaje de modelado estándar son:

- Concurrencia: es un lenguaje distribuido y adecuado a las necesidades de conectividad actual y futura.
- Ampliamente utilizado por la industria desde su adopción por OMG.5
- Reemplaza a decenas de notaciones empleadas con otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soportan tienen su fundamento en las tecnologías orientadas a objetos, tales como objetos, clase, componentes y nodos.

[5] OMG acrónimo de Object Management Group

- Emplea operaciones abstractas como guía para variaciones futuras, añadiendo variables si es necesario.

- Comportamiento del sistema: casos de uso, diagramas de secuencia y de colaboraciones, que sirven para evaluar el estado de las máquinas

2.5 Visual Paradigm para UML

Visual Paradigm para UML (VP-UML) *Enterprise Edition* apoya las últimas notaciones de UML, ingeniería reversa, generación de UML del código, importa desde Rational Rose. Además puede generar código fuente de programación del diagrama de clases de un proyecto. Utiliza técnicas de ingeniería inversa para llevar del código fuente a sus diagramas de clases. De esta forma mantiene un alto nivel de sincronización entre el modelo y el código. Corre sobre las plataformas: Windows (98, 2000, XP, o Vista), Linux, Mac OS X y Solaris. Permite a los desarrolladores de un mismo equipo trabajar sobre un mismo proyecto. **(13)**

Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML.

2.6 El Proceso Unificado de Desarrollo de Software (RUP) como base en el desarrollo de la solución.

RUP como metodología es un “proceso de desarrollo de software que define quién está haciendo qué, cuándo y cómo alcanzar un determinado objetivo’’, además “es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto” **(12).**

Para la realización del producto software la metodología RUP divide el trabajo en 9 flujos de trabajo y 4 fases y tiene como características fundamentales ser iterativo e incremental, dirigido por casos de usos y centrado en la arquitectura.

El ser iterativo e incremental le permite al equipo de desarrollo en un tiempo estimado obtener una pequeña parte del producto de acuerdo a una fase que pase por todas las disciplinas, esto se conoce

por iteración, permite además según cada iteración un crecimiento del producto, conocido como incremento.

Sobre esto recae gran importancia, pues se puede ir observando poco a poco el adelanto que va teniendo el producto que se está desarrollando y que se puedan corregir a tiempo las deficiencias.

La característica de ser dirigido por casos de uso ayuda también al equipo de trabajo a desarrollar el sistema según los requisitos solicitados por el cliente, pues a través de los mismos se modelan los casos de uso. Por último, no por ello menos importante, posee la característica de estar centrado en la arquitectura, esta indica cómo puede construirse el sistema y en qué orden. Todas estas características fueron destacas por los autores del proceso unificado desde sus inicios.

Las disciplinas del RUP según el orden serían las siguientes:

- Modelamiento del Negocio: cuyo objetivo es obtener una breve descripción de la panorámica actual del negocio en el que se va a trabajar e identificar lo que se necesita automatizar.
- Requerimientos: cuyo objetivo es identificar las exigencias que el producto software debe cumplir de forma detallada donde se cumpla lo solicitado por el cliente y además se obtienen los prototipos que tendrá cada funcionalidad en el sistema.
- Análisis y Diseño: cuyo objetivo es indicar lo que hay que programar según lo especificado en los requerimientos, además se desarrolla la arquitectura del software.
- Implementación: cuyo objetivo es desarrollar el producto software mediante una organización del código en capas, además de integrar lo desarrollado por todo el equipo en un fichero ejecutable.
- Prueba: cuyo objetivo es encontrar las deficiencias tanto en el código del software como en la documentación, esto tiene que corresponder. Además, debe validar que se cumplan los requisitos planteados por el cliente y como resultado de esta disciplina se obtiene el producto final.
- Despliegue: cuyo objetivo es entregar el producto software para que se proceda a su instalación.

- Administración y Configuración de Cambios: cuyo objetivo es que se pueda tener control y administración de todas las actividades, artefactos y código que se esté desarrollando por cada integrante del equipo.
- Administración del Proyecto: cuyo objetivo es administrar el personal del equipo, su capacitación, planificar el desarrollo del proyecto, controlar los posibles riesgos que puedan aparecer en el desarrollo del software.
- Ambiente: cuyo objetivo es configurar el proyecto de acuerdo a sus especificaciones, tanto en herramientas como en actividades.

Las Fases del RUP son las siguientes**:**

- Inicio: esta fase tiene como objetivo obtener toda la información del negocio del proyecto con sus exigencias más importantes y de ahí preparar y planificar los casos de uso, establecer el costo del proyecto, esquematizar la arquitectura posible y preparar el ambiente del proyecto.
- Elaboración: esta fase tiene como objetivo definir la arquitectura del proyecto, desarrollar los casos de usos según las prioridades desarrollando casi totalmente el sistema.
- Construcción: esta fase tiene como objetivo optimizar el proceso para obtener el software que cumpla lo requerido por el cliente.
- Transición: esta fase tiene como objetivo entregar el material de soporte del producto y el software al cliente para su posterior utilización.

2.7 Conclusiones Parciales

La adopción de herramientas libres para el desarrollo de un sistema informático que compile un fichero de mapas, permite no depender de sistemas que impongan licencias para su uso. Adoptar RUP como metodología de desarrollo de software proporciona que los artefactos generados permanezcan debidamente documentados para el posterior análisis por otros miembros del centro GEYSED. Utilizar Ply como herramienta de apoyo en la confección del compilador permitirá un ahorro de tiempo y esfuerzo al contar con una gran parte de las fases del análisis léxico y sintáctico ya implementadas.

Capítulo 3: Descripción de la Solución Propuesta

En el presente capítulo se aborda todo lo referente al modelo de dominio en el cual se analizan todas las entidades y conceptos presentes en el entorno donde se aplica el compilador. Se especifican detalladamente los requisitos funcionales y los no funcionales que cumple dicho sistema y se describen los actores y los casos de usos del sistema.

3.1 Modelo de Dominio

El Modelo de Dominio o Modelo Conceptual, permite de manera visual mostrar al usuario los principales conceptos que se manejan en el dominio del problema. Representa las clases conceptuales del mundo real, no de componentes de software.

Puede considerarse como un diccionario visual de las abstracciones relevantes, vocabulario e información del dominio. Aprovechando las oportunidades de los diagramas UML para representar conceptos, el Modelo de Dominio se presenta en forma de diagrama de clases donde figuran los principales conceptos y roles del sistema en cuestión.

Teniendo en cuenta que no se tienen bien definidos los procesos del negocio se realizará una modelación del dominio *(Figura 6)* y se procederá a explicar cada uno de los conceptos que forman parte del mismo. Todo ello para tener una mejor comprensión de la estructura y dinámica del sistema, los problemas actuales dentro de este e identificar las mejoras potenciales.

3.1.1 Diagrama de clases del Modelo de Dominio

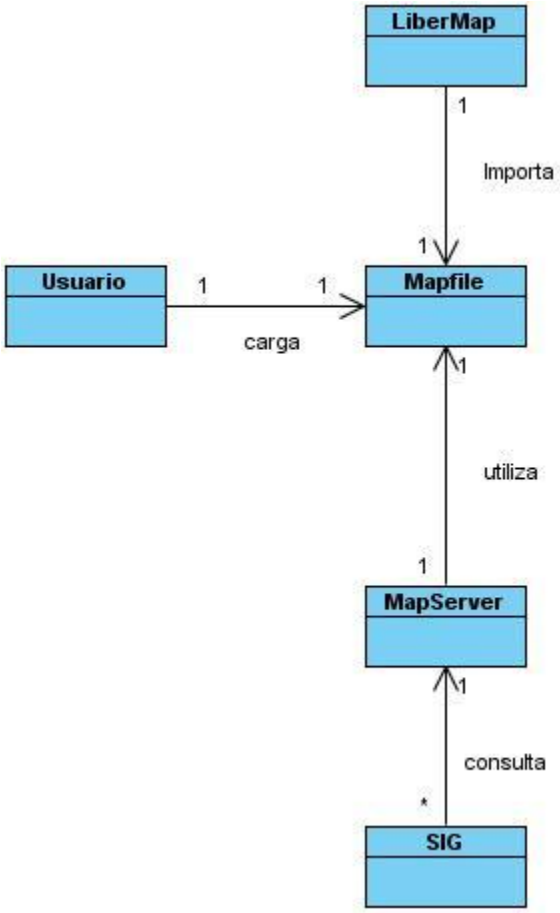


Figura 6: Diagrama del Modelo de Dominio

3.1.2 Glosario de Términos del Dominio

SIG: Sistema de Información Geográfica.

MapServer

Servidor de Mapas que es consultado por los SIG para visualizar mapas.

Mapfile

Fichero que constituye el corazón del servidor de mapas MapServer con extensión .map, el cual agrupa los objetos que componen un mapa y sobre el que MapServer basa su configuración.

LiberMap

Catálogo de mapas que se utiliza para la edición dinámica del Mapfile.

Usuario

Persona que interactúa con el sistema.

3.2 Levantamiento de Requisitos

La *IEEE Standard Glossary of Software Engineering Terminology* define un requerimiento con las siguientes definiciones:

- Condición o capacidad que necesita un usuario para resolver un problema o lograr un objetivo.
- Condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente.

Todas las ideas que tengan los clientes, usuarios y miembros del equipo de proyecto acerca de lo que debe hacer el sistema, deben ser analizadas como candidatas a requisitos.

La validación de requisitos es una actividad muy importante, pues un levantamiento de requisitos con errores que no se detecten a tiempo, además de no conducir a resultados inesperados, provoca costos excesivos y gran pérdida de tiempo.

3.2.1 Requisitos funcionales del sistema

El sistema debe ser capaz de:

RF1 - Cargar fichero: El sistema debe permitir que el usuario pueda cargar un fichero.

RF2 - Reconocer estructuras: El sistema debe ser capaz de reconocer las diferentes estructuras por las que está conformado el archivo Mapfile (MAP, LAYER, REFERENCE, METADATA, PROJECTION, etc.)

RF3 - Almacenar Estructuras: El sistema debe ser capaz de propiciar un mecanismo para almacenar las diferentes estructuras del archivo Mapfile.

RF4 - Informar errores: El sistema debe ser capaz de informar los diferentes errores que puedan presentarse en el compilador (Errores, Lexicográficos, Errores Semánticas, Errores Sintácticos, Errores Lógicos).

3.2.2 Requisitos no funcionales del sistema

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades son las características que hacen al producto atractivo, usable, rápido o confiable.

Usabilidad

El sistema podrá ser usado por personas con conocimientos básicos en el manejo de LiberMap.

Confiabilidad

La herramienta de implementación a utilizar debe tener soporte para recuperación ante fallos y errores**.**

Apariencia o interfaz externa

Debe brindar una interfaz amigable, interactiva, intuitiva y de fácil comprensión para el usuario, facilitando en todo momento la interacción de este con el sistema.

Portabilidad y operatividad.

El sistema debe ser compatible con los Sistemas Operativos: Windows y GNU/Linux.

Seguridad

El sistema debe de ser capaz de recuperarse ante cualquier falla para así estar disponible siempre que sea necesario.

Hardware

Las computadoras que utilizarán el sistema deben tener:

-256 MB de memoria RAM como mínimo

-Python v2.5 o una versión superior

Rendimiento

El tiempo de respuesta para visualizar la interfaz en la pantalla será de 1 segundo, y para procesar, actualizar y compilar información de 2 a 4 segundos, en dependencia de la cantidad de información que se encuentre dentro del Mapfile.

3.3 Descripción del Sistema Propuesto

Para complementar los objetivos propuestos al inicio de este trabajo, y teniendo en cuenta todos los requerimientos planteados, el sistema que se propone debe tener como funcionalidad principal importar el fichero Mapfile. Con el sistema interactuará un usuario que será el encargado de cargar el Mapfile para su posterior compilación. La propuesta de solución se divide en cuatro paquetes, en los cuales se agruparán los módulos necesarios para la implementación de las distintas funcionalidades.

3.3.1 Descripción de los actores del sistema

Utilizando las facilidades que brinda el UML, se representarán los requisitos funcionales del sistema mediante un diagrama de casos de uso. Para ello hay que definir de acuerdo a lo planteado en los epígrafes anteriores, cuáles serían los actores que van a interactuar con el sistema, y los casos de uso que van a representar las funcionalidades

Un actor del sistema no es parte del sistema en desarrollo, es un agente externo que intercambia información con el mismo en pos de obtener un resultado esperado. Este sistema cuenta con el siguiente actor:

Tabla 3: Actores del sistema

Actor	Descripción
Usuario	Es la persona que se encarga de subir un fichero a través del sistema

3.3.2 Diagrama de Casos de Uso del Sistema

Un caso de uso es una unidad coherente de funcionalidad, externamente visible, proporcionada por una unidad del sistema y expresada por secuencias de mensajes intercambiados por la unidad del sistema y uno o más actores. El propósito de un caso de uso es definir una pieza de comportamiento coherente, sin revelar la estructura interna del sistema.

La definición de un caso de uso incluye todo el comportamiento que implica: las líneas principales, las diferentes variaciones sobre el comportamiento normal, y todas las condiciones excepcionales, que pueden ocurrir con tal comportamiento, junto con la respuesta deseada. Desde el punto de vista de los usuarios, éstas pueden ser situaciones anormales. (*Figura 7*)

.

Figura 7: Diagrama de Casos de Uso del Sistema

La herramienta de compilación a implementar constituirá un módulo más del catálogo de mapas Libermap por lo que en la siguiente figura se muestra cómo quedará estructurado el Diagrama de Casos de Uso del Sistema de LiberMap al incluir los casos de usos del compilador.

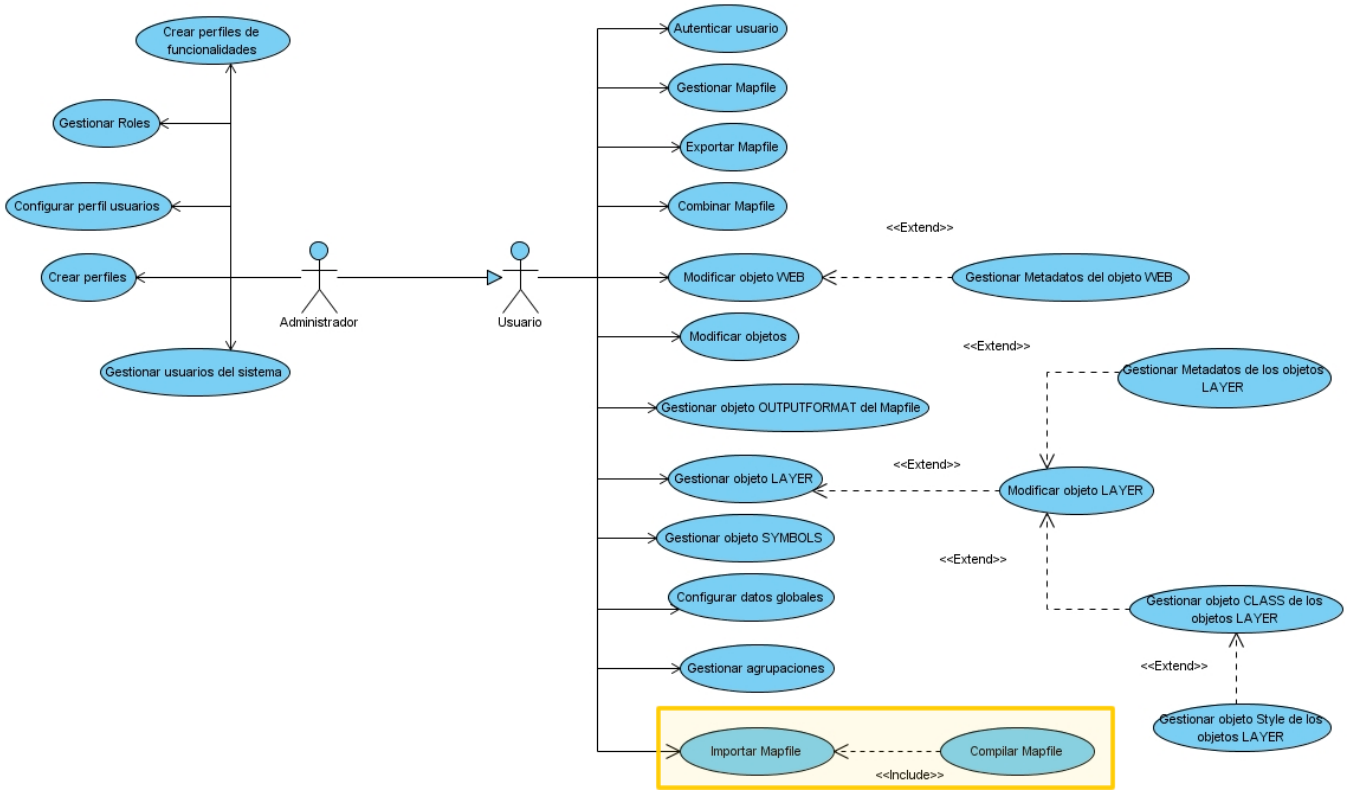


Figura 8: Diagrama de Casos de Uso del Sistema de Libermap

3.3.3 Descripción Textual de los Casos de Uso del Sistema

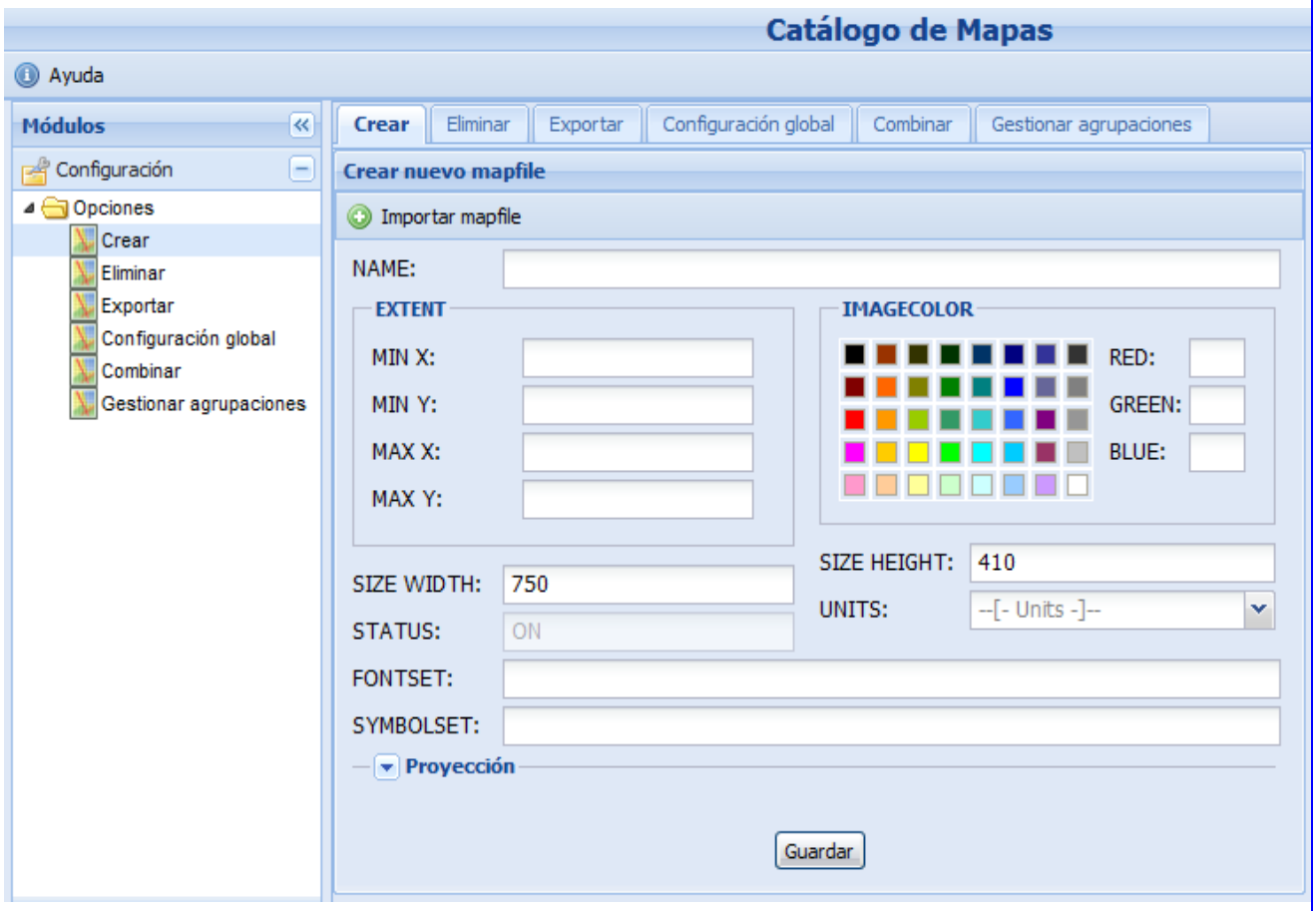
A continuación se presentan los casos de uso determinados para satisfacer los requerimientos funcionales del sistema (*Tabla 4 y 5*).

Casos de Usos del Sistema.

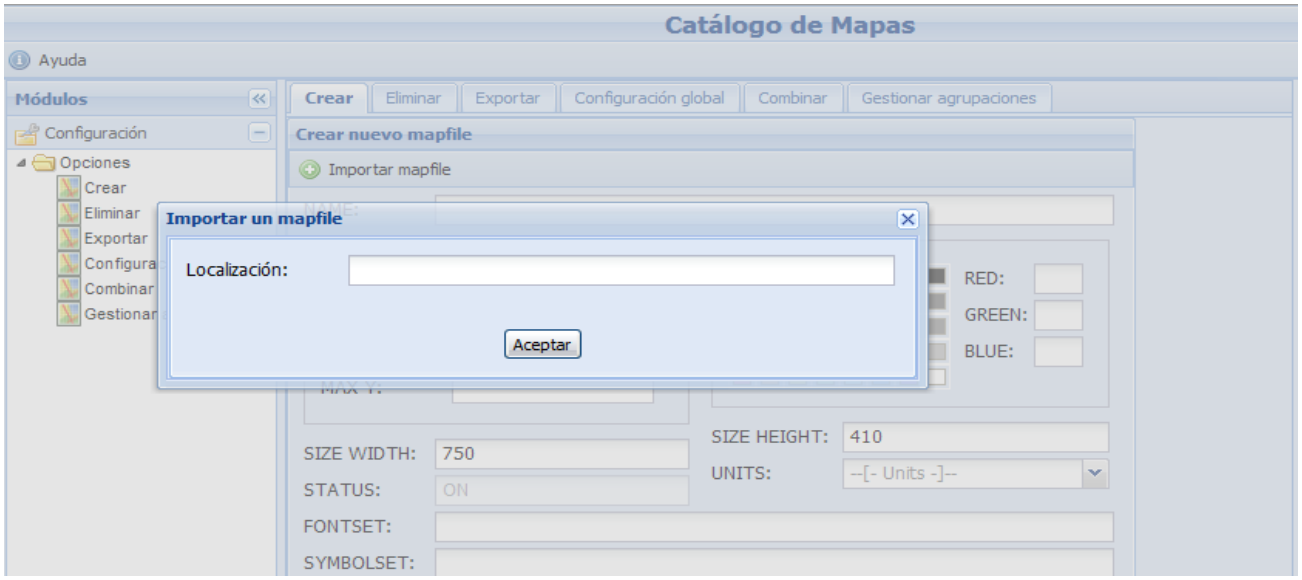
Tabla 4: Descripción del Caso de Uso Importar Mapfile

Caso de Uso:	Importar Mapfile.
Actores:	Usuario
Propósito	Importar el Mapfile hacia LiberMap
Resumen:	El caso de uso se inicia cuando el usuario selecciona la opción “Configuración” y dentro de ella la opción de “Importar un Mapfile”, y termina cuando el usuario luego de haber seleccionado la dirección donde se encuentra el archivo, selecciona la opción de “Importar” y se almacena el Mapfile con todos sus componentes de forma persistente en la base de datos.
Precondiciones:	El usuario debe estar autenticado y poseer los privilegios necesarios para acceder a esta opción.
Referencias	RF1, RF2, RF3, RF4
Prioridad	Crítico
Flujo Normal de Eventos	
1. El caso de uso comienza cuando el usuario selecciona la opción “Importar un Mapfile”. (Interfaz 1)	2. El sistema muestra una ventana “Importar un Mapfile”, mostrando un TextBox para entrar la Localización como se muestra en la interfaz 2.
3. El usuario introduce la dirección donde se encuentra el Mapfile a importar.	4. El caso de uso termina cuando el sistema carga el Mapfile desde la dirección y lo almacena en la base de datos.
Prototipo de Interfaz	

Interfaz 1


Catálogo de Mapas
Ayuda
Módulos
Configuración
Opciones
Crear
Eliminar
Exportar
Configuración global
Combinar
Gestionar agrupaciones
Crear
Eliminar
Exportar
Configuración global
Combinar
Gestionar agrupaciones
Crear nuevo mapfile
Importar mapfile
NAME:
EXTENT
MIN X:
MIN Y:
MAX X:
MAX Y:
IMAGECOLOR
RED:
GREEN:
BLUE:
SIZE WIDTH: 750
SIZE HEIGHT: 410
UNITS: --[- Units -]--
STATUS: ON
FONTSET:
SYMBOLSET:
Proyección
Guardar

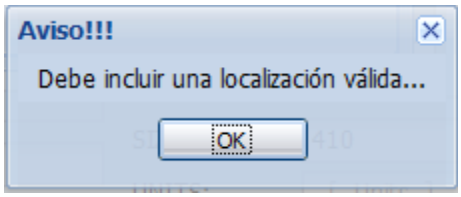
Interfaz 2



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3. El usuario introduce una dirección incorrecta de donde se encuentra el fichero Mapfile.	4. Se muestra una ventana “Aviso” de error como se visualiza en la Interfaz 3.

Prototipo de Interfaz

Interfaz 3



Poscondiciones	Se importa el fichero Mapfile

Tabla 5: Descripción del Caso de Uso Compilar Mapfile

Caso de Uso:	Compilar Mapfile.
Actores:	Usuario
Propósito	Reconocer y almacenar las estructuras contenidas dentro del Mapfile mientras que sean soportadas por el lenguaje, así como gestionar los errores que se encuentren.
Resumen:	El caso de uso se inicia cuando el usuario carga el fichero Mapfile y termina cuando es compilado el Mapfile.
Precondiciones:	El archivo Mapfile debe ser cargado por el usuario
Referencias	RF2, RF3, RF4
Prioridad	Crítico
Flujo Normal de Eventos	
1. El usuario carga el archivo.	2. El sistema compila el fichero Mapfile
Prototipo de Interfaz	
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	3. Si se detectan errores 4.El sistema debe mostrar la información de los errores encontrados.

3.4 Conclusiones Parciales

En este capítulo se desarrolló la propuesta de solución, obteniéndose a partir del análisis del proceso de negocio, el modelo de dominio, el cual permitió conocer con mayor profundidad los conceptos presentes en el entorno donde coexiste el problema. Se determinaron los requisitos funcionales que el sistema debe cumplir y los no funcionales que debe tener. Luego de establecidos los requisitos funcionales, se pudo identificar fácilmente el actor del sistema y los casos de usos referentes a los requisitos, los cuales fueron plasmados en el diagrama de casos de uso del sistema. La realización de la descripción textual de los casos de uso, posibilitó un mejor entendimiento de cómo funciona el sistema.

Capítulo 4: Construcción de la solución propuesta

En este capítulo se visualiza el diseño del compilador que se realizó. En él se encuentran los principios de diseño, una explicación de cómo se realiza la compilación del fichero Mapfile, así como los estándares de programación utilizados. Además se efectúa la validación de la solución propuesta.

4.1 Fase de Análisis de la Propuesta de Solución

Durante el análisis, se estudian los requisitos que fueron descritos en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar todo el sistema, incluyendo su arquitectura.

Aunque en el modelo del análisis hay un refinamiento de los requisitos, no se tiene en cuenta el lenguaje de programación a usar en la construcción, la plataforma en la que se ejecutará la aplicación, los componentes prefabricados o reusables de otras aplicaciones, entre otras características que afectan al sistema, ya que el objetivo del análisis es comprender perfectamente los requisitos del software y no precisar cómo se implementará la solución.

Se utilizará la arquitectura en capas, el objetivo primordial de la programación por capas es la separación de la lógica de negocios de la lógica de diseño, la arquitectura más simple aplicando este patrón consiste en separar la capa de datos de la capa de presentación al usuario, actividad que usualmente suele hacerse estando inconscientemente aplicando esta pauta de diseño de software.

La mayor ventaja de este patrón es que como el desarrollo está distribuido en varios niveles o capas, en caso que sea necesario algún cambio, solo se cambiaría lo necesario en la capa en que se encuentra el componente, sin tener que revisar entre todo el código.

Además, permite distribuir el trabajo de desarrollo de un software por niveles; logrando trabajar en varios grupos al mismo tiempo sin tener que preocuparse unos del otro, de forma que bastaría con conocer la API que existe entre niveles.

A continuación se exponen los diagramas de secuencia y de colaboración del análisis con el objetivo de comprender mejor los procesos que se llevan a cabo en el dominio del sistema, así como cuáles son los eventos y operaciones que realizará.

Diagramas de Interacción (Secuencia y Colaboración)

Un diagrama de secuencia muestra una interacción que está organizada como una secuencia temporal. En particular, muestra los objetos que participan en la interacción mediante sus líneas de vida y mediante los mensajes que intercambian, organizados en forma de una secuencia temporal. Un diagrama de secuencia no muestra los enlaces existentes entre objetos.

Una colaboración modela los objetos y los enlaces significativos dentro de una interacción. Los objetos y los enlaces son significativos solamente en el contexto proporcionado por la interacción. Un rol describe un objeto, y un rol en la asociación describe un enlace dentro de una colaboración.

Un diagrama de colaboración muestra los roles en la interacción en una disposición geométrica. Los mensajes se muestran como flechas, ligadas a las líneas de la relación, que conectan a los roles. La secuencia de mensajes, se indica con los números secuenciales que preceden a las descripciones del mensaje. Los diagramas se organizarán por casos de usos para una mayor comprensión.

4.1.1 Caso de uso Importar Mapfile

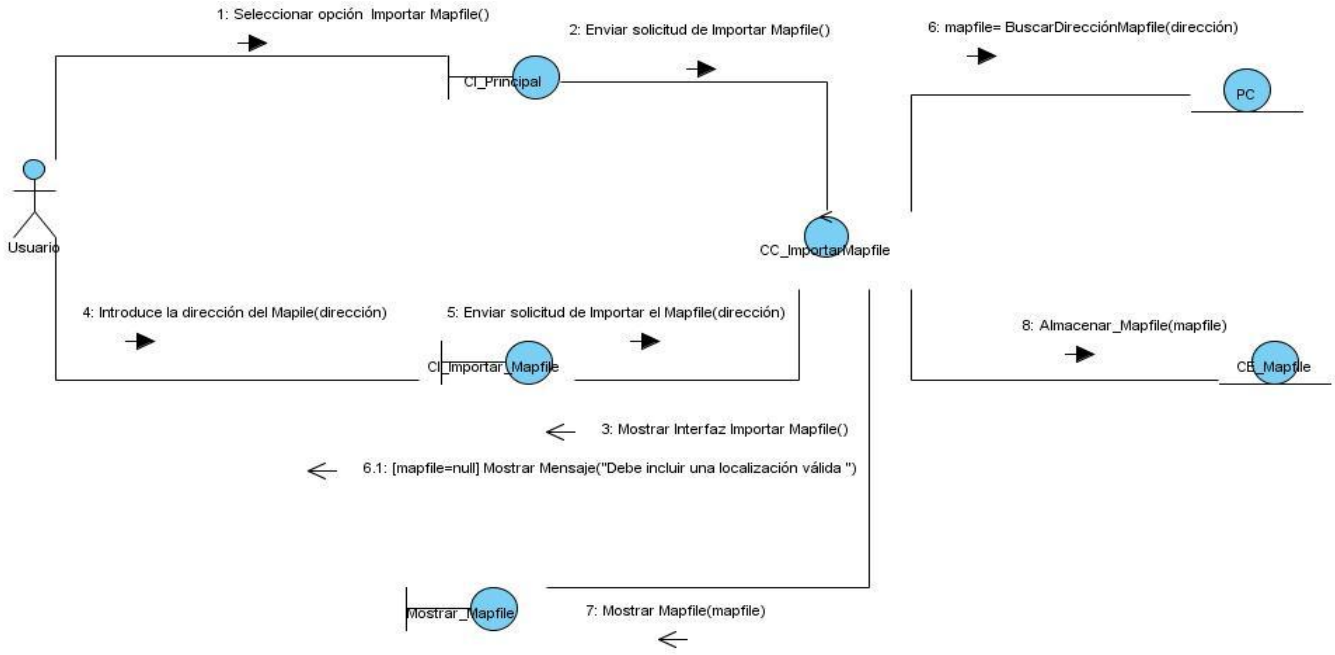


Figura 9: Diagrama de Colaboración Caso de Uso Importar Fichero.

Este diagrama describe todo el proceso *de Importar* el Mapfile hacia el catálogo de mapas LiberMap. Ver *Figura 10.*

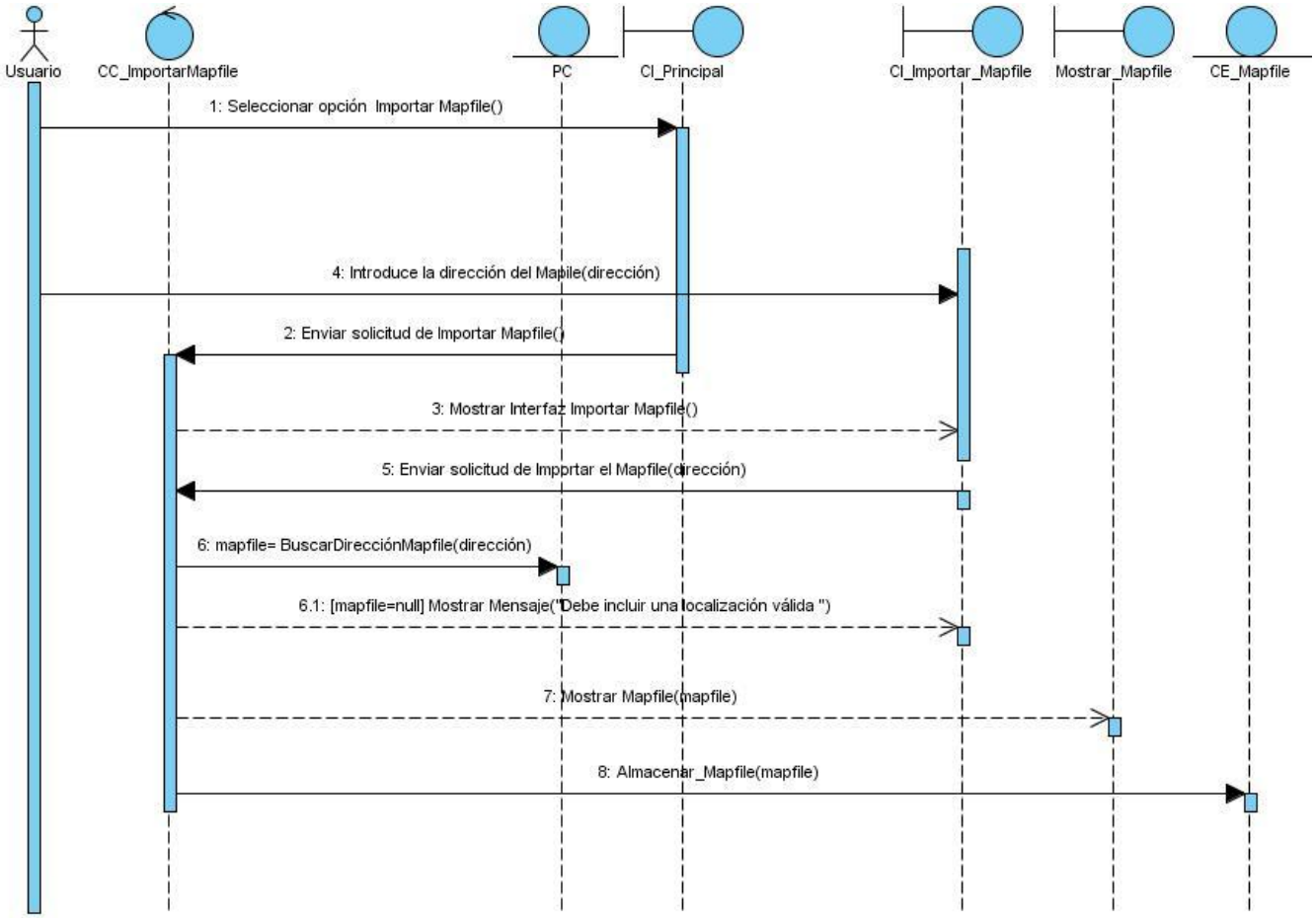


Figura 10: Diagrama de Secuencia Importar Mapfile.

Después de haberse expuesto los principales eventos y operaciones que se desarrollan dentro de cada caso de uso en el análisis, se tiene una primera aproximación de lo que será el sistema.

4.2 Fase de Diseño de la Solución Propuesta

El objetivo de la fase de diseño es la generación de una descripción sobre cómo sintetizar los objetos extraídos del dominio de la solución propuesta al problema donde el principal resultado del diseño es

el modelo del diseño, que se esfuerza en conservar la estructura del sistema, impuesta por el modelo de análisis, y que sirve como esquema para la implementación.

En el diseño se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Es el centro de atención al final de la fase de elaboración y el comienzo de las iteraciones de construcción. Esto contribuye a una arquitectura estable y sólida, y crear un plano del modelo de implementación.

Como se explicaba con anterioridad es una abstracción del Modelo de Implementación y su código fuente, el cual fundamentalmente se emplea para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas a implementación. Representa a los casos de uso en el dominio de la solución.

Puede contener: los diagramas, las clases, paquetes, subsistemas, cápsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo.

Concretamente se definen como propósitos del diseño:

- Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia y tecnologías de interfaz de usuario.
- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo.
- Capturar las interfaces entre los subsistemas antes en el ciclo de vida del software, lo cual es muy útil cuando se utilizan interfaces como elementos de sincronización entre diferentes equipos de desarrollo.

4.2.1 Principios de diseño

El diseño de un sistema informático sea cual sea su finalidad, siempre tiene que favorecer al usuario. Por lo cual se utilizan ciertos principios y patrones que en general ayudarán a garantizar la usabilidad de cualquier sistema.

Patrón Experto: la clase que cuenta con la información necesaria es la ideal para cumplir la responsabilidad.

“Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Con él no se pretende designar una idea oscura ni extraña; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen”. **(14)**

Creador: guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.

“El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento.” **(14)**

Alta Cohesión: asignar una responsabilidad de modo que la cohesión siga siendo alta. Es un principio que se debe tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica al valorar sus decisiones de diseño. **(14)**

Bajo Acoplamiento: Asignar una responsabilidad para mantener bajo acoplamiento. Es un principio que debemos recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño.

Controlador: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase.

A continuación se representan los distintos diagramas de clases del diseño vinculados a los diferentes casos de uso del sistema propuesto, con sus respectivas descripciones. (*Figura 12 y 13*)

4.2.2 Descripción de la propuesta de solución. Diagramas de Clases del Diseño

Debido a que la aplicación desarrollada fue integrada al catálogo de mapas LiberMap a continuación se realiza una descripción de la arquitectura del mismo para una mejor comprensión del diseño.

Descripción del diagrama clases del diseño del caso de uso Importar Mapfile. Arquitectura de LiberMap

Organización de la estructura de LiberMap con Symfony

Symfony implementa la arquitectura MVC permitiendo que se desarrollen aplicaciones con mayor rapidez y sea un proceso sencillo. Este framework genera la estructura de los proyectos que lo emplean.

Modelo

Se divide en las capas de acceso a datos y la capa de abstracción de la Base de Datos. La primera representa la lógica del negocio del sistema, es decir, las reglas del negocio y los requerimientos funcionales que debe cumplir el sistema. La segunda capa brinda la posibilidad de no tener que generar sentencias SQL, lo que les facilita el trabajo a los programadores. Además de permitir que si se desea cambiar el gestor de Base Datos, esto sea posible sin tener que realizar cambios en la lógica del negocio, para ello se auxilia de la librería de Propel[6].

En las aplicaciones realizadas con Symfony, el acceso y la modificación de los datos almacenados en la base de datos se realiza mediante objetos; por lo que nunca se accede de forma explícita a la base de datos. Además genera por cada tabla de la Base de Datos, cuatro clases: NombreTabla, NombreTablaPeer, BaseNombreTabla, BaseNombreTablaPeer, las cuales son las que se encargan en conjunto de realizar todo el acceso de los datos y la lógica de la aplicación.

- **NombreTabla y NombreTablaPeer:** Se encargan de toda la lógica del negocio.
- **BaseNombreTabla:** Contiene todos los atributos definidos en la tabla y un conjunto de métodos ya implementados que gestionan el acceso a los datos, aceleran y simplifican el trabajo del equipo.

[6] ORM acrónimo de (object-relational mapping) para PHP que facilita la labor de desarrollo de aplicaciones web, gracias a la capa que transforma el tratamiento de la BD mediante objetos, con la que se puede recuperar, insertar y modificar datos.

- **BaseNombreTablaPeer:** Contiene un conjunto de métodos estáticos que complementan el acceso y la lógica de los datos.

Vista

En Symfony la vista está formada principalmente por plantillas en PHP. Es aquí donde se transforma el modelo en una página Web con la que el usuario interactúa. Consta de tres partes fundamentales:

- **Layout (plantilla global):** Contiene el código HTML que es común en todas las páginas, evitando de esta forma tener que repetirlo en todas las páginas.
- **Complemento de las acciones (plantillas):** Toman los resultados de la acción y se insertan en el cuerpo del layout para generar finalmente la página Web como resultado de la petición de un usuario.
- **Páginas clientes y formularios:** Son las páginas que se generan finalmente y con las que el usuario interactúa.

Controlador

Se encarga de procesar las interacciones del usuario y realizar los cambios apropiados en el modelo o en la vista. Es el responsable del manejo de las peticiones del usuario, cargar la configuración de la aplicación, el manejo de la seguridad, entre otras tareas. Se divide en un controlador frontal y las acciones.

- **Acciones:** Se encargan de obtener los resultados del modelo y definen variables para la vista. Representan una actividad específica que el sistema debe realizar, para ofrecer un resultado determinado a una petición.
- **Controlador Frontal:** Es el único punto de entrada de la aplicación, carga la configuración y determina las acciones a ejecutarse. El controlador frontal, al igual que el layout es común para todas las acciones de la aplicación.

Organización de la estructura de LiberMap con EXTJS.

Las interfaces de la aplicación se realizarán con EXTJS. La utilización de esta librería de JavaScript, se expresará a través de un paquete que estará representando todos los componentes de la propia librería. Se utilizarán un número de ficheros JS comunes para cada caso de uso. Estos ficheros serán los encargados de configurar algunos elementos generales del funcionamiento de la aplicación. Al mismo tiempo en cada caso de uso se estará haciendo uso de otros ficheros JS específicos para cada uno, dichos ficheros se diseñarán en el propio caso de uso al que pertenezcan.

En resumen, la estructura general del catálogo de mapas LiberMap, quedaría de la forma que se muestra en el siguiente diagrama de paquetes (*Figura 11*).

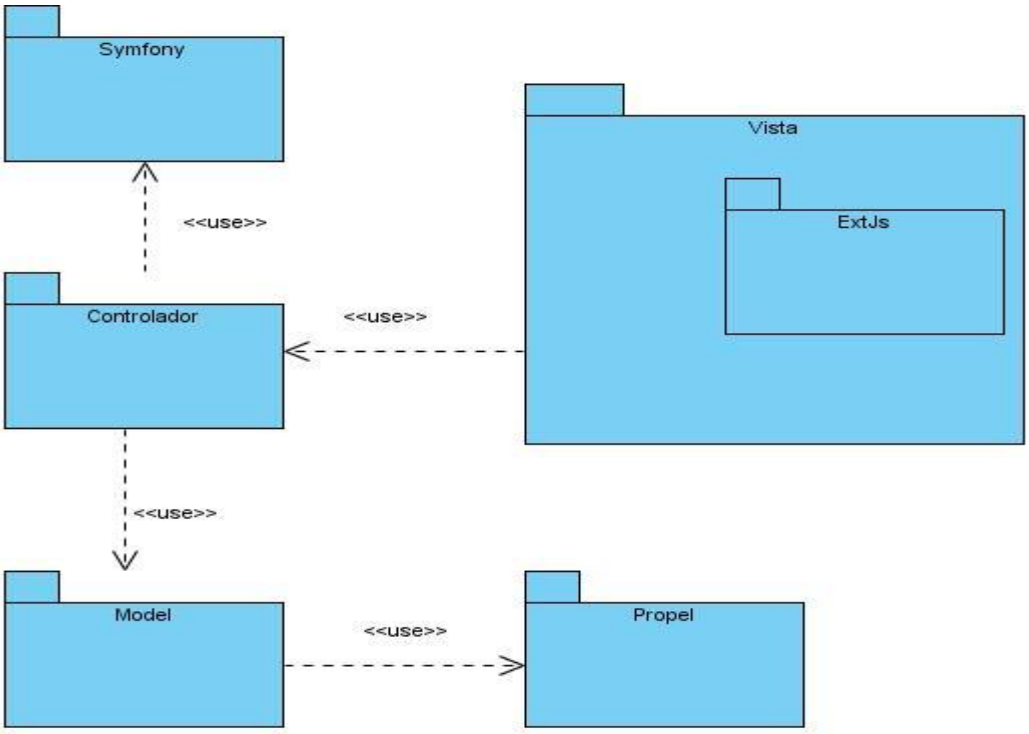


Figura 6: Estructura de LiberMap.

4.2.3 Diagrama de clase del diseño del Caso de Uso Importar Mapfile.

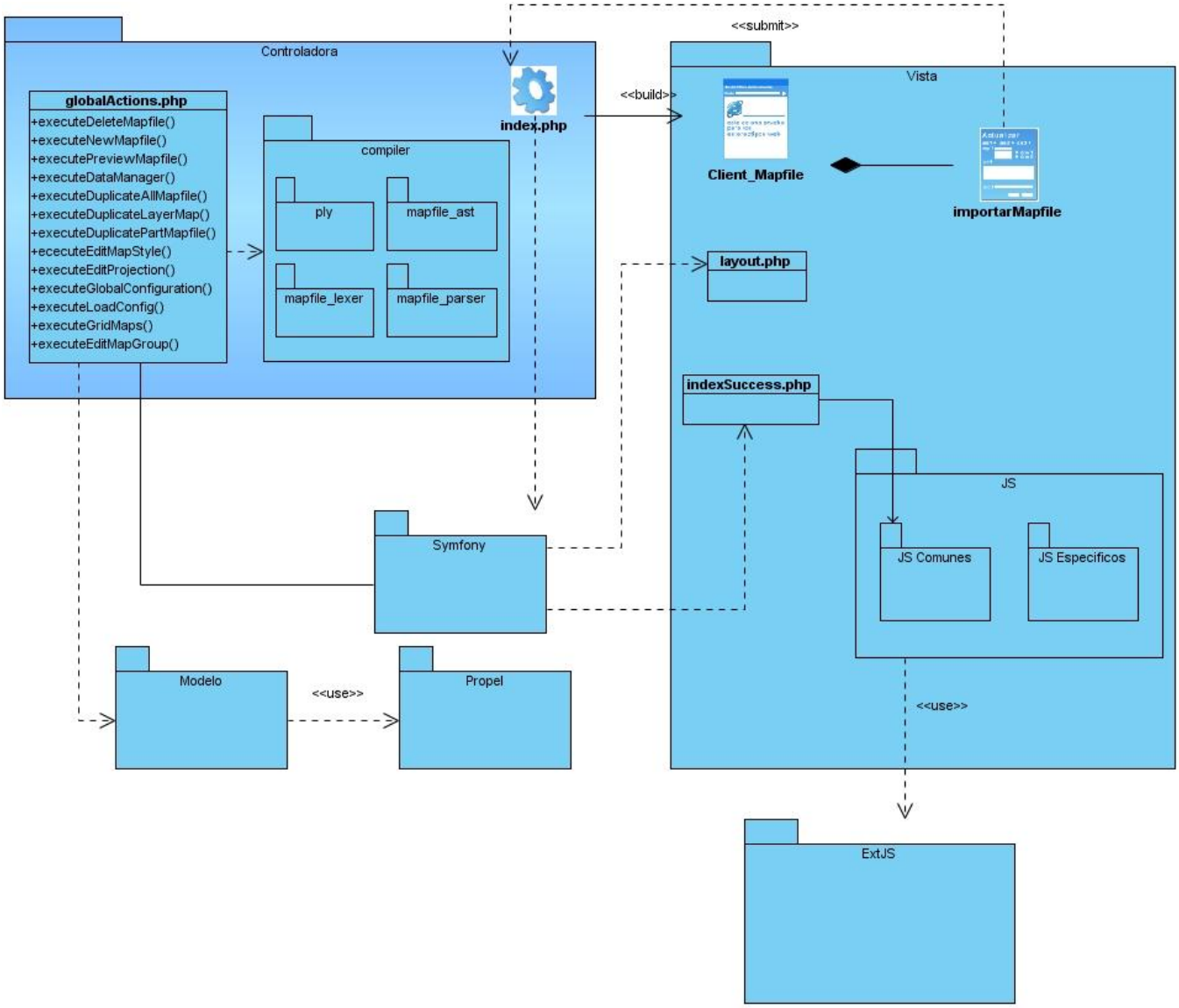


Figura 7: Diagrama de clase del diseño del Caso de Uso Importar Mapfile.

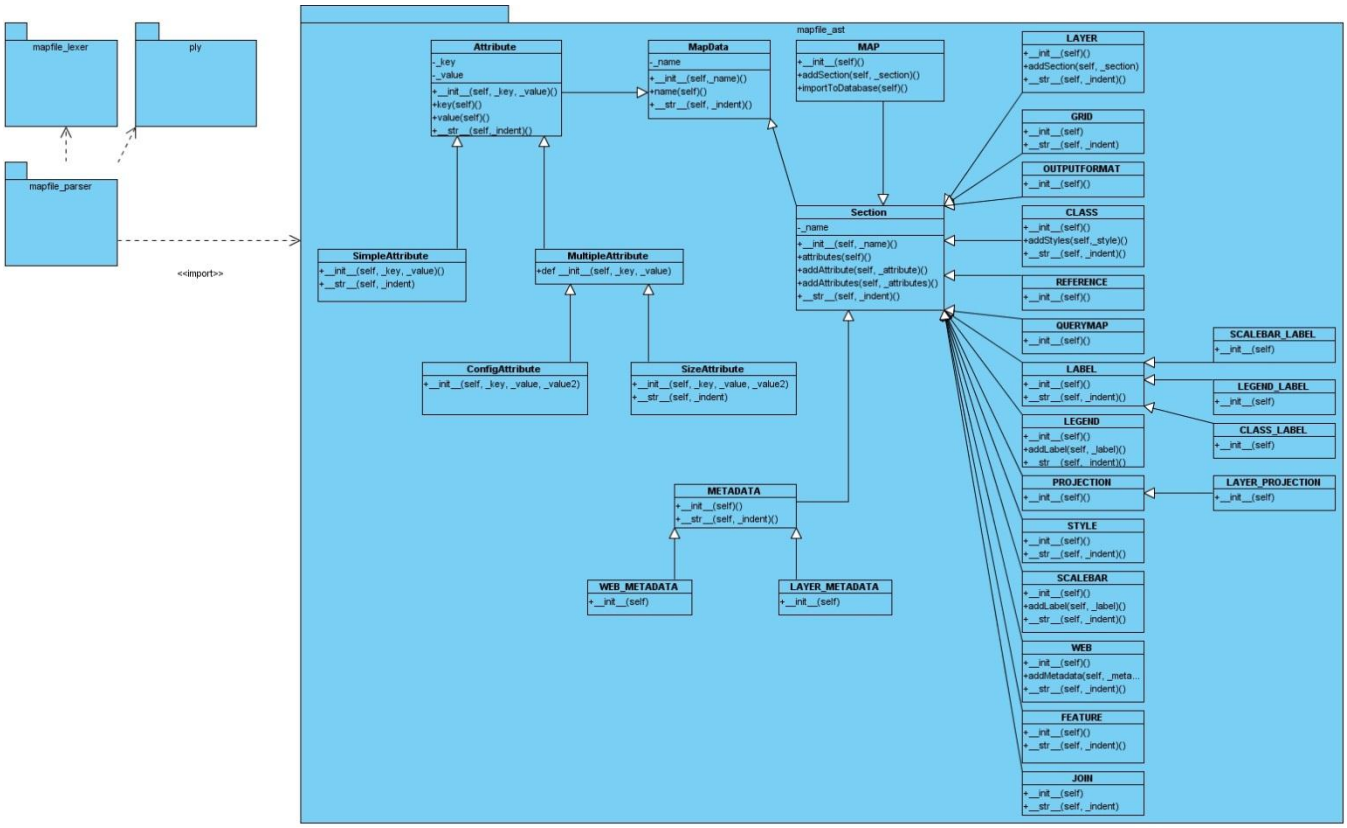


Figura 8: Diagrama de clase del diseño del Caso de Uso Compilar Mapfile.

Para una mejor comprensión del Diagrama de clase del diseño del Caso de Uso Compilar Mapfile el mismo se dividirá en diferentes vistas. Ver (*Figura 15,16 y 17*)

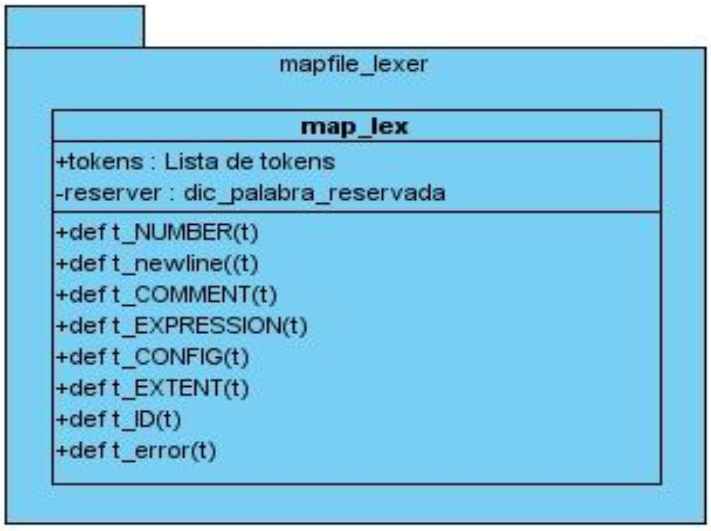


Figura 9: Vista del diseño del módulo map_lex

Figura 10: Vista del diseño del módulo map_yacc

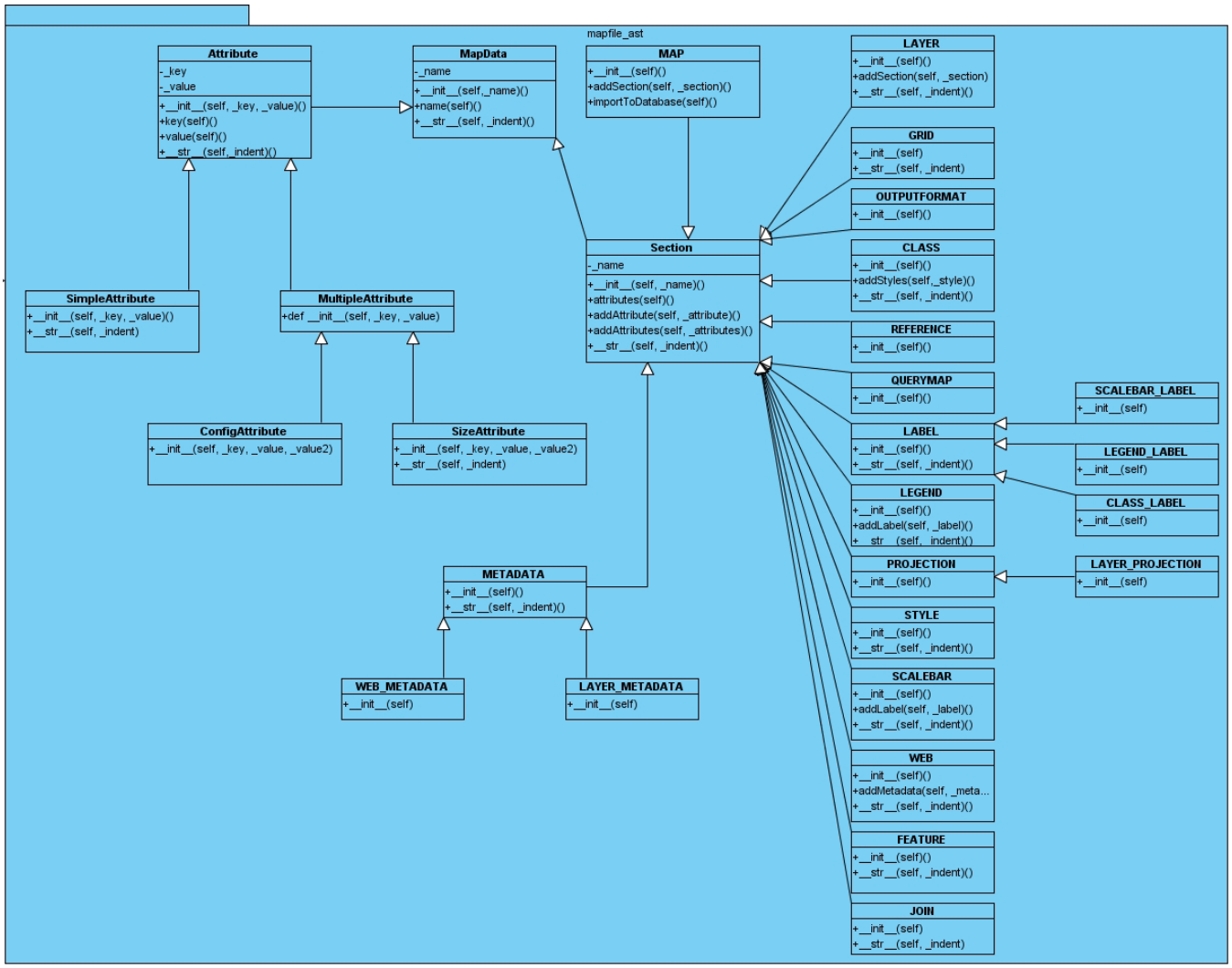


Figura 11: Vista del diseño del módulo map_ast

Descripción de los diagramas de clases del diseño de los casos de uso.

Lexer

La función principal de un *lexer* es tomar un flujo de caracteres entrada y devolver un flujo de *tokens* como salida. Al módulo que generará un analizador léxico se le llamó map_lex.py .Dentro de él están contenidas las principales funcionalidades, expresiones regulares y estructuras de datos que ayudaran a realizar el análisis léxico de la cadena entrada.

Se creó una tupla con todos los nombres de los tokens a reconocer (por convención se escribieron los nombres en mayúsculas). Esta tupla contiene una lista de símbolos que define todos los tokens de

posibles nombres que pueden ser producidos por el analizador léxico Esta lista siempre es necesaria y se utiliza para realizar una variedad de controles de validación. La lista de tokens también es utilizada por el módulo yacc.py para identificar los terminales. Ejemplo de la lista de tokens:

```
Tokens= ('MAP','END','REFERENCE','QUERYMAP','WEB','METADATA',
'OUTPUTFORMAT','LEGEND','SCALEBAR','LABEL','LAYER','FEATURE', 'SUB','NUMBER',
'CONFIG', 'SIZE', 'EXTENT','ID', 'EXPRESSION')
```

Se emplea un diccionario en el cual la clave es una palabra reservada y el valor uno de los tokens de la tupla anterior:

```
reserved = {
'MAP' : 'MAP',
'OUTPUTFORMAT' : 'OUTPUTFORMAT',
'SCALEBAR' : 'SCALEBAR',
'LABEL' : 'LABEL',
'LEGEND' : 'LEGEND',
'METADATA' : 'METADATA',
'REFERENCE' : 'REFERENCE',
'QUERYMAP' : 'QUERYMAP',
'WEB' : 'WEB',
'LAYER' : 'LAYER',
'FEATURE' : 'FEATURE',
'END' : 'END'
}
```

Lo siguiente es definir las expresiones regulares para cada token. Existen dos formas de hacerlo, mediante *strings* o mediante funciones. El primer caso se usa cuando el token no requiere ningún tipo de procesamiento luego de ser encontrado:

```
t_SUB    = r'-'
```

Se usan *raw strings* de Python para escribir las expresiones regulares que posteriormente serán compiladas y usadas (PLY utiliza el módulo re en su análisis léxico).

Para los tokens correspondientes a NUMBER, CONFIG, SIZE, EXTENT entre otros más, se le hacen algunas verificaciones antes de devolverlo, donde la especificación del token se genera mediante una función:

```
deft_NUMBER(t):
    r'[0-9][.0-9]*'
    try:
       t.value = int(t.value)
    exceptValueError:
```

```
        try:
           t.value = float(t.value)
        exceptValueError:
            print "Line %d: Number %s is too large!" % (t.lineno,t.value)
            t.value = 0
return t
```

Luego de encontrar una secuencia de caracteres que corresponda con la expresión regular de los símbolos, el lexer comprueba que no sea una palabra reservada. Si lo es, en t.type se guardará el nombre de token correspondiente.

El orden en que estas definiciones son usadas es el siguiente: primero los strings en orden descendiente de la longitud de la expresión regular y luego las funciones en el orden que fueron escritas.

Números de línea y la información de posición

De forma predeterminada, lex.py no sabe nada de números de línea. Esto se debe a lex.py no reconoce lo que es una "línea" de entrada (por ejemplo, el carácter de nueva línea, o incluso si la entrada es de datos textuales). Para actualizar esta información, se requiere escribir una nueva regla que ayude al analizador seguir los números de línea. En la aplicación, el t_newline () es la regla que se encarga de este proceso.

```
deft_newline(t):
   r'\n+'
   t.lexer.lineno += len(t.value)
```

Símbolos descartados

Para descartar un símbolo, como un comentario, sólo se tiene que definir una regla que no devuelve ningún valor. Por ejemplo:

```
deft_COMMENT(t):
r'\#.*'
pass
```

Gestión de errores

El t_error () se utiliza para controlar los errores de léxico que se producen cuando los caracteres ilegales son detectados En este caso, el atributo t.value contiene el resto de la cadena de entrada que no se ha cortado. En el programa la función de error se define como sigue:

```
deft_error(t):
   print "Illegal character '%s'" % t.value[0]
   t.lexer.skip(1)
```

En este caso, basta con imprimir el carácter ilegal y pasar por delante un carácter llamando t.lexer.skip (1).

Caracteres ignorados

La regla t_ignore está especialmente reservada por lex.py para los caracteres que deben ser completamente ignorados en el flujo de entrada. Por lo general, esta se utiliza para saltar sobre los caracteres no esenciales, espacios en blanco y tabs Aunque es posible definir una expresión regular para los espacios en blanco de manera similar a t_newline (), el uso de t_ignore proporciona un mejor rendimiento sustancialmente léxico, ya que se trata como un caso especial y se verificará en una forma mucho más eficiente que las reglas normales de expresiones regulares.

Cuando se ejecuta, el módulo producirá el siguiente resultado:

```
LexToken(MAP,'MAP',3,52)
LexToken(CONFIG,'CONFIG',4,68)
LexToken(ID,'PPP',4,75)
LexToken(ID,'PPP',4,79)
LexToken(EXTENT,'EXTENT',5,95)
LexToken(NUMBER,1,5,102)
LexToken(NUMBER,1,5,104)
LexToken(NUMBER,1,5,106)
LexToken(NUMBER,1,5,108)
LexToken(OUTPUTFORMAT,'OUTPUTFORMAT',6,122)
```

Como muestra la ejecución del módulo, los tokens que devuelve lexer.token () son instancias de LexToken. Este objeto tiene atributos tok.type, tok.value, tok.lineno y tok.lexpos. Los atributos tok.value y tok.type contienen el valor y el tipo de la propia tupla. Tok.line y tok.lexpos contienen

información sobre la ubicación del símbolo. Tok.lexpos es el índice de la relación simbólica con el inicio de la entrada de texto.

Construcción y utilización del lexer

Para generar el analizador léxico, se utiliza la función el lex.lex(). Esta función utiliza la reflexión de Python (o introspección) para leer las normas de la expresión regular fuera del contexto de llamada y construir el analizador léxico. Una vez que el analizador léxico se ha construido, dos métodos se pueden utilizar para controlar el léxico.

- lexer.input (data).Se utiliza para restablecer el léxico y almacenar una nueva cadena de entrada.
- lexer.token (). Devuelve el siguiente token. Devuelve un LexToken en el caso especial que se tenga éxito o ninguno si se ha alcanzado el final del texto.

```
lex.input(_data)

    # Tokenize
    while 1:
        tok = lex.token()
        if not tok: break      # No more input
        print tok
    #print tok.type, tok.value, tok.lineno, tok.lexpos
```

Parser

El módulo map_yacc.py invoca al analizador léxico para obtener los tokens. Por lo que se tiene que importar el analizador léxico que se ha descrito antes. Se debe definir una función por cada una de las reglas previas (el *docstring* de la función corresponde a la regla). Cada función recibe como parámetro un objeto iterable (muy parecido a una lista, pero que no se comporta totalmente como tal, por lo se debe tener cuidado con los subíndices negativos) que contiene los valores de cada símbolo de la regla. Dentro de la función se puede escribir las acciones semánticas necesarias para ser hechas.

Cada función acepta un único argumento $\boldsymbol{p}$, formando una secuencia que contiene los valores de cada símbolo de la gramática en la norma correspondiente. Los valores de $\boldsymbol{p}[\boldsymbol{i}]$ se le asignan a los símbolos de la gramática, como se muestra en el siguiente ejemplo:

```
defp_expression_plus(p):
    'expression : expression PLUS term'
    #   ^            ^        ^    ^
   #   p[0]         p[1]     p[2] p[3]

      p[0] =       p[1]     +    p[3]
```

Fragmento de código del parser:

```
defp_mapfile(p):
    'mapfile : MAP section_list END'

#regla recursiva que permite tener un conjunto de secciones
    defp_section_list(p):
        '''section_list :section_list section
                        | section
        '''
        p[0] = p[1]
```

Para las fichas, el "valor" de la p correspondiente a la posición $[i]$ es el mismo que el atributo p.value asignado en el módulo analizador léxico. Para los no-terminales, el valor está determinado por lo que se coloca en la $p[0]$ cuando las reglas se reducen. Este valor puede ser cualquier cosa (cualquier tipo de dato de Python). Todas las otras reglas simplemente realizan varios tipos de operaciones con enteros y propagan el resultado. De esta forma queda expresada la gramática en código**.**

Definiendo la Gramática del compilador

Para comenzar la construcción del compilador, se tiene que tener bien definido el lenguaje a reconocer, así como la gramática que guiará a este proceso. Esto se hace al escribir un conjunto de reglas o gramática para el lenguaje particular. Las palabras en mayúsculas representan símbolos terminales y las palabras en minúsculas símbolos no terminales, a la izquierda de la regla siempre va un único elemento y el símbolo (:) puede leerse como 'es'. De esta forma queda definida la gramática:

mapfile : MAP section_list END

section_list : section_list section | section

section : outputformat | projection | scalebar | legend | web | reference | querymap | layer | attribute

outputformat : OUTPUTFORMAT attributes_list END
| OUTPUTFORMAT empty END

projection : PROJECTION EXPRESSION END

scalebar : SCALEBAR scalebar_section_list END
| SCALEBAR empty END

scalebar_section_list : scalebar_section_list scalebar_section
| scalebar_section

scalebar_section : label | attribute | STYLE ID | STYLE numeric_value

legend : LEGEND legend_section_list END
| LEGEND empty END

legend_section_list : legend_section_list legend_section | legend_section

legend_section : label | attribute

class : CLASS class_section_list END | CLASS empty END

class_section_list : class_section_list class_section | class_section

class_section : label | attribute | style

reference : REFERENCE attributes_list END
| REFERENCE empty END

querymap : QUERYMAP query_section_list END
| QUERYMAP empty END

query_section_list : query_section_list query_section | query_section

query_section : attribute | STYLE ID | STYLE numeric_value

web : WEB attributes_list metadata END
| WEB empty END

metadata : METADATA metas_list END
| METADATA empty END

label : LABEL attributes_list END | LABEL empty END

layer : LAYER layer_section_list END
| LAYER empty END

layer_section_list : layer_section_list layer_section | layer_section

layer_section : metadata | feature | grid | class | join | Projection |attribute

feature : FEATURE attributes_list END
| FEATURE empty END

grid : GRID attributes_list END
| GRID empty END

join : JOIN attributes_list END
| JOIN empty END

style : STYLE attributes_list END | STYLE empty END

attributes_list : attributes_list attribute | attribute

attribute : simple | multiple

simple : ID ID | ID EXPRESSION | ID numeric_value

multiple : color_attribute | size_attribute | extent_attribute | config_attribute

color_attribute : ID numeric_value numeric_value numeric_value

size_attribute : ID numeric_value numeric_value

extent_attribute : EXTENT numeric_value numeric_value numeric_value numeric_value

config_attribute : CONFIG ID ID | CONFIG ID EXPRESSION
| CONFIG EXPRESSION ID
| CONFIG EXPRESSION EXPRESSION

numeric_value : NUMBER | SUB NUMBER

metas_list : metas_list meta | meta

meta : ID ID | ID EXPRESSION | EXPRESSION ID | EXPRESSION EXPRESSION

Construcción del AST

Para la construcción del árbol de sintaxis abstracta se crea un conjunto de estructuras de datos para diferentes tipos de nodos del AST y se le asignan los nodos de $p[0]$ en cada regla. Por ejemplo:

```
class Attribute(MapData):

    def __init__(self, _key, _value):
       MapData.__init__(self, 'Attribute')
       self._key = _key
       self._value = _value

    @property
    def key(self):
      returnself._key

    @property
```

```
    def value(self):
       returnself._value

class SimpleAttribute(Attribute):

    def __init__(self, _key, _value):
       MapData.__init__(self, 'SimpleAttribute')
       Attribute.__init__(self, _key, _value)

    def __str__(self,_indent):
       print _indent + self._key, self._value
```

La siguiente figura muestra una sección del AST que genera la herramienta de compilación. *Figura 18*.

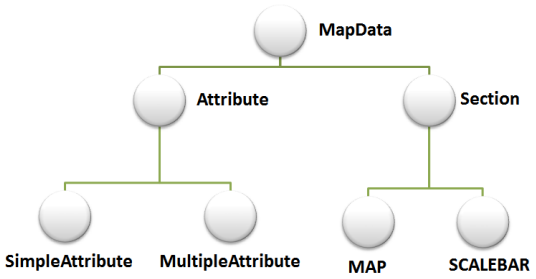


Figura 12: Sección del AST

La ventaja de este enfoque es que puede hacer que sea más fácil colocar la semántica, la comprobación de tipos, la generación de código, y otras características a las clases nodo.

Una vez compilado el fichero (.map) la aplicación habrá generado un árbol de sintaxis abstracta conformado por un conjunto de estructuras de datos para diferentes tipos de nodos del AST. Donde el siguiente paso del proceso para importar el Mapfile será el almacenamiento de las estructuras en la base de datos de LiberMap. Esto se logra mediante la utilización del ORM Sqlalchemy, el cual provee la funcionalidad de convertir los datos entre el sistema de tipos utilizado en Python y el utilizado en la base de datos relacional, utilizando un motor de persistencia, es decir se estará creando una base de datos orientada a objetos virtual, sobre la base de datos relacional. (*Figura 18*)

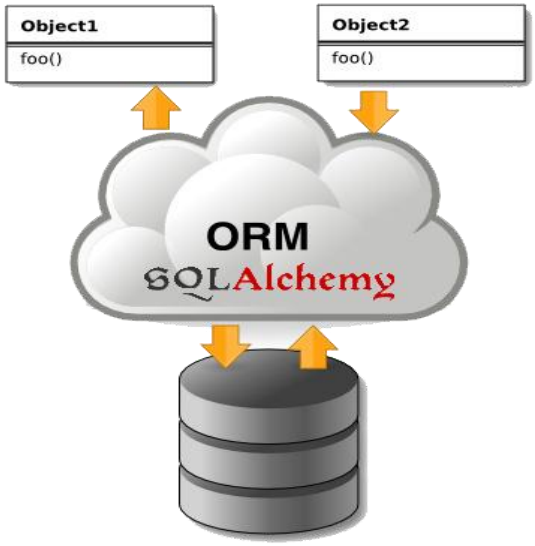


Figura 13: Proceso de la ORM Sqlalchemy

Utilizar el ORM Sqlalchemy tiene una serie de ventajas que facilitan enormemente tareas comunes y de mantenimiento:

- **Reutilización:** La principal ventaja que aporta un ORM es la reutilización permitiendo llamar a los métodos de un objeto de datos desde distintas partes de la aplicación e incluso desde diferentes aplicaciones.
- **Encapsulamiento:** La capa ORM encapsula la lógica de los datos permitiendo hacer cambios que afectan a toda la aplicación únicamente modificando una función.
- **Portabilidad:** Utilizar una capa de abstracción nos permite cambiar en mitad de un proyecto de una base de datos PostgresSQL a una Oracle sin ningún tipo de complicación. Esto es debido a que no utilizamos una sintaxis PostgresSQL, Oracle o SQLite para acceder a nuestro modelo, sino una sintaxis propia del ORM utilizado que es capaz de traducir a diferentes tipos de bases de datos.
- **Seguridad:** Los ORM suelen implementar mecanismos de seguridad que protegen nuestra aplicación de los ataques más comunes como SQL Injections.
- **Mantenimiento del código:** Gracias a la correcta ordenación de la capa de datos, modificar y mantener nuestro código es una tarea sencilla.

4.3 Modelo de Despliegue

La vista de despliegue que será mostrada suministra una base para la comprensión de la distribución física de la herramienta de compilación a través de un conjunto de nodos. Se podrá observar además cómo existe una traza directa del modelo de implementación, puesto que cada componente físico identificado en el modelo anteriormente mencionado, debe estar almacenado en un nodo.

4.3.1 Definiciones, acrónimos y abreviaturas del modelo de despliegue

Nodos: Elementos de procesamiento con al menos un procesador, memoria, y posiblemente otros dispositivos.
Dispositivos: Nodos estereotipados sin capacidad de procesamiento en el nivel de abstracción que se modela.
Conectores: Expresa el tipo de conector o protocolo utilizado entre el resto de los elementos del modelo.
FTP: Protocolo de transferencia de archivos.
TCP: Protocolo de control de trasmisión.
IP: Protocolo de internet.
HTTP: Protocolo de transferencia de hipertexto.
HTTPS: Protocolo Seguro de Transferencia de Hipertexto.

USB: Conductor universal en serie.

4.3.2 Diagrama de despliegue

El Modelo de despliegue se encarga de capturar la configuración de los elementos de procesamiento, y las conexiones entre estos elementos en el sistema. Consiste en uno o más nodos, dispositivos y conectores, estos últimos estarán ubicados entre nodos, y entre nodos y dispositivos. El modelo de despliegue también mapea procesos dentro de estos elementos de procesamiento, permitiendo la distribución del comportamiento a través de los nodos que son representados. También se utiliza para visualizar la distribución de los componentes de software en los nodos físicos. (*Figura 19*)

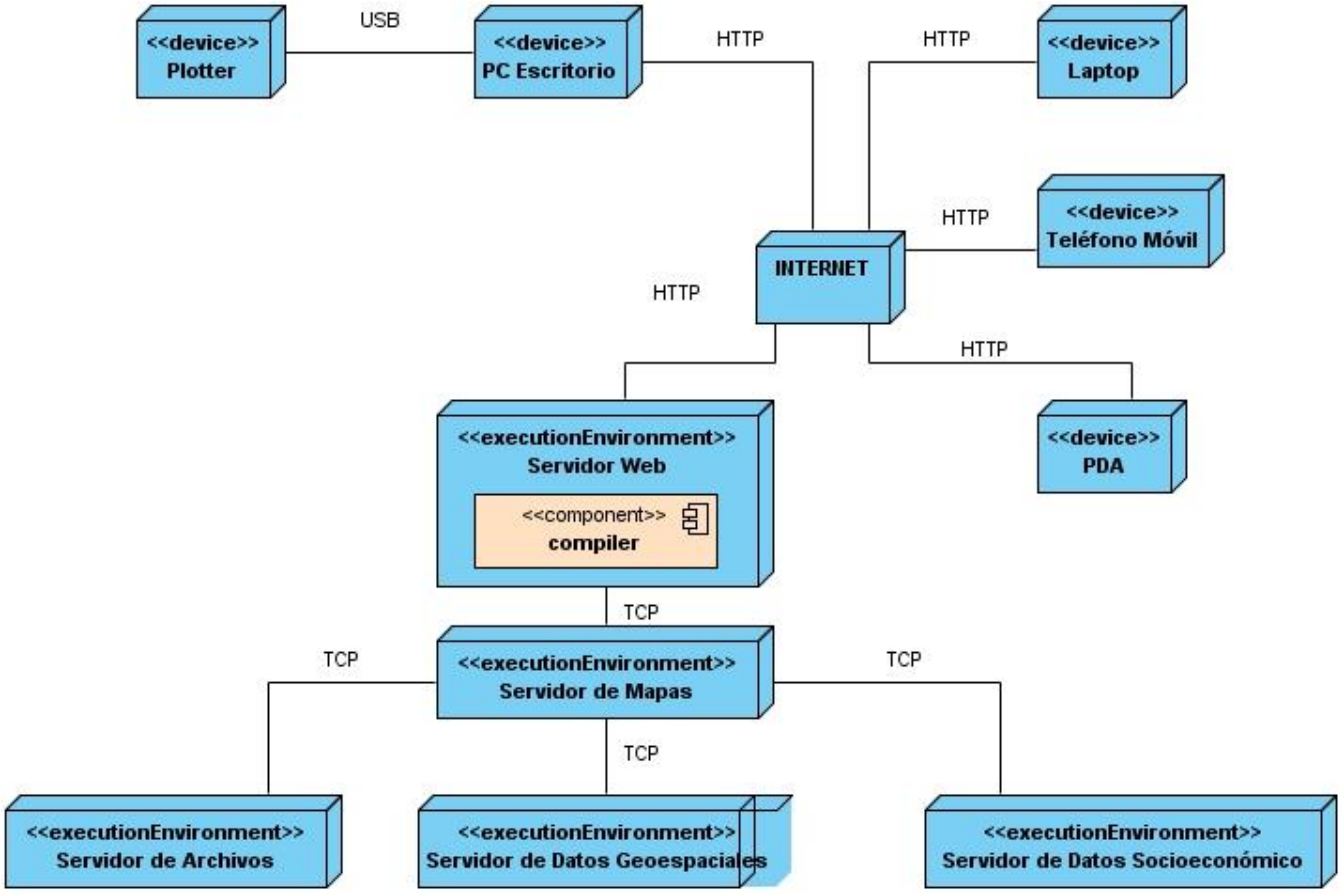


Figura 19: Diagrama de despliegue

4.4 Modelo de Implementación

Durante el flujo de trabajo de Implementación en la fase de construcción unos de los artefactos que se elaboran es el diagrama de componentes que muestra las organizaciones y dependencias lógicas entre componentes del software, sean estos componentes de código fuente, binarios o ejecutables. Los elementos de modelado dentro del diagrama de componentes serán componentes y paquetes.

A continuación se muestra en la *Figura 20* la vista general del diagrama de componentes correspondiente al sistema desarrollado.

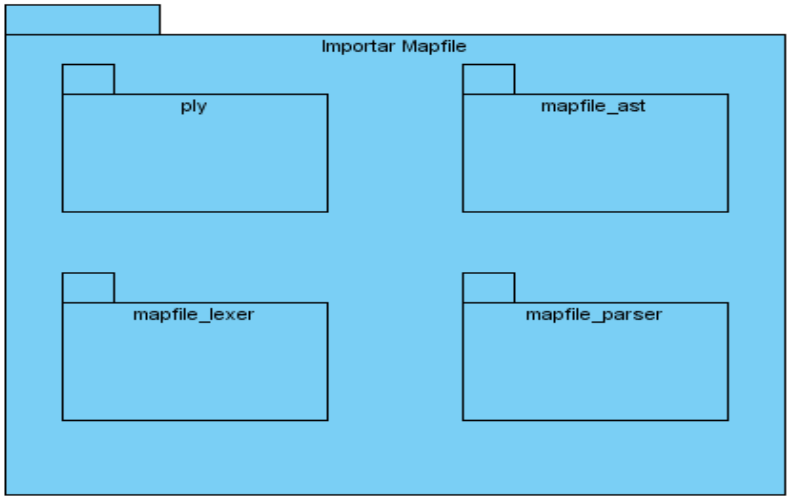


Figura 14: Diagrama de Paquetes.

El sistema estará estructurado en cuatro paquetes y cinco componentes, como se muestra en la *Figura 22.*

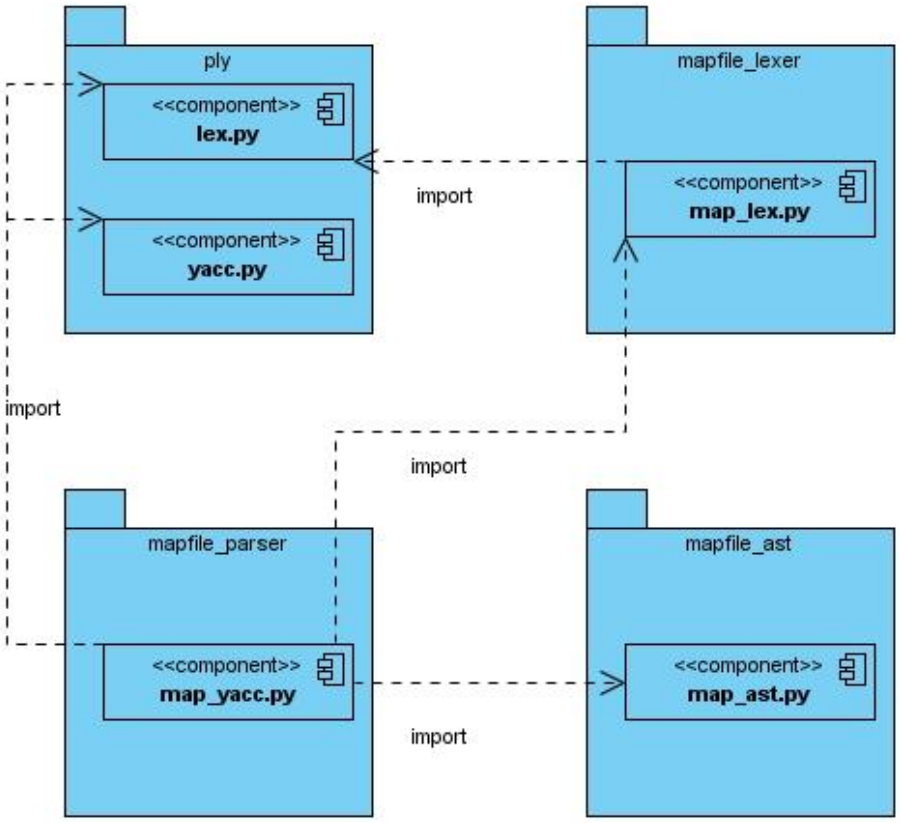


Figura 15: Diagrama de Componentes.

4.5 Validación de la solución propuesta

Las pruebas de un software están constituidas por un conjunto de herramientas, técnicas y métodos que tributan a la excelencia del desempeño de un programa. Técnicamente consisten en inspeccionar manualmente el código o hacer pruebas de ensayos (ejecutar el software y ver los resultados), el objetivo principal es encontrar errores o defectos en el software.

Las pruebas que se realizaran al software serán de caja negra, estas pruebas se llevan a cabo sobre la interfaz del software, y es completamente indiferente el comportamiento interno y la estructura del programa. Los casos de prueba de la caja negra pretenden demostrar que:

- Las funciones del software son operativas.
- La entrada se acepta de forma adecuada.
- Se produce una salida correcta
- La integridad de la información externa se mantiene.

Se derivan conjuntos de condiciones de entrada que ejerciten completamente todos los requerimientos funcionales del programa.

La prueba de la caja negra intenta encontrar errores de las siguientes categorías:

- Funciones incorrectas o ausentes.
- Errores de interfaz.
- Errores en estructuras de datos o en accesos a la base de datos.
- Errores de rendimiento.
- Errores de inicialización y de terminación.

Utilizando la Técnica Partición Equivalente

Para detallar el caso de uso se utiliza una tabla, donde se desglosa esta funcionalidad en secciones y a su vez estas en escenarios, para hacer más fructífera la ejecución de las pruebas. Esta tabla contiene los campos:

- Nombre de la sección: Se especifica el nombre de la sección [SC 1: Nombre de la sección].
- Escenarios de la sección: Se especifican los escenarios de cada sección [EC 1.1: Nombre del Escenario].

- Descripción de la funcionalidad: Se describe brevemente la funcionalidad del escenario.

A continuación se muestra el caso de prueba de caja negra para el caso de uso Importar Mapfile.

4.5.1 Caso de Prueba del caso de uso Importar Mapfile

Descripción General

El caso de uso se inicia cuando el usuario selecciona la opción "Configuración" y dentro de ella la opción de "Importar un Mapfile" y termina cuando el usuario luego de haber seleccionado la dirección donde se encuentra el Mapfile, selecciona la opción de "Importar" y se almacena el Mapfile con todos sus componentes de forma persistente en la base de datos.

Condiciones de Ejecución

El usuario debe estar autenticado y poseer los privilegios necesarios para acceder a esta opción.

Sección a probar en el Caso de Uso

Tabla 6: Sección Importar Mapfile

Nombre de la sección	Escenarios de la sección	Descripción de la funcionalidad
SC1: Importar Mapfile	EC 1.1: Importar Mapfile	El sistema brinda la posibilidad al usuario de importar un Mapfile.
	EC1.2: Tratamiento de errores cuando se Importa un Mapfile.	El sistema brinda la posibilidad de avisarle al usuario que debe de introducir la dirección y que sea una dirección válida.

Descripción de variable

Tabla 7: Descripción de variable

No	Nombre de campo	Clasificación	Valor Nulo	Descripción
1	Localización	Campo de texto (TextBox)	No	Se debe introducir la Localización.

Matriz de Datos

SC 1<Importar Mapfile>

Tabla 8: Matriz de datos

Escenario	Variable Localización	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
Importar Mapfile	V	El sistema brinda la posibilidad al usuario de importar un Mapfiles.	Satisfactorio	Paso 1: Se brinda la posibilidad de seleccionar la opción de “Importar Mapfile”. Paso 2: El sistema muestra una ventana donde se introduce la dirección. Paso 3: El usuario introduce la dirección donde se encuentra el Mapfile a importar.
Tratamiento de errores cuando se Importa un mapfile.	I	El sistema brinda la posibilidad de avisarle al usuario que debe de introducir la dirección y que sea una dirección válida.	Satisfactorio	Paso 1: El usuario no introduce la dirección donde se encuentra el Mapfile a importar. Paso 2: El usuario introduce una dirección no válida. Paso 3: El sistema muestra un mensaje informando que la dirección no es válida. Paso 4: El sistema muestra un mensaje informando que se debe de introducir la dirección y marca en rojo el campo que debe ser llenado.

Para el desarrollo de la prueba se importaron diversos ficheros Mapfiles donde cada uno trae incluido diferentes secciones con la información sobre el tipo de mapa de retorno, la extensión, la escala mínima y máxima, las definiciones de las capas, entre otros conjuntos de secciones y atributos.

Las pruebas funcionales se realizaron para verificar que el sistema implementa adecuadamente cada uno de los requisitos acordados para el software y que funcionan correctamente. Haciendo uso de los casos de prueba diseñados, donde se recogen los escenarios correspondientes a los casos de uso del sistema utilizados para probar estos escenarios, se probó cada funcionalidad comparando los resultados obtenidos con los resultados esperados, estos últimos también contenidos en los casos de prueba. De esta manera se verificó que la herramienta de compilación respondía a las necesidades de Libermap. Se revisó también que las interfaces de la aplicación utilizarán correctamente el idioma definido, que mantuvieran concordancia entre sus textos y buena ortografía, que los mensajes mostrados fueran claros y concisos. Otro aspecto que se tuvo en cuenta fue la estética y visualización de la interfaz.

De forma general, las pruebas se ejecutaron exitosamente partiendo de que el éxito de las pruebas se encuentra en la detección de errores. Se lograron los objetivos propuestos, que la herramienta de compilación a partir de las pruebas realizadas y su corrección posee una mayor calidad y un alto nivel técnico.

4.6 Conclusiones Parciales

Definir el modelo de dominio permitió tener una mejor visión del problema científico a resolver. A través de las vistas de los diagramas del proceso de análisis y diseño se logró describir la propuesta de solución para el desarrollo del software. La realización del Modelo de Implementación, permitió detallar los componentes creados para el desarrollo de la aplicación y la relación entre ellos. Se llevó a cabo la elaboración de los diseños de pruebas de Caja Negra, con el que se logró probar cada uno de los elementos que componen la interfaz, permitiendo así validar la solución propuesta.

CONCLUSIONES

Después de haber realizado la investigación y llevar a cabo el desarrollo del software que actualmente se encuentra en la etapa de pruebas, se ha obtenido un compilador para la manipulación de los Mapfile. Con esta herramienta se logró reducir el trabajo de las personas encargadas de configurar el Mapfile. La herramienta creada permitió reconocer todas las estructuras contenidas dentro del extenso volumen de información del fichero (.map).

El estudio de las fases del proceso de compilación, así como de las herramientas de apoyo en el desarrollo de compiladores ha permitido contar con una base sólida para el desarrollo del sistema. Además se puede contar con la documentación técnica del proceso de desarrollo, la cual permite un mejor entendimiento del compilador desarrollado, y puede ser utilizada como guía para futuras actualizaciones o para continuar el desarrollo del sistema, llevando a cabo la implementación de nuevas funcionalidades.

La generación por parte del compilador de un árbol de sintaxis abstracta, le permitirá a la herramienta generar múltiples salidas (almacenar en una base de datos, exportar a diferentes formatos: .map, GML), dándole así la posibilidad a LiberMap de importar un Mapfile completamente, eliminando las restricciones de la Mapscript en cuanto al acceso a sus atributos.

RECOMENDACIONES

- ✓ Incorporar el soporte para el manejo de ficheros de extensión **.lay** y **.sym**.
- ✓ Implementar la exportación a formato GML de un *mapfile* a partir del Árbol de Sintaxis Abstracta.
- ✓ Continuar perfeccionando la estrategia de gestión de errores del compilador.

Referencias Bibliográficas

1. Martínez, Yusnier Valle. ***Presentación de Libermap.*** **Cuba : s.n., 2010.**

2. ***Sistemas de Información Geográfica.*** **2007.**

3. Ramon, M. ***Sistema de Información Geográfica, Aplicaciones y Métodos Analíticos.*** **Cuba : s.n., 2005.**

4. ***CGI MapServer.*** Jeff McKenna, Daniel Morissette, Frank Koormann. **Estados Unidos : s.n., 2004.**

5. Membrides, Antonio. ***Fundamentos del MapServer, Mapscript, PosGIS y su integración.*** **Cuba : s.n., 2010.**

6. Valmarcatastral. **Valmarcatastral. [En línea] [Citado el: 20 de Noviembre de 2010.] http://valmarcatastral.blogspot.com/2010/06/servicios-de-mapas-en-la-web-3.html.**

7. ***Conferencia 1 : Introducción a las técnicas de compilación.*** Programación, Departamento de. **Cuba : s.n., 2010.**

8. Diaz, Emiliano Llano. ***Análisis y Diseño de Compiladores.*** **Mexico : Exa ingeniería SA, 2002.**

9. Ullman, Alfred V. Aho y Jeffrey D. ***Compiladores Principios,Técnicas y Herramientas.*** **Estados Unidos : s.n., 2005.**

10. ***Conferencia 7 :Árbol de Sintaxis Abstracta (AST). Generación del AST .*** Programación, Departamento de. **Cuba : s.n., 2010.**

11. Duque, Raúl González. ***Python para todos.***

12. James Rumbaugh, Ivar Jacobson , Grady Booch. **El Lenguaje Unificado de Modelado .Manual de Referencia. Estados Unidos : Addison Wesley, 1998.**

13. UCI. **Entorno Virtual de Aprendizaje. [En línea] [Citado el: 2 de Mayo de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14095.**

14. Gracia, Joaquin. ***IngenieroSoftware.*** **[En línea] 27 de Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.**

Bibliografía

1. Martínez, Yusnier Valle. ***Presentación de Libermap.*** **Cuba : s.n., 2010.**

2. ***Sistemas de Información Geográfica.*** **2007.**

3. Ramon, M. ***Sistema de Información Geográfica, Aplicaciones y Métodos Analíticos.*** **Cuba : s.n., 2005.**

4. ***CGI MapServer.*** Jeff McKenna, Daniel Morissette, Frank Koormann. **Estados Unidos : s.n., 2004.**

5. Membrides, Antonio. ***Fundamentos del MapServer, Mapscript, PosGIS y su integración.*** **Cuba : s.n., 2010.**

6. Valmarcatastral. **Valmarcatastral. [En línea] [Citado el: 20 de Noviembre de 2010.] http://valmarcatastral.blogspot.com/2010/06/servicios-de-mapas-en-la-web-3.html.**

7. ***Conferencia 1 : Introducción a las técnicas de compilación.*** Programación, Departamento de. **Cuba : s.n., 2010.**

8. Diaz, Emiliano Llano. ***Análisis y Diseño de Compiladores.*** **Mexico : Exa ingeniería SA, 2002.**

9. Ullman, Alfred V. Aho y Jeffrey D. ***Compiladores Principios,Técnicas y Herramientas.*** **Estados Unidos : s.n., 2005.**

10. ***Conferencia 7 :Árbol de Sintaxis Abstracta (AST). Generación del AST .*** Programación, Departamento de. **Cuba : s.n., 2010.**

11. Duque, Raúl González. ***Python para todos.***

12. James Rumbaugh, Ivar Jacobson , Grady Booch. **El Lenguaje Unificado de Modelado .Manual de Referencia. Estados Unidos : Addison Wesley, 1998.**

13. UCI. **Entorno Virtual de Aprendizaje. [En línea] [Citado el: 2 de Mayo de 2010.] http://eva.uci.cu/mod/resource/view.php?id=14095.**

14. Gracia, Joaquin. ***IngenieroSoftware.*** **[En línea] 27 de Mayo de 2005. http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.**

15. Allegue, Facundo G. y Bugaletto, Lic. Guillermo. **UML Lenguaje Unificado de Modelamiento. [En línea] 1.4, Septiembre de 2001. http://www.neuronsrl.com.ar/training/uml/uml_despliegue.html.**

16. UCI. ***Entorno Virtual de Aprendizaje.*** **[En línea] 2010. http://eva.uci.cu/mod/resource/view.php?id=14075.**

17. Olaya, Victor y Luaces, Miguel R. ***Sistemas de Información Geográfica.*** **Primera edición. 2007.**

18. ***El Lenguaje Unificado de Modelado (UML).*** Hernández Orallo, Enrique.

19. Orallo, Enrique Hernández. ***El Lenguaje Unificado de Modelado (UML).***

20. Garcia, Fernando. ***Arquitectura 3 capas.*** **2009.**

21. CHACÓN, JULIO CÉSAR RUEDA. ***APLICACIÓN DE LA METODOLOGÍA RUP PARA EL ESTÁNDAR J2EE.*** **Guatemala : s.n., marzo de 2006.**

22. Pressman, Roger. ***Ingeniería de Software. Un enfoque práctico. Vol I, vol II.*** **La Habana, Cuba : Editorial Félix Varela, 2004.**

23. jacobson, booch y rumbaugh. ***El proceso unificado de desarrollo de software.*** **2000.**

24. Conceptos Básicos del MapServer. 2007.

25. MapServer. [En línea] http://MapServer.gis.umn.edu/.

26. DÍAZ, ALVARO ANTONIO PENROZ. ***Graphical User Interface (GUI) para el programa servidor de mapas MapServer.*** **2005.**

27. Dieguez, N. ***Impacto de los SIG en la sociedad.*** **Cuba : s.n., 2003.**

28. ***MapFile Reference.*** Doyon, Jean-Francois. **Estados Unidos : s.n., 2005.**

29. ***MapServer y su aplicacion a SIG.*** Jaramillo, Victor H. Gonzalez. **2005.**

30. Kropla, Bill. ***MapServer,Open Sourse GIS Development.*** **Estados Unidos : s.n., 2005.**

31. Larman, Craig. ***UML y Patrones Introducción al análisis y diseño orientado a objetos.*** **México : PRENTICE HALL, 1999. 970-17-0261-1.**

32. ***Conferencia 5 : Fase de analisis sintactico.*** Programación, Departamento de. **Cuba : s.n., 2010.**

33. Salinas, J. ***Información Geográfica, Software Libre e Infraestructuras de Datos.*** **Cuba : s.n., 2007.**

34. Open Geospatial Consorcium. [En línea] [Citado el: 17 de Noviembre de 2010.] http://www.opengeospatial.org/.

35. Wirth, Niklaus. ***Compiler Construction.*** **Zürich : Addison-Wesley, 2005.**

36. Grune, Dick. ***Parsing.*** **Amsterdam : s.n., 1998.**

37. Suomitech. [En línea] [Citado el: 23 de Noviembre de 2010.] http://www.suomitech.com/por_que_software_libre.

38. Charles Donnelly, Richard Stallman. **Wikilearning. [En línea] [Citado el: 5 de Noviembre de 2010.] http://www.wikilearning.com/tutorial/manual_de_bison-el_generador_de_analizadores_sintacticos_compatible_con_yacc/9576-1.**

39. James Rumbaugh, Ivar Jacobson , Grady Booch. ***El Lenguaje Unificado de Modelado .Manual de Referencia .*** **Estados Unidos : Addison Wesley, 1998.**

40. Schmuller, Joseph. **Aprendiendo UML en 24 horas. Mexico : Pearson Educacion, 200.**

41. ***Aplicación de Sistemas de Información Geográfica en Cuba.*** Dr. Batista Silva y José Luis. **[ed.] José Luis Dr. Batista Silva. Noviembre de 2005, Revista Internacional de Ciencias de la Tierra. ISSN: 1.131-9.100.**

Glosario de Términos

CGI : (*Common Gateway Interface*) método para la transmisión de información hacia un compilador instalado en el servidor. Su función principal es la de añadir una mayor interacción a los documentos web que por medio del HTML se presentan de forma estática.

UML: (*Unified Modeling Language*) Lenguaje Unificado de Modelado, es un lenguaje que proporciona un vocabulario y reglas para permitir una comunicación.

Lex: es un programa que genera analizadores léxicos ("scanners" o "lexers"). Se utiliza comúnmente con el generador de análisis sintáctico yacc.

Yacc: (*Yet Another Compiler-Compiler*) se trata de un generador de analizadores sintácticos, común en los sistemas Unix.

TIC: (Tecnologías de la Informática y las Comunicaciones) son herramientas que amplían nuestras capacidades físicas y mentales así como las posibilidades de desarrollo social.

GRASP: (*General Responsibility Assignment Software Patterns*) en español patrones generales de software para asignar responsabilidades, el nombre se eligió para indicar la importancia de captar (*grasping*) estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

RAM :(*Random Access Memory Module*) se compone de uno o más chips y se utiliza como memoria de trabajo para programas y datos. Es un tipo de memoria temporal que pierde sus datos cuando se queda sin energía (por ejemplo, al apagar la computadora), por lo cual es una memoria volátil.

OMG: (*Object Management Group*) es un grupo destinado originalmente a establecer normas para distribuir sistema orientado a objetos y ahora se centra en el modelado (programas, sistemas y procesos de negocio).

LR: Constituye una de las más poderosas familias de algoritmos de análisis sintáctico para gramáticas independientes del contexto.

LALR: Intenta construir un árbol de análisis sintáctico, empezando desde la raíz y descendiendo hacia las hojas. Lo que es lo mismo que intentar obtener una derivación por la izquierda para una cadena de entrada, comenzando desde la raíz y creando los nodos del árbol en orden previo.

RUP: (*Rational Unified Process* o Proceso Unificado de Desarrollo de Software) Su objetivo es garantizar la producción de alta calidad de software que satisface las necesidades de sus usuarios finales, en un previsible calendario y el presupuesto.

GNU/GPL: Licencia Pública General de GNU o más conocida por su nombre en inglés GNU General *Public License* o simplemente su acrónimo del inglés GNU GPL, es una licencia creada por la *Free Software Foundation* a mediados de los 80, y está orientada principalmente a proteger la libre distribución, modificación y uso de software. Su propósito es declarar que el software cubierto por esta licencia es software libre y protegerlo de intentos de apropiación que restrinjan esas libertades a los usuarios.