UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS
Facultad 9

Diseño de la arquitectura base del catálogo de mapas LiberMaps.

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

AUTOR: Lilianne Martínez Ledea

TUTOR: MSc. Yusnier Valle Martínez

CO-TUTOR: Ing. Annabell Schelton Lima

Ciudad de La Habana, Mayo de 2010

Agradecimientos

Mi primer pensamiento en estos momentos es para mi mamita querida. Eres lo más importante y lo que más amo en esta vida. Ahora es mi turno de repetir lo que escribiste en tu tesis cuando apenas tenía 2 años: eres el motor impulsor que me renueva las fuerzas cuando estas parecen haberse agotado. Mami, eres mi razón de ser, mi mayor orgullo, mi fuente de inspiración. Te debo todo lo que soy y seré. El haber llegado hasta este punto es fruto de tu dedicación y esfuerzo. Muchas gracias mamita. Te amo.

A mi hermanita Laura, por cuidar de nuestra mami estos 5 años que he estado lejos de casa.

A mis abuelos: mima, mamá Carmita, papá Pedro y especialmente a pipo, que no pudo verme con el título de Ingeniera y que se me fue sin que pudiera decirle por última vez lo mucho que lo quiero.

A todos y cada uno de mis familiares: mi papá, mis tías, tíos y mis primas queridas.

A mi novio David, por todo su apoyo y amor. Gracias por soporta heroicamente mis malacrianzas, por hacerme creer en mí misma y darme la seguridad de que podía lograrlo en esos momentos de desesperación que surgieron durante el desarrollo de la tesis.

A Mita y Lulú, por haber sido amigas, psicólogas, consejeras y hasta doctoras. No logro imaginar mi tiempo en la UCI sin ustedes. No importa lo que nos depare el futuro ni que nuestras vidas tomen rumbos distintos, nada cambiará. Somos BFF. A Luis también, por supuesto, se que aguantarme no es una tarea fácil, jeje… Gracias a los 3 por estar ahí.

A mis amigos de Jiguaní, pues son una parte muy importante de mi vida.

A mami Ana, por ser mi mamá acá en Occidente y por hacerme sentir siempre en casa. Has hecho un excelente trabajo con David y gracias a eso tengo un novio maravilloso.

A mi tutor Valle y mi cotutora Annabell, por ayudarme y guiarme siempre, muchísimas gracias.

A todos mis amigos y amigas de la UCI, pues con ellos he compartido estos 5 años inolvidables.

Debido al creciente auge de los Sistemas de Información Geográfica (SIG) así como la importancia que se le concede a los mismos, se desea contar con herramientas cubanas que posibiliten el manejo de la información georeferenciada. Es por ello que se decide crear en el Departamento de Geoinformática de la Facultad 9 de la Universidad de Ciencias Informáticas, la plataforma GeneSIG para la futura creación de este tipo de sistemas. Se hace necesario además contar con alguna aplicación que brinde servicios de catálogos de mapas en la web y que complemente dicha plataforma. Es con este objetivo que se ha concebido el surgimiento de LiberMaps.

El presente trabajo de diploma pretende realizar la propuesta arquitectónica de esta aplicación que estará basada en tecnologías libres. Se ha llevado a cabo un profundo análisis de todos los elementos necesarios para diseñar una arquitectura de software, los principales conceptos asociados, estilos arquitectónicos y patrones de diseño. Se tuvieron en cuenta además aspectos fundamentales como las herramientas, tecnologías, lenguajes de programación y metodologías. Con todos estos elementos, se realizó una propuesta de lo que se considera es lo más factible para utilizar en el desarrollo del catálogo.

Finalmente se definió la línea base de la arquitectura para LiberMaps, teniendo en cuenta que la misma fuera la más adecuada para el sistema a desarrollar, permitiéndole cumplir con los requisitos establecidos.

Para comprobar lo anteriormente planteado se evaluó la propuesta arquitectónica verificando si cumplía con los atributos de calidad definidos, logrando de esta forma obtener una arquitectura robusta y flexible para el catálogo de mapas LiberMaps de la plataforma GeneSIG.

Palabras claves:

- Sistema de Información Geográfica
- Arquitectura de Software
- Estilos Arquitectónicos
- Patrones de Diseño

INTRODUCCIÓN

Un SIG es una integración muy organizada de software, hardware y datos geográficos, diseñado para capturar, analizar, almacenar, manipular y desplegar la información geográficamente referenciada. Estos deben ser capaces de ubicar un objeto determinado en el espacio; de encontrar donde está un cuerpo con respecto a otro; de brindar información sobre su perímetro, área y volumen; de encontrar el camino mínimo de un punto a otro, así como la generación de modelos a partir de fenómenos o actuaciones simuladas **(Bravo, 2000)**. Estos sistemas son muy utilizados en la actualidad en distintos campos, ya sea de forma directa (servicio de información para el transporte urbano, turismo, etc.) o indirecta (control epidemiológico, defensa civil, etc.).

La Universidad de las Ciencias Informáticas se ha sumado al uso e implementación de esta tecnología. Surgiendo En el Departamento de Geoinformática de la Facultad 9 se ha creado el proyecto GeneSIG, que ha realizado una plataforma con el mismo nombre para la futura creación de SIG, la cual cuenta con las funcionalidades comunes de estos sistemas que resultan esenciales en el comportamiento básico de los mismos. Paralelo al surgimiento de esta plataforma se ha concebido la creación de LiberMaps, un sistema que ofrece servicios de catálogo de mapas y que complementa a GeneSIG. Esta aplicación estará basada en herramientas y tecnologías libres y brindará servicios de catálogo de mapas en la web, permitiendo tener un control sobre el nivel de acceso a la información y gestionar las propiedades de los mapas en función de las necesidades del usuario.

El principal objetivo del desarrollo de dicha plataforma es poder contar con herramientas cubanas que faciliten el manejo de la información georeferenciada, además de contribuir con las entidades que están inmersas en un proceso de informatización y migración hacia el software libre, logrando una mayor eficiencia en este sentido. Esto es posible fundamentalmente debido a las funcionalidades que el sistema ofrece a los usuarios y que contribuyen a que los procesos que se realicen a través de su uso disminuyan en costo y tiempo.

Por todo lo expuesto anteriormente se hacía necesario el diseño de una arquitectura de software con las características necesarias que le permitieran al sistema cumplir con los requerimientos establecidos.

Contando con todos los elementos descritos, se identificó como **Problema a resolver:** ¿Cuáles son los principales componentes y sus relaciones orientados a satisfacer los requerimientos funcionales y no funcionales de LiberMaps?

Seleccionando como **Objeto de estudio:** El proceso de diseño de una arquitectura basada en tecnologías de software libre. Además de tener como **Campo de acción:** El diseño y descripción de la arquitectura base para el desarrollo del catálogo de mapas LiberMaps.

Teniendo en cuenta el problema planteado se define como **Objetivo General:** Diseñar una arquitectura para el módulo de catálogo de mapas LiberMaps de la plataforma GeneSIG.

En función del objetivo anteriormente planteado se propone la siguiente **Idea a defender:** Con el diseño de una arquitectura robusta y flexible se garantizará una correcta organización del módulo de catálogo de mapas LiberMaps de la plataforma GeneSIG.

Esperando obtener como **Posibles resultados:** Una propuesta arquitectónica para el módulo de catálogo de mapas LiberMaps de la plataforma GeneSIG, desarrollado por el Grupo de Sistemas de Información Geográfica.

CAPÍTULO 1: Fundamentación Teórica.

1.1. Introducción

En este capítulo se realiza un breve análisis del surgimiento de la arquitectura del software (en lo adelante AS) y su definición, así como los principales conceptos asociados al tema, además de una descripción de las principales herramientas y tecnologías imprescindibles para desarrollar una buena arquitectura.

1.2. Surgimiento de la Arquitectura de Software. Definición.

La Arquitectura de Software remonta sus antecedentes hasta la década de 1960, sin embargo, su historia no ha sido tan continua como la del campo en el que se enmarca: la Ingeniería de Software. Se puede decir que la Arquitectura de Software está implicada en la Ingeniería de Software, puesto que se enmarca dentro de sus 3 grandes temas fundamentales: Patrones, Métodos Heterodoxos (Métodos Ágiles como eXtreme Programming, Scrum, Crystal) y Arquitectura de Software.

Si bien se reconoce que Edsger Dijkstra fue quien, en un principio, propuso que se estableciera una estructuración en el proceso de desarrollo de software antes de comenzar a programar; fue sin dudas, en los estudios realizados por Dewayne E. Perry y Alexander L. Wolf, donde aparece por primera vez la expresión “arquitectura de software” en el sentido con que se conoce en la actualidad, anticipando además cuando ocurriría el auge de la AS: La década de 1990, será la década de la arquitectura de software.(…) Es tiempo de re-examinar el papel de la arquitectura de software en el contexto más amplio del proceso de software y de su administración, así como señalar las nuevas técnicas que han sido adoptadas **(Wolf, 1992)**.

Se plantea que, hasta el momento, ninguna definición de AS es respaldada en su totalidad por los arquitectos. El número de definiciones circulantes amenaza con alcanzar ya los 4 dígitos. Es por esto que se ha decidido mencionar a continuación algunas de las más reconocidas.

Tal es el caso de la de Paul Clements: La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del

sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones **(Clements, 1996)**.

En otra definición David Garlan establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño **(Garlan, 2000)**.

Por otro lado, para Bass, la AS de un sistema es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos **(Pressman, 2002)**.

No obstante, la definición más oficial de AS se ha acordado que sea la que ofrece el documento de IEEE Std 1471-2000, adoptada también por Microsoft: La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.

En resumen, se puede decir que la AS indica la estructura, funcionamiento e interacción entre las partes del software, es decir, es el diseño de más alto nivel de la estructura de un sistema.

1.3. Estilos arquitectónicos

Los patrones coronan una práctica de diseño que se origina antes que la arquitectura de software se distinguiera como discurso en perpetuo estado de formación y proclamara su independencia de la ingeniería en general y el modelado en particular. Los estilos, en cambio, expresan la arquitectura en el sentido más formal y teórico **(Kicillof, 2004)**.

Robert Allen y David Garlan asemejan los estilos arquitectónicos a descripciones informales de arquitecturas basadas en una colección de componentes computacionales, junto a una colección de conectores que describen las interacciones entre los componentes **(Garlan, 1996)**.

Por su parte, Pressman considera que un estilo describe una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema **(Pressman, 2002)**.

Se puede concluir que un estilo define una forma de organización arquitectónica y conjuga elementos (componentes), conectores, configuraciones y restricciones.

1.4. Algunos estilos arquitectónicos

Los estilos que se mencionan a continuación no son todos los que existen actualmente, pero sí los más significativos y los más usados.

- **Estilos de Flujo de Datos**
  - Tubería y filtros
- **Estilos Centrados en Datos**
  - Arquitecturas de Pizarra o Repositorio
- **Estilos de Llamada y Retorno**
  - Modelo Vista Controlador (MVC)
  - Arquitecturas en Capas
  - Arquitecturas Orientadas a Objetos
  - Arquitecturas Basadas en Componentes
- **Estilos de Código Móvil**
  - Arquitecturas de Máquinas Virtuales
- **Estilos Peer-to-Peer**
  - Arquitecturas Basadas en Eventos
  - Arquitecturas Orientadas a Servicios

1.4.1. Descripción de los estilos arquitectónicos.

1.4.1.1. Estilos de Flujo de Datos

Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote **(Kicillof, 2004)**.

Tubería y filtros:

Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo, Figura 1. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. (…) Debido a su simplicidad y su facilidad para captar una funcionalidad, es una

arquitectura mascota cada vez que se trata de demostrar ideas sobre la formalización del espacio de diseño arquitectónico **(Doberkat, 2002)**.

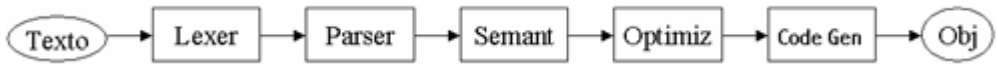

Figura 1 -- Compilador en tubería-filtro (Doberkat, 2002)

El estilo tubería-filtros se concibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. Esto puede verse claramente en un ejemplo de procesamiento de datos. Es decir, el cliente hace una petición, esta se valida; un servidor web toma el elemento requerido de la base de datos, lo convierte a HTML y lo muestra en pantalla.

1.4.1.2. Estilos Centrados en Datos

Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se basen en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra **(Reynoso, 2004)**.

Arquitecturas de Pizarra o Repositorio:

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él como se muestra en la Figura 2 **(Shaw, 1996)**.

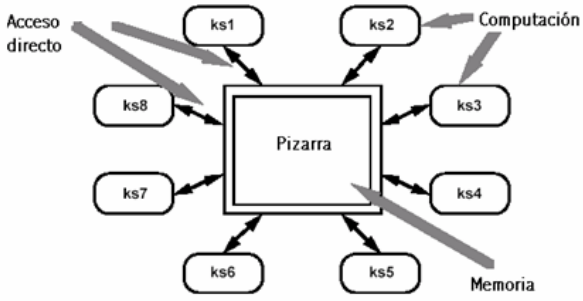

Figura 2 – Pizarra (Shaw, 1996)

Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. También se han implementado estilos de este tipo en procesos en lotes de base de datos y ambientes de programación organizados como colecciones de herramientas en torno a un repositorio común. **(Shaw, 1994)**.

Los estilos de pizarra se utilizan principalmente en el campo de la robótica, en programación evolutiva, inteligencia artificial, gramáticas complejas, etc. Es decir, un sistema de pizarra se implementa cuando existen problemas para los que no se encuentra una solución analítica.

1.4.1.3. Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de esta familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas **(Kicillof, 2004)**.

Modelo Vista Controlador (MVC):

Al estilo MVC en ocasiones suele definírsele como un patrón de diseño y consiste en separar el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes **(Burbeck, 1992)**:

- Modelo: Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada. El modelo debe de preservar la integridad de los datos.
- Vista: Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.
- Controlador: Recibe las entradas, usualmente como eventos, e interpreta las operaciones del usuario; codificando los movimientos, pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a solicitudes de servicio para el modelo o la vista. Es el que debe de controlar los eventos.

Arquitecturas en Capas:

Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En un estilo en capas, los conectores se definen mediante los protocolos que determinan las formas de la interacción **(Shaw, 1994)**.

Este estilo incluye en sus restricciones topológicas, una limitación que exige que cada capa deba operar solamente con las capas adyacentes, así como los elementos de una capa deben entenderse sólo con elementos de la misma. Algunos ejemplos de este estilo son muchos de los protocolos de comunicación en capas, siendo el ejemplo más característico el modelo OSI con sus siete niveles.

Arquitecturas Orientadas a Objetos:

Este tipo de arquitecturas también es conocida como: Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos.

Sus componentes son los objetos, o de forma más acertada, las instancias de los tipos de dato abstractos. Garlan y Mary Shaw definen que los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación y la representación interna de un objeto no es accesible desde otros objetos **(Shaw, 1994)**.

Arquitecturas Basadas en Componentes:

En la actualidad se trata mucho el tema de la reutilización, ya sea de módulos o partes de software existentes, tratando así de optimizar el proceso de desarrollo de nuevas aplicaciones. Dentro de este contexto, el término componente es muy empleado, pues en los procesos de ingeniería, dichas partes de software que pueden ser útiles para la creación de aplicaciones, son conocidas como: componentes de software.

Existen numerosas definiciones de componentes, en este caso se enuncia la que brinda Clemens Alden Szyperski: un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas **(Szyperski, 1998)**.

El hecho de que sea una unidad de composición y no de construcción equivale a que no es preciso confeccionarla, se puede adquirir ya hecha, o se puede construir para que otras aplicaciones la utilicen.

1.4.1.4. Estilos de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico **(Kicillof, 2004)**.

Arquitectura de Máquinas Virtuales:

Este tipo de arquitectura se le conoce también como intérpretes basados en tablas y esto se debe al hecho de que todo intérprete incluye una máquina virtual implementada.

Se puede decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución (o registro de activación). La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. De este modo, un intérprete posee por lo general cuatro componentes: (1) una máquina de interpretación que lleva a cabo la tarea, (2) una memoria que contiene el seudo-código a interpretar, (3) una representación del estado de control de la máquina de interpretación, y (4) una representación del estado actual del programa que se simula **(Kicillof, 2004)**.

Dado que hasta cierto punto las máquinas virtuales no son una opción sino que devienen inevitables en ciertos contextos, no existen informes en la literatura especializada identificando sus ventajas y deméritos.

1.4.1.5. Estilos Peer-to-Peer

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente **(Kicillof, 2004)**.

Arquitecturas Basadas en Eventos:

Conocida también como invocación implícita, este tipo de arquitectura se relaciona con sistemas basados en actores y redes de conmutación de paquetes.

Desde el punto de vista arquitectónico, los componentes de un estilo de invocación implícita son módulos cuyas interfaces proporcionan tanto una colección de procedimientos como un conjunto de eventos. Los procedimientos se pueden invocar a la manera usual en modelos orientados a objeto, o mediante el

sistema de suscripción que se ha descrito.(…) El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia, en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos, y en editores sintácticamente orientados para proporcionar verificación semántica incremental **(Kicillof, 2004)**.

Arquitecturas Orientadas a Servicios:

Las Arquitecturas Orientadas a Servicios (SOA, por sus siglas en inglés: Service Oriented Architecture) tienen por componentes, como su nombre lo indica, los servicios.

Un servicio no es más que una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas **(Kicillof, 2004)**.

Estos servicios están débilmente acoplados por lo que la funcionalidad de cada uno puede ser modificada sin temor a afectar cualquier otro servicio. Para establecer comunicación entre ellos, se basan en una definición formal que es independiente tanto de la plataforma como del lenguaje de programación o de la tecnología de desarrollo. Esto permite que los componentes de software que se desarrollen sean altamente reusables, pues un servicio desarrollado en un tipo de lenguaje puede ser utilizado sin problemas en cualquier otro.

1.5. Patrones arquitectónicos

Un patrón es definido por Buschmann como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución.

En líneas generales, un patrón sigue el siguiente esquema:

- Contexto: Es una situación de diseño en la que aparece un problema de diseño
- Problema: Es un conjunto de fuerzas que aparecen repetidamente en el contexto
- Solución: Es una configuración que equilibra estas fuerzas. Ésta abarca:
    - Estructura con componentes y relaciones
    - Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc **(Buschmann, 1996)**.

Es decir, un patrón puede ser visto como una solución a un problema determinado, una codificación de algo específico basado en un conocimiento acumulado por la experiencia.

Los patrones arquitectónicos según Buschmann expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software **(Camacho, 2004)**.

Los patrones son similares a los estilos en la medida en que definen una familia de sistemas de software que comparte características comunes, pero también difieren en dos importantes aspectos. Un estilo representa específicamente una familia arquitectónica, construida a partir de bloques de construcción arquitectónicos, tales como los componentes y los conectores. Los patrones, en cambio, atraviesan diferentes niveles de abstracción y etapas del ciclo de vida partiendo del análisis del dominio, pasando por la arquitectura de software y yendo hacia el nivel de los lenguajes de programación **(Reynoso, 2004)**.

Partiendo del nivel de abstracción de cada patrón, estos pueden dividirse según lo planteado por Buschmann en las categorías de Patrones de Diseño, Patrones de Arquitectura e Idiomas.

1.5.1. Patrones de Diseño

Buschmann plantea que un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular **(Buschmann, 1996)**.

Los patrones de diseño son menores en escala que los patrones arquitectónicos y por lo general tienden a ser independientes de los lenguajes y paradigmas de programación. Estos pueden dividirse en diferentes grupos, ya sea según su propósito o su ámbito.

Según su propósito se encuentran:

- Patrones de Creación.
- Patrones Estructurales.
- Patrones de Comportamiento.

Según su ámbito se agrupan en:

- De clase: Basados en la herencia de clases.
- De objeto: Basados en la utilización dinámica de objetos.

1.5.1.1. Patrones generales de software para asignar responsabilidades (GRASP[1])

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Aunque algunos consideran que en realidad son un conjunto de "buenas prácticas" recomendables a usar a la hora de diseñar el software, en esta investigación se incluyen dentro de los patrones de diseño, luego de analizar que los mismos sirven de guía para encontrar los patrones de diseño y ayudan además a entender el diseño de objeto esencial y aplicar el razonamiento para el diseño de una forma sistemática, racional y explicable.

Ejemplos de patrones GRASP.

Existen diferentes patrones de este tipo, a continuación se hace referencia a los principales:

- Experto: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos).
- Creador: Se asigna la responsabilidad de que una clase B cree un objeto de la clase A.
- Alta cohesión: Cada elemento del diseño debe realizar una labor única dentro del sistema.
- Bajo acoplamiento: Deben existir pocas dependencias entre las clases.
- Controlador: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.

Además de los ya mencionados existen 4 patrones adicionales:

- Fabricación Pura.
- Polimorfismo.
- Indirección.
- No hables con extraños.

1.5.1.2. Patrones Gang of Four (GoF)

[1] General Responsibility Assignment Software Patterns

Estos patrones son reconocidos dentro del campo del desarrollo de software en el libro Design Patterns, escrito por los que comúnmente se conocen como GoF (Gang of Four o "pandilla de los cuatro": Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides. Fueron recopilados y documentados 23 patrones los cuales se clasificaron en tres grandes categorías. A continuación se mencionan algunos de los más empleados.

Patrones creacionales (Gamma, 1995)

- Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- Factory Method (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado.
- Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Patrones Estructurales (Gamma, 1995)

- Composite (Objeto compuesto): Permite tratar objetos compuestos como si se tratase de un objeto simple.
- Decorator (Envoltorio): Añade funcionalidad a una clase dinámicamente.
- Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.

Patrones de Comportamiento (Gamma, 1995)

- Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

- State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

1.6. Metodologías de Desarrollo de Software.

El desarrollo de software es, sin dudas, una tarea ardua, por lo que se hacía necesario imponer cierta disciplina sobre todo el proceso para lograr mayor eficiencia en el mismo. Como respuesta a este problema es que surgen las metodologías, las cuales definen Quién está realizando Qué, Cuándo lo lleva a cabo y Cómo lo está haciendo.

La aplicación de metodologías tradicionales, sin embargo, no brindan una solución óptima para los proyectos donde los requisitos no son reconocidos con exactitud o el entorno es volátil, pues no están concebidas para trabajar con incertidumbre.

Es por ello que surgieron otras metodologías que tratan de adaptarse a la realidad del desarrollo de software, las cuales son conocidas como metodologías ágiles.

1.6.1. Metodologías robustas.

1.6.1.1. Proceso Unificado de Desarrollo (RUP)

RUP es una metodología orientada a objetos basada en el Lenguaje Unificado de Modelado (UML), cuyo ciclo de vida se caracteriza por:

- Dirigido por casos de usos.
- Centrado en la arquitectura.
- Iterativo e incremental.

Sus principales elementos son:

- Trabajadores (Quién)
- Actividades (Cómo)
- Artefactos (Qué)

- Flujo de actividades (Cuándo)

En esta metodología se han agrupado las actividades en 9 flujos de trabajo, siendo los 6 primeros flujos de ingeniería y los 3 últimos de apoyo. Además presenta 4 fases:

- Inicio
- Elaboración
- Construcción
- Transición

1.6.2. Metodologías ágiles.

1.6.2.1. Programación Extrema (XP)

Ideada por Kent Beck, XP es un proceso de desarrollo de software diferente al convencional. Se caracteriza por ser flexible, poseer bajo riesgo y eficiente.

Se centra en potenciar las relaciones entre cliente y equipo de desarrollo. Trata de dar al cliente el software que necesita y cuando lo necesita. Así como potenciar al máximo el trabajo en equipo.

1.7. Lenguaje Unificado de Modelado (UML)

Es un lenguaje que permite visualizar, especificar, construir y documentar los artefactos de los sistemas de software **(Larman, 1999)**.

Precisamente esta definición aporta las principales funciones de UML:

- Visualizar: Expresa de forma gráfica un sistema de forma que sea fácil de comprender.
- Especificar: Especifica cuáles son las características de un sistema antes de su construcción.
- Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.
- Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión.

Permite la modelación de sistemas con tecnología orientada a objetos. Un modelo UML describe lo que supuestamente hará un sistema pero no dice como implementar dicho sistema. En él se identifican:

- Elementos: estructurales, comportamiento, agrupación, anotación.
- Relaciones: dependencia, asociación, generalización/especialización, realización.
- Diagramas: clases, objetos, casos de uso, secuencia, colaboración, estados, actividades, componentes, despliegue.

1.8. Herramientas de Ingeniería de Software Asistidas por Computadora (CASE)

Las herramientas CASE (Computer-Aided Software Engineering) permiten automatizar el desarrollo de software, lo que implica un aumento de la productividad y la calidad de este proceso. Su objetivo fundamental es proveer un lenguaje para describir el sistema general que sea lo suficientemente explícito para generar todos los programas necesarios.

1.8.1. Rational Rose Enterprise Edition

Rational Rose Enterprise Edition es el producto más completo de la familia Rational Rose, incluyendo, al igual que el resto de los productos de esta familia, soporte UML.
Entre sus principales características se encuentran:

- Proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente.
- Provee visualización, modelación y herramientas para el desarrollo de aplicaciones Web.
- Ofrece habilidades de análisis de calidad de código y generación del código.
- Permite generar código a partir de modelos Ada, ANSI C++, C++, CORBA, Java/J2EE, Visual C++ y Visual Basic.

1.8.2. Visual Paradigm para UML

Esta herramienta UML profesional ayuda a construir aplicaciones de calidad con mayor rapidez, mejores y a un menor coste. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Proporciona también abundantes tutoriales de UML, demostraciones interactivas y proyectos que empleen UML. Entre sus características principales se encuentran:

- Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación.
- Es multiplataforma.
- Puede integrarse en los principales IDEs de desarrollo.
- Soporta aplicaciones Web.

1.9. Lenguajes de Programación

Un lenguaje de programación permite crear programas para controlar el comportamiento físico y lógico de una máquina. Su estructura está definida por un conjunto de símbolos y reglas sintácticas y semánticas, que le dan significado a sus elementos y sus expresiones. Se pueden encontrar lenguajes de programación que permiten implementar tanto aplicaciones de escritorio como aplicaciones web. Además si se desea desarrollar una aplicación web se deben tener en cuenta tanto los lenguajes del lado del servidor como los del lado del cliente

1.9.1. Lenguajes del lado del servidor

Los lenguajes del lado del servidor no dependen del navegador a utilizar, esto implica que no sean sensibles a posibles cambios en este sentido. Este tipo de lenguajes se ejecutan e interpretan en el propio servidor. Otra de sus características es que el código de los script se almacena en el servidor, este se encarga de ejecutarlo y traducirlo a HTML de forma tal que no sea visible para el cliente, lo que constituye una gran ventaja a la hora de proteger el trabajo de los programadores. Aunque existen varios lenguajes de programación del lado del servidor, tales como Perl, ASP.Net, Phyton, PHP y Ruby, a continuación se brindan características fundamentales de algunos de estos lenguajes.

1.9.1.1. Ruby

El lenguaje Ruby presenta tres cualidades fundamentales: es reflexivo, interpretado y orientado a objetos. Esto posibilita que ofrezca un conjunto de características en tiempo de ejecución que le permite a los programas modificar su estructura dinámicamente. Su sintaxis es sencilla y fácil de comprender. Ruby es multiparadigma, permitiendo programación orientada a objetos, procedural y funcional. Permite realizar varias tareas de forma concurrente pues tiene soporte para hilos de ejecución. Brinda soporte además para la Inyección de Dependencias, que es un patrón de diseño que permite incorporarle objetos a una

clase en lugar de ser la clase la que los cree. Ruby trata todos los elementos como objetos, incluyendo aquellos que otros lenguajes tienen como tipos primitivos (int, bool, etc), por lo que se les puede incluir tanto propiedades como métodos.

1.9.1.2. PHP5

Es un lenguaje de código abierto, multiplataforma, interpretado en el lado del servidor que se utiliza para la creación de páginas web dinámicas. Permite la conexión con la mayoría de los sistemas gestores de bases de datos: MySQL, PostgreSQL, Oracle, MS SQL Server, entre otras.
La versión PHP5 fue lanzada en julio del 2004 y la más reciente es la versión 5.3.1 que salió al mercado en noviembre del 2009. PHP5 presenta un mejor soporte para la Programación Orientada a Objetos y para el gestor de bases de datos MySQL. Además de que posee mejoras de rendimiento y manejo de excepciones.

1.9.1.3. Phyton

Este lenguaje de programación creado en 1990, permite la creación de todo tipo de programas incluyendo los sitios web. Su código no necesita ser compilado, es decir es un lenguaje interpretado. Es un lenguaje multiparadigma:

- Programación orientada a objetos.
- Programación estructurada.
- Programación funcional.
- Programación orientada a aspectos.

Phyton es un lenguaje libre, de código abierto, sencillo y rápido de programar. Entre sus ventajas, además de las ya mencionadas se encuentra el hecho de que es multiplataforma y portable.

1.9.2. Lenguaje del lado del cliente

Los lenguajes del lado del cliente son independientes del servidor, esto posibilita que las páginas puedan guardarse en cualquier sitio. En este sentido es fundamental abordar sobre ECMAScript, un estándar que define un lenguaje de tipos dinámicos y que ha servido de base para otros lenguajes del lado del cliente

como JavaScript y ActionScript. La mayoría de los navegadores incluyen una implementación de este estándar.

1.9.2.1. JavaScript

JavaScript es un lenguaje del lado del cliente que no requiere de compilación, es decir, es interpretado. Ha tenido influencias de múltiples lenguajes (Perl, Phyton) y fue diseñado con una sintáxis similar a la de Java. La principal diferencia radica en que Java es un lenguaje orientado a objetos y JavaScript está basado en prototipos. Es compatible con la mayoría de los navegadores como Netscape, Internet Explorer, Mozilla Firefox, entre otros.
Para que interactúe con una página web evitando incompatibilidades, el World Wide Web Consortium (W3C) diseñó el estándar DOM (Modelo de Objetos del Documento). Finalmente se debe señalar que este lenguaje de scripting es seguro y fiable.

1.10. Framework

En términos generales, se puede decir que un framework es un conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software. Un framework ofrece una guía arquitectónica partiendo del diseño en clases abstractas y definiendo sus responsabilidades y sus colaboraciones. Un desarrollador personaliza el framework para una aplicación particular mediante herencia y composición de instancias de las clases del framework **(Gamma, 1995)**.
Partiendo de este concepto se puede concluir que un framework es la representación de un arquitectura de software que se adapta a las particularidades de un determinado dominio de aplicación, proporcionando una estructura y una metodología de trabajo que van a utilizar las aplicaciones de dicho dominio. Puede incluir además bibliotecas, soporte de programas y un lenguaje que puede ser interpretado entre otros programas, lo que contribuye a desarrollar y unificar los diferentes componentes de un proyecto de software.

1.10.1. Ruby on Rails (RoR)

Ruby on Rails es un framework de código abierto, que está escrito en el lenguaje Ruby y se basa en el estilo arquitectónico Modelo Vista Controlador. Fue lanzado en el 2004 y supuso una revolución en el

desarrollo de las aplicaciones web. Utiliza el patrón de diseño Active Record para gestionar las clases del modelo y el acceso a los datos en la base de datos. Entre sus principales características se encuentran:

- Soporta diferentes gestores de bases de datos.
- Multiplataforma.
- Facilita el envío de correo electrónico a través del módulo Action Mailer.
- Incluye el framework Prototype perteneciente a JavaScript.

1.10.2. Django

Este framework de código abierto está escrito completamente en Phyton. A pesar de que toma ideas del estilo Modelo Vista Controlador, sus desarrolladores no han seguido fielmente los conceptos de este tipo de arquitectura, es por ello que a lo que recibe el nombre de controlador en los frameworks de PHP, que sí se basan en este estilo, en Django lo tratan como vista; nombrando a su vez a la vista del MVC como plantilla. Entre sus principales características se encuentran:

- Presenta un sistema extensible de plantillas basado en etiquetas, con herencia de plantillas.
- Aunque se recomienda utilizar como gestor de base de datos a PostgreSQL, también ofrece soporte para MySQL y SQLite.
- Es multiplataforma.

1.10.3. Frameworks PHP

Los frameworks PHP ofrecen estructuras básicas que se utilizan para construir aplicaciones web, lo que permite que sea mucho más dinámico el desarrollo las mismas. Una de las ventajas que brinda la utilización de estos frameworks es que acelera todo el proceso de creación, pues permite la reutilización de código y reduce también la cantidad de código repetitivo para los desarrolladores.
Estos frameworks se basa en el patrón arquitectónico Modelo Vista Controlador (MVC), el cual separa el proceso de desarrollo de una aplicación por lo que se puede trabajar sobre elementos individuales sin afectar los otros. Precisamente esta propiedad es la que posibilita que la codificación en PHP sea más rápida y menos complicada.

1.10.3.1. CakePHP

Es un framework de desarrollo de aplicaciones web escrito en PHP, que fue creado con base en los conceptos de Ruby on Rails. Por lo que, al igual que RoR, CakePHP facilita al usuario la interacción con la base de datos mediante el uso de Active Record.
A continuación se mencionan algunas de las características del framework CakePHP:

- Presenta un sistema de plantillas rápido y flexible.
- Es compatible con PHP 4 y PHP 5.
- Es multiplataforma.
- Posee validación integrada.

1.10.3.2. Zend Framework

Este framework emplea código orientado a objetos y permite la creación de aplicaciones web con PHP5. Sus componentes siguen los principios de bajo acoplamiento por lo que existe una dependencia muy baja entre ellos. Esto les facilita a los programadores la utilización de los mismos de forma independiente. Implementa MVC y ofrece un gran rendimiento. Algunas de sus características fundamentales son:

- Ofrece robustas clases para autenticación y filtrado de entrada.
- Amplia documentación y pruebas de alta calidad.
- Una abstracción de base de datos fácil de usar.

1.10.3.3. Symfony

Symfony es uno de los frameworks de PHP más populares. Se plantea que es una de las mejores copias creadas para PHP del famoso framework RoR. Symfony toma las mejores ideas de RoR junto con la de muchos otros e incorpora las suyas propias, de ahí que el resultado sea un framework estable, productivo, elegante y muy bien documentado. Este framework sigue la mayoría de mejores prácticas y patrones de diseño para la web.
Este framework de código abierto separa la lógica del negocio, la lógica del servidor y la presentación, pues al igual que el resto de los frameworks PHP se basa en el patrón MVC. Ofrece varias herramientas y clases con el objetivo de reducir el tiempo de desarrollo de una aplicación web compleja. Tiene como ventaja además que automatiza las tareas más comunes, por lo que el desarrollador puede dedicarse por completo a los aspectos específicos de cada aplicación.

Symfony fue diseñado para cumplir una serie de características, algunas de ellas son:

- Es multiplataforma.
- Es compatible con la mayoría de los sistemas gestores de bases de datos: MySQL, Oracle, PostgresSQL, Microsoft SQL Server.
- Utiliza programación orientada a objetos, de ahí la necesidad del uso de PHP5.
- Fácil de extender, lo que permite su integración con las bibliotecas de otros fabricantes.

1.10.3.4. Comparación entre Frameworks PHP

El proveedor francés de software, Clever Age, realizó un profundo estudio sobre los 3 frameworks de PHP vistos anteriormente y arriba a conclusiones en el documento "Libro blanco sobre los frameworks de PHP para empresas". En este estudio se compara además el framework CodeIgniter. A continuación se hace referencia a algunos de los resultados que arrojo dicha investigación, en la que Symfony con un 74, 41 % termina por delante de Zend Framework que obtuvo el 74,10% luego de haber analizado y medido las ventajas que ofrecen cada uno los frameworks estudiados.

Tabla 1 -- Comparación Frameworks PHP (Ravoallan, 2005)

	Zend Framework	Cake PHP	Symfony
Documentación	Documentación detallada presentada bajo la forma de un guía de referencia disponible y descargable en línea	Documentación presentada bajo la forma de un manual accesible únicamente en línea	Documentación importante presentada bajo la forma de un manual disponible en línea y en libro
Facilidad de extensión del código	Posibilidad de utilizar extensiones	Posibilidad de utilizar componentes, modelos y plugins adicionales	Posibilidad de utilizar plugins
Durabilidad	90,00%	80,00%	80,00%
Solución industrializada	64,71%	50,00%	67,65%
Adaptabilidad	100,00%	83,33%	100,00%
Estrategia	41,67%	50,00%	50,00%
Soporte de distintos SGB	Sí	Sí	Sí

1.10.4. Frameworks AJAX[2]

AJAX es una tecnología que permite construir páginas web dinámicas del lado del cliente, incorporando otras tecnologías como JavaScript y XML. En estas páginas la información es leída desde el servidor o enviada a éste a través de peticiones JavaScript, pero se hace necesario algún procesamiento del lado del servidor para que sea posible manejar las peticiones, un ejemplo de esto sería la búsqueda o almacenamiento de información. Esto se logra con la utilización de un framework dedicado a procesar peticiones AJAX, los cuales ayudan a desarrollar aplicaciones web basadas en esta tecnología.

Los frameworks de AJAX pretenden reducir la espera del usuario cuando una página trata de acceder al servidor y proveen funciones asociadas al servidor y del lado del cliente. Es decir, en el lado del cliente ofrecen funciones JavaScript para enviar peticiones al servidor y en el lado del servidor posibilita buscar la información solicitada en las peticiones una vez procesadas las mismas para luego transmitirla al navegador.

Aunque existe una amplia gama de este tipo de frameworks, a continuación se definen algunos de los más utilizados.

1.10.4.1. Dojo

Este framework empleado para construir aplicaciones web que empleen tecnología AJAX, agrupa diversas opciones en una sola biblioteca JavaScript y es compatible con navegadores antiguos. Con Dojo la comunicación entre el navegador y el servidor se realiza mediante una capa de abstracción con la que se pueden emplear distintos tipos de datos. Ofrece además un sistema de paquetes facilitando de esta forma el desarrollo modular.

1.10.4.2. Prototype

Prototype surge a la par del proyecto Ruby on Rails, como una librería que permitía la implementación de AJAX en los sitios web. Este framework escrito en JavaScript permite crear aplicaciones web dinámicas y sencillas. Una de sus ventajas es que es muy útil cuando se pretende desarrollar páginas con un alto grado de interactividad, simplificando una buena parte del trabajo.

[2] AJAX: Asynchronous Javascript And XML

1.10.4.3. Ext JS

Es un framework de JavaScript para desarrollar aplicaciones web interactivas, mediante el uso de tecnologías AJAX, DHTML y DOM. Estaba basado originalmente en YUI (Yahoo User Interface)[3] pero actualmente es independiente del framework que se utilice, incluso puede usarse sin frameworks. Además posibilita la reutilización de código y es adaptable a diferentes frameworks: Prototype, Jquery.

1.11. Sistemas de Gestión de Bases de Datos (SGBD)

Un SGBD es un tipo de software que sirve de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Es un conjunto de programas que permiten la creación de bases de datos asegurando su integridad, confidencialidad y seguridad. Estos sistemas permiten la manipulación de grandes volúmenes de información así como el control de la redundancia.

1.11.1. MySQL

MySQL surge como idea de la empresa opensource MySQL AB con el objetivo de que cumpla el estándar SQL (Lenguaje de Consulta Estructurado). Es un sistema de gestión de base de datos relacional, esto significa que archiva los datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Es multihilo por lo que puede realizar varias tareas a la vez y es también multiusuario, lo que le permite satisfacer de forma simultánea, las necesidades de varios usuarios que comparten los mismos recursos.
Es un software de código abierto por lo que cualquier persona puede usarlo y modificarlo. Funciona además sobre múltiples plataformas.

1.11.2. PostgresSQL

Es un sistema de gestión de base de datos orientada a objetos, libre y multiplataforma. Incluye herencia entre tablas e incorpora una estructura de datos array. Además es altamente extensible. PostgresSQL es un gestor de base de datos objeto-relacional, que utiliza Control de Concurrencia Multi-Versión con el fin de evitar bloqueos innecesarios, esta estrategia permite obtener una mejor respuesta en ambientes de grandes volúmenes.

[3] Una serie de bibliotecas escritas en JavaScript para la construcción de aplicaciones interactivas.

La flexibilidad del API (Interfaz de programación de aplicaciones) de PostgreSQL permite facilitar soporte para el desarrollo con este SGBD en varios lenguajes de programación: Object Pascal, Python, Perl, PHP, Java/JDBC, Ruby, C/C++.
Incluye también una extensión que le añade soporte de objetos geográficos, PostGIS, que permite realizar análisis mediante consultas SQL espaciales o mediante conexión a aplicaciones GIS.

1.12. Entornos de desarrollo integrado

Un entorno de desarrollo integrado (IDE: Integrated Development Environment) es un conjunto de herramientas de programación que conforman un programa informático. Los IDEs pueden utilizarse para un solo lenguaje o para varios, y pueden ser aplicaciones independientes o estar incluidas en otras ya existentes.
Componentes de los IDEs:

- Editor de código
- Compilador
- Intérprete
- Depurador
- Constructor de interfaces gráficas de usuario (GUI)

1.12.1. Zend Studio

Este IDE escrito en Java es un software comercial. Ofrece un entorno de trabajo para realizar aplicaciones web en PHP, donde Zend Studio funciona como la parte cliente y Zend Platform como la parte servidora, ofreciendo soporte para PHP4 y PHP5. Aunque fue diseñado para este lenguaje propiamente, soporta además otros lenguajes como JavaScript, HTML y XML. Otra de sus características es que permite detectar errores de sintaxis en tiempo real y tiene integrado un manual de PHP.

1.12.2. Eclipse

Eclipse es todo un proyecto donde, conjuntamente con la creación del IDE Eclipse se han desarrollado algunos de los plug-in más importantes: JDT para Java y CTD para C/C++. Eclipse es un IDE de código abierto del cual existen versiones instalables para las plataformas más importantes, incluyendo el código fuente y los plug-ins más usados.

Aunque en el desarrollo de este proyecto se emplea mayormente Java y por lo tanto su principal uso es como IDE de Java, se le puede añadir mediante módulos o plug-in soporte para lenguajes adicionales. La arquitectura de plug-ins de Eclipse además de que brinda la posibilidad de integrar varios lenguajes sobre un mismo IDE, permite introducir otras aplicaciones como es el caso de herramientas UML.

1.12.3. Netbeans

La plataforma Netbeans es un proyecto de código abierto que permite desarrollar las aplicaciones mediante módulos. Estos módulos están formados por un archivo Java que les permite interactuar con las APIs de Netbeans y un archivo especial (manifest file) que es el que los identifica como módulos. Las aplicaciones creadas a partir de módulos pueden ser extendidas añadiendo nuevos módulos. Aunque el IDE Netbeans está escrito en Java usando la plataforma Netbeans, puede servir para otros lenguajes de programación.

Netbeans permite crear aplicaciones web con PHP5 y presenta además soporte para el framework Symfony. También posibilita el desarrollo de aplicaciones en Phyton y posee facilidades para la creación de aplicaciones en Ruby y Ruby on Rails.

1.13. Servidores de Mapas

Los servidores de mapas le brindan al usuario la máxima interacción con la información geográfica. De forma tal que el cliente accede a información en su formato original, siendo posible realizar consultas tan complejas como las que haría un SIG.

Un servidor de mapas funciona enviando, a petición del cliente, desde su navegador de internet, una serie de páginas HTML (normalmente de contenido dinámico DHTML), con una cartografía asociada en formato de imagen (por ejemplo, una imagen GIF o JPG sensitiva). Un servidor de mapas es, de hecho, un SIG a través de internet. Cualquier Servidor de Mapas en Internet puede actuar como cliente y como servidor y así compartir cartografía, visualizarla y operar simultáneamente con datos propios y remotos **(Rozo, 2007)**.

1.13.1. Geoserver

Es una aplicación de código abierto construida en Java que permite publicar datos geoespaciales usando el Web Map Server[4]. Se distingue por tener instaladores sencillos y herramientas de configuración basadas en red.

Características principales:

- Soporte robusto para PostGIS, Oracle, ArcSDE, DB2 y Shapefiles.
- Soporte para transacciones atómicas en datos de respaldo con WFS-T.
- Salidas como GML, JPEG, GIF, PNG, SVG, KML y PDF.
- Compatibilidad con ASP para el desarrollo de sitios web.

1.13.2. MapServer

Es un entorno de desarrollo en código abierto. Permite la creación de aplicaciones SIG con el fin de visualizar, consultar y analizar información geográfica a través de la red mediante la tecnología Internet Map Server (IMS)[5].

Sus características principales son:

- Multiplataforma: Corre bajo plataformas Linux/Apache y Windows.
- Formatos vectoriales soportados: ESRI shapefiles, PostGIS, ESRI ArcSDE, GML
- Formatos raster soportados: TIFF/GeoTIFF, GIF, PNG, ERDAS, JPEG y EPPL7
- Soporta fuentes TrueType.

1.14. Servidor Web

Un servidor web es un programa que se encarga de almacenar documentos HTML, así como imágenes, archivos de textos y demás elementos web que contengan datos, para enviar esta información a los usuarios que la soliciten por la red. Este intercambio es posible pues los servidores web implementan el protocolo HTTP.

1.14.1. Microsoft Internet Information Services

[4] Produce mapas de datos referenciados espacialmente, de forma dinámica a partir de información geográfica.

[5] Servidor de cartografía digital que provee cartografía a través de la red tanto en modo vectorial como con imágenes.

Es un servidor web propietario cuyos servicios se limitan a los ordenadores que trabajan con Windows. Estos servicios facilitan las funciones y herramientas necesarias para administrar de forma sencilla un servidor web seguro. Además se basa en varios módulos que le dan la posibilidad de poder procesar distintos tipos de páginas

1.14.2. Apache

Este servidor HTTP es uno de los más utilizados y populares pues es compatible con múltiples sistemas operativos y es una tecnología libre de código abierto. Las capacidades de este servidor pueden ser ampliadas incorporándole nuevos módulos, pues su diseño modular es altamente configurable. Esto le ha permitido incorporar nuevas extensiones entre las que se destaca PHP. Se debe destacar que algunos de los más grandes sitios web del mundo se ejecutan sobre Apache y otros utilizan versiones modificadas del mismo, por su robustez, flexibilidad, estabilidad y eficiencia.

1.15. Conclusiones parciales

En este primer capítulo se han abordado los principales conceptos que permiten comprender mejor que es la arquitectura de software y cuáles son los elementos fundamentales a tener en cuenta para el desarrollo de una correcta arquitectura en un sistema teniendo en cuenta las características particulares del mismo. Se han analizado también algunas herramientas candidatas a ser utilizadas en la aplicación, así como los lenguajes de programación, ambientes de desarrollo, servidores de mapas y servidores web.

Luego de haber realizado dicho análisis se puede concluir que los distintos tipos de estilos y patrones arquitectónicos, aún teniendo características específicas que los diferencian, tienen como objetivo común el facilitar el diseño de aplicaciones robustas y confiables. Además de destacar que las tecnologías que brindan servicios de mapas complementan los sistemas de apoyo al proceso de toma de decisiones.

CAPÍTULO 2: Propuesta de tecnologías a utilizar.

2.1. Introducción

Después de haber realizado en el anterior capítulo un profundo análisis sobre los principales aspectos de la Arquitectura de Software y de todo el proceso de diseño arquitectónico de un sistema, en este capítulo se brinda el resultado obtenido en dicho estudio. Especificando el estilo arquitectónico y los distintos patrones de diseño que guiarán el desarrollo de la solución más adecuada para el diseño de la arquitectura base del catálogo de mapas LiberMaps. Además de definir las distintas tecnologías que darán soporte a la aplicación. En este sentido se tuvieron en cuenta dos aspectos fundamentales a la hora de realizar la selección: que dichas metodologías, herramientas y tecnologías fueran las que mejor se ajustaran al diseño arquitectónico del sistema y que fueran libres.

2.2. Estilo arquitectónico Modelo Vista Controlador

El estilo seleccionado para modelar la base arquitectónica de LiberMaps pertenece a los estilos de Llamada y retorno, pues las arquitecturas que se encuentran dentro de esta familia son las más generalizadas en sistemas a gran escala. Este estilo es el MVC, que fue creado para implementar interfaces de usuario donde las responsabilidades están bien distribuidas en distintas partes o componentes del diseño. Es actualmente el más usado en la confección de aplicaciones debido a las ventajas que ofrece debido a la forma en que organiza los elementos de la aplicación, pues al separar cada una de las capas que lo conforman en 3 clases diferentes brinda facilidades en el desarrollo de la aplicación.

Modelo: Responsable de administrar el comportamiento y los datos del dominio de aplicación. Coordina la lógica de la aplicación, acceso a BD y otras partes no visuales del sistema. Representa la lógica de negocio de la aplicación.
Vista: Responsable de manejar la visualización de la información. Interactúa con el usuario.
Controlador: Responsable de interpretar las acciones del usuario, y traducirlas como solicitudes de servicios para el modelo o la vista. Se encarga de realizar las siguientes tareas:

- Control de la seguridad.
- Identificación de eventos.
- Preparar el modelo.
- Procesar el evento.
- Manejar los errores.
- Provocar la generación de la respuesta.

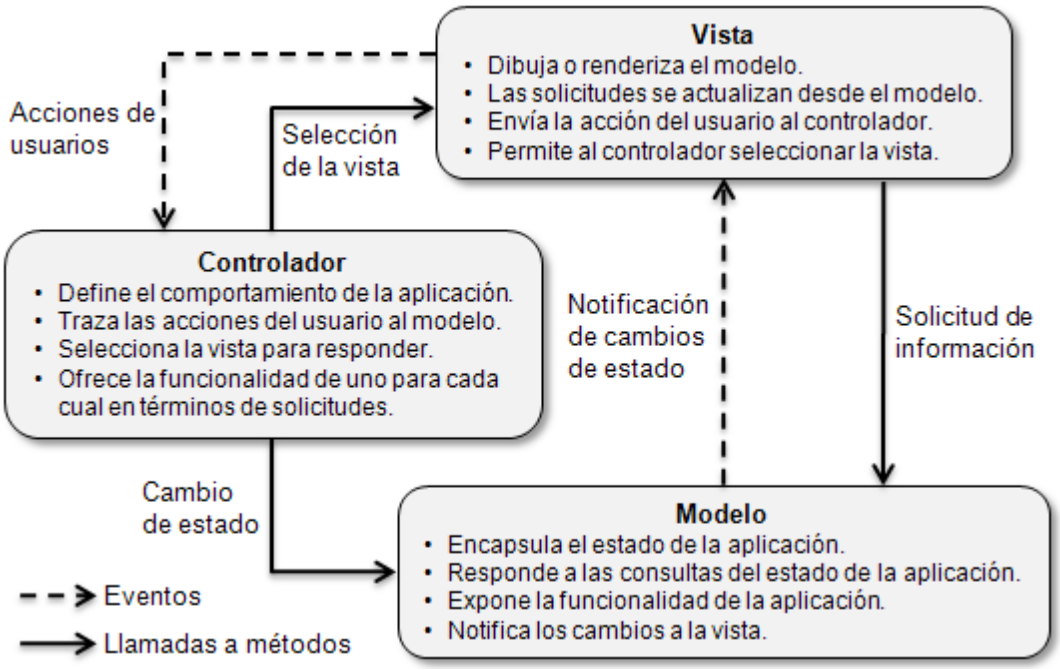

Figura 3 – Representación Del MVC (Buschmann, 1996).

Esta separación en tres niveles garantiza el funcionamiento seguro de la aplicación y permite lograr aplicaciones fácilmente escalables y reutilizables. Además ofrece un entorno de trabajo bien definido y organizado, lo que permite que los problemas sean identificados dentro de su ámbito correcto en la arquitectura, por lo que las soluciones que se generan son específicas y sencillas.

Una de las ventajas fundamentales de este estilo, es que al encapsular el modelo de una aplicación en componentes se facilita la depuración, la reutilización de código y aumenta la calidad del sistema.

2.2.1. MVC2

Específicamente se escogió MVC2 que es una implementación modificada del MVC. La diferencia entre el MVC y el MVC2 es que este último logra una independencia entre la vista y el modelo, aunque aún quedan algunos accesos a formularios que se encuentran en el modelo. Es decir, con el MVC2 la comunicación entre las vistas y el modelo se realiza mediante el controlador. El hecho de separar el modelo de la vista permite la construcción de interfaces con diferentes apariencias.

2.3. Patrones de diseño seleccionados

Para el desarrollo del sistema se propone que además de hacer uso de patrones de diseño muy conocidos pues se utilizan en la mayoría de las aplicaciones como es el caso de los patrones GRASP (creador, controlador, alta cohesión, bajo acoplamiento), se implementen otros más específicos en correspondencia con la aplicación a desarrollar. Estos patrones se han definidos pensando en las características específicas de LiberMaps, así como los requisitos que se espera cumpla esta aplicación.

2.3.1. Observador

Este patrón de comportamiento se emplea cuando se tiene un objeto observado y varios observadores, encargándose de notificar cualquier cambio ocurrido en el objeto observado. En el caso particular de LiberMaps los observadores serían las listas desplegables o combobox. Esto garantiza que si ocurre algún cambio, todos se modifiquen con los datos actuales del sistema. Un ejemplo de ello sería cuando se desea duplicar un mapa en la aplicación. Una vez duplicado, el nuevo mapa debe aparecer inmediatamente en la relación de mapas de todos los combobox de la interfaz en la que se está trabajando para que el usuario pueda interactuar con el mismo. Esta es la función del patrón observador y de ahí la importancia de su uso en el desarrollo del sistema.

2.3.2. Fachada

Fachada es un patrón estructural que tiene como objetivo establecer una interfaz simple y fácil de comprender para los usuarios. En dicha interfaz se añaden las funcionalidades más utilizadas y se habilita un punto de entrada para unificar el subsistema en cuestión. La función de este patrón en la aplicación a desarrollar es mediar entre la interfaz y las llamadas al servidor. Es decir, encapsula los pedidos AJAX al servidor, los envía al controlador frontal y este se encarga de enviarle los pedidos a los controladores

específicos de las clases que poseen la información necesaria para darle respuesta a las solicitudes realizadas.

2.3.3. Fábrica abstracta

Con este patrón creacional se proporciona una interfaz para la creación de familias de objetos relacionados sin especificar sus clases concretas. Se utiliza porque el sistema es independiente de cómo se crean sus objetos. Este patrón se encarga de proporcionar el objeto creado y la aplicación lo utiliza sin necesidad de conocer como se realizó el proceso de creación de dicho objeto. Este objeto es el que le permite al sistema saber la configuración que deberá tener la interfaz a visualizar según los permisos y roles del usuario que se ha autenticado. Los objetos que se crean son los manejadores de los distintos formularios que se mostrarán.

2.4. Metodología de desarrollo de software seleccionada

2.4.1. RUP

Se define RUP como la metodología más adecuada a utilizar por ser robusta, esto es lo que le permite soportar el desarrollo de grandes proyectos, permitiendo además la generación de toda la documentación asociada al mismo en cada una de sus fases.
RUP agrupa las mejores prácticas de metodologías anteriores y las modernas, adaptándose al contexto y necesidad de cada aplicación. El hecho de que RUP sea iterativo da la posibilidad de organizar los proyectos por fases las cuales van a constar de una o más iteraciones. Esto ayuda a mitigar los riesgos tempranamente pues en cada fase se obtendrán versiones que se irán modificando y mejorando a lo largo del ciclo de vida del software. Es por ello que esta metodología garantiza que se logre desde las primeras fases de creación un producto de calidad.

2.5. UML

El lenguaje de modelado UML es el más eficiente si se va a emplear RUP como metodología de desarrollo de software. Esta combinación de RUP con UML es la más utilizada para realizar el análisis, implementación y documentación de los sistemas orientados a objetos. UML se ha ido desarrollando y han ido surgiendo nuevas versiones. En este caso se ha seleccionado UML 2.1, que define 13 tipos

básicos de diagramas, divididos en dos grupos generales: Diagramas de Modelado de Comportamiento (casos de usos, actividades, máquinas de estados, comunicaciones, secuencia, tiempo, interacción) y Diagramas de Modelado Estructurado (paquetes, clases, objetos, estructuras compuestas, componentes, despliegue)

2.6. Herramienta Case seleccionada

2.6.1. Visual Paradigm

Visual Paradigm se considera la herramienta de modelado más adecuada para trabajar en software libre, esta propiedad es básicamente, la que constituyó un hecho determinante en su selección para ser utilizada en el proceso de modelado del catálogo de mapas LiberMaps. También se tuvieron en cuenta sus características principales y las facilidades que brinda a los usuarios pues posee una interfaz amigable y fácil de utilizar.

2.7. Lenguajes de programación seleccionados

2.7.1. PHP5

El lenguaje de programación del lado del servidor escogido es PHP5, pues por sus características se considera que posee la mejor mezcla entre flexibilidad y rendimiento para la realización de páginas web dinámicas. Además cuenta con abundante documentación lo que permite un mayor entendimiento del mismo. Este lenguaje está asociado con una extensa biblioteca que va creciendo conforme se realizan nuevas versiones, la cual permite la realización de disímiles tareas como la encriptación, la creación de PDF, el acceso a BD y el tratamiento de ficheros.

2.7.2. JavaScript

JavaScript es compatible con la mayoría de los navegadores, esto es lo que lo convierte en el lenguaje del lado del cliente más utilizado. Es por ello que se propone para su utilización, además de sus características anteriormente analizadas, así como la posibilidad que tiene de poder incluirse en cualquier documento, ya sea PHP, JS, ASP, entre otros. Se escogió también por ser un lenguaje sencillo que permite a las personas que no tengan una experiencia previa en la programación aprender este lenguaje fácilmente y utilizarlo en toda su potencia con sólo un poco de práctica.

2.8. Frameworks seleccionados

2.8.1. Symfony

De los frameworks de PHP analizados se escogió Symfony, pues además de estar basado, al igual que el resto de los frameworks de PHP, en el estilo arquitectónico seleccionado previamente: MVC; implementa específicamente MVC2. Además de ser consecuentes con el lenguaje escogido del lado del servidor, pues Symfony está desarrollado completamente en PHP5. Si bien es cierto que este framework no es de los mejores en rendimiento, esto se debe precisamente a las ventajas que brinda para reducir el tiempo de desarrollo de un sistema, pues sus propias clases consumen recursos y tiempo. No obstante, posee una serie de cualidades que compensan este detalle y que lo convierten en el más completo de los frameworks de PHP.

En Symfony, el acceso y la modificación de los datos almacenados en la base de datos se realiza mediante objetos. Esto le permite un alto nivel de abstracción y una fácil portabilidad. Un aspecto clave a tener en cuenta para el desarrollo de LiberMaps es la seguridad de la información. Esta es otra de las razones por la que se escoge Symfony, pues el mismo da la posibilidad de restringir el acceso de los usuarios a la información según sus privilegios. Para que esto se cumpla se deben declarar los requerimientos de seguridad para cada acción y establecer los niveles de acceso de los usuarios.

2.8.2. ExtJS

ExtJS fue el framework de AJAX seleccionado para apoyar el lenguaje del lado del cliente JavaScript, pues permite la creación de aplicaciones complejas de forma más eficiente. Este framework permite distribuir la carga de procesamiento por lo que se logra establecer un balance entre el cliente y el servidor. Esto posibilita que el servidor, al tener menos carga, pueda atender más clientes a la vez.

2.9. Sistema gestor de bases de datos seleccionado

2.9.1. PostgresSQL

El gestor de base de datos elegido fue PostgresSQL, pues permite el soporte para datos espaciales en un Sistema de Información Geográfica mediante la extensión PostGIS, conjuntamente con la gestión de objetos geográficos. Aspecto fundamental dadas las características de la aplicación que se desea desarrollar. Se tuvo en cuenta, además de sus características analizadas previamente, que cuenta con un

mejor soporte para triggers, vistas, herencia y procedimientos almacenados, así como el hecho de que escala muy bien cuando aumenta el número de CPUs y la cantidad de RAM, debido a su arquitectura de diseño.

2.10. Herramienta IDE seleccionada

2.10.1. Netbeans

Netbeans se selecciona como herramienta de entorno de desarrollo integrado siendo consecuentes con uno de los lenguajes y frameworks previamente propuestos, pues brinda la posibilidad de desarrollar aplicaciones web utilizando PHP 5 y ofrece soporte para Symfony.

2.11. Servidor de mapas seleccionado

2.11.1. MapServer

Como servidor de mapas se seleccionó a MapServer pues de todas las tecnologías de código abierto que permiten la representación geoespacial, este es hasta el momento el más estable. Además, su funcionalidad aumenta si se combina su instalación con PostgresSQL como gestor de BD y PHP como lenguaje de programación, por lo que se ajusta a las tecnologías seleccionadas anteriormente.

2.12. Servidor web seleccionado

2.12.1. Apache

Como servidor web se escogió Apache por ser una tecnología libre, en contraste con el Microsoft Internet Information Server que es solamente para Windows. Además de ser actualmente el servidor web más utilizado debido a las funcionalidades y eficiencia que brinda y ser el más adecuado para PHP.

2.13. Conclusiones

En este capítulo se definieron, luego de haber realizado un análisis con los elementos obtenidos en la investigación del primer capítulo, los estilos arquitectónicos y patrones de diseño, así como las herramientas y tecnologías que mejor se ajustan en el desarrollo del catálogo de mapas LiberMaps. Es vital que todo arquitecto domine o al menos tenga conocimientos de diferentes tecnologías de software y

prácticas de diseño pues de ello depende que las decisiones que tome sean las que mejor se adapten a las necesidades del proyecto en cuestión y las soluciones que se logren sean las óptimas.

Se debe destacar que a la hora de tomar decisiones sobre los elementos a emplear para desarrollar el catálogo de mapas LiberMaps, se tuvo en cuenta la experiencia previa que tiene el equipo de trabajo con el uso de esas tecnologías, además de ser consecuentes con los lineamientos y la arquitectura existente en el grupo al cual pertenece el equipo de desarrollo de la aplicación.

CAPÍTULO 3: Línea base de la arquitectura.

3.1. Introducción

Para poder cumplir con el objetivo trazado en esta investigación, es necesario proponer una arquitectura que le permita al catálogo de mapas LiberMaps contar con las funcionalidades básicas requeridas y que le proporcione, mediante la organización de sus componentes, una mayor rapidez en el proceso de desarrollo. En este capítulo se brindará la descripción de dicha arquitectura, mediante la representación de las 4+1 vistas arquitectónicas que se definen en la metodología de software que se emplea: RUP. Se detallarán además los elementos significativos de la arquitectura propuesta para el sistema a desarrollar, así como la estructuración de los componentes que forman parte de la misma.

3.2. Organización de la estructura de LiberMaps 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.

3.2.1. Modelo

Se divide en las capas de acceso a datos y la capa de abstracción de BD. 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 BD, 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

[6] Es un ORM (object-relational mapping ó mapeo objeto-relacional) 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.

datos. Además genera por cada tabla de la BD, 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.
- **BaseNombreTablaPeer:** Contiene un conjunto de métodos estáticos que complementan el acceso y la lógica de los datos.

3.2.2. 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.

3.2.3. 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.

3.3. Organización de la estructura de LiberMaps con EXTJS.

Las interfaces de la aplicación se realizarán con EXTJS. La utilización de esta librería de JavaScript, descrita en los capítulos anteriores se expresará a través de un paquete que estará representando todos los componentes de la propia librería, tal como se representa en la Figura 4:

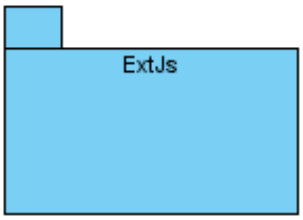

Figura 4 – Paquete Librería EXTJS

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, Figura 5.

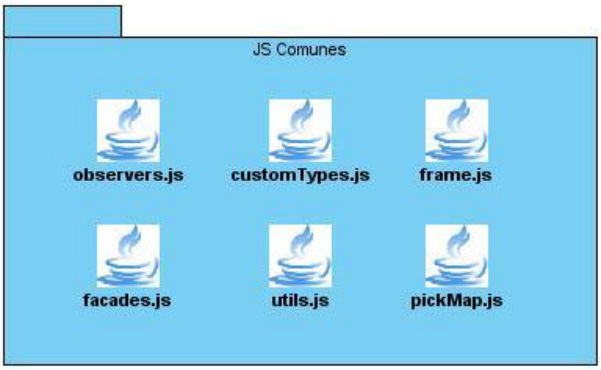

Figura 5 – Paquete de JS comunes para los casos de uso.

En resumen, la estructura general del catálogo de mapas LiberMaps, quedaría de la forma que se muestra en el siguiente diagrama de paquetes:

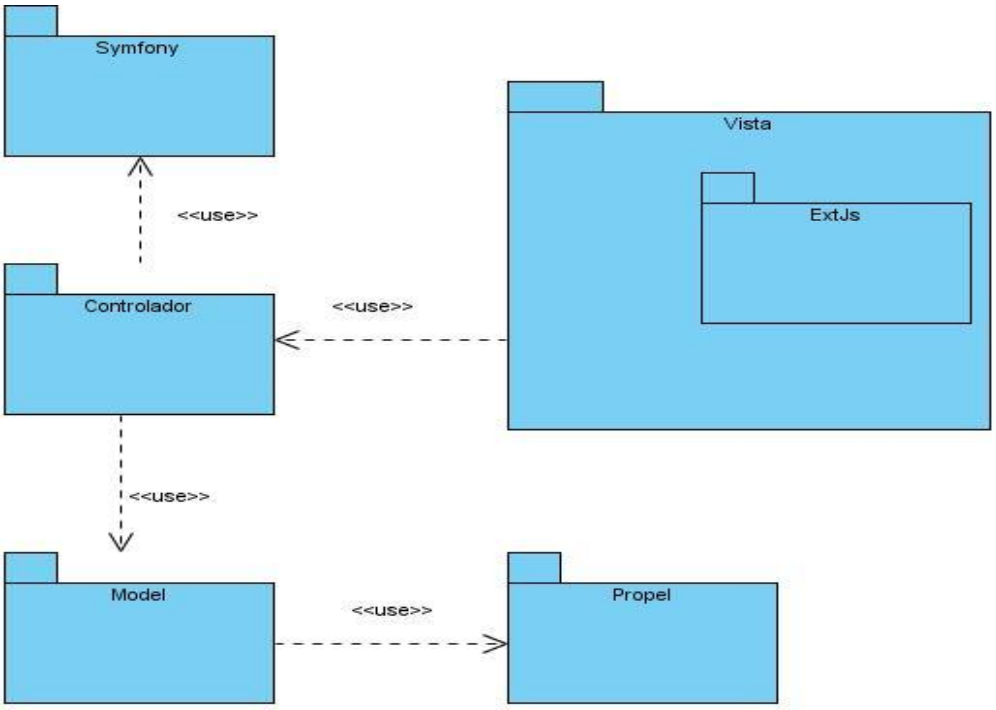

Figura 6 – Estructura de LiberMaps.

3.4. Metas y restricciones de la arquitectura

LiberMaps está conformado por dos módulos: Gestión de Mapas y Gestión de Perfiles.

- Gestión de mapas: Es en este módulo donde se crean y modifican los mapas o mapfiles[7] con todos sus objetos. Se brindan además otras funcionalidades como la exportación de un mapfile, duplicado, creación de estilos y proyecciones.
- Gestión de Perfiles: Es en este módulo donde se crean los perfiles de usuarios tanto sobre los datos como a las funcionalidades del catálogo, permitiendo otorgar a cada usuario permisos sobre un determinado mapa y a tantas capas como se desee, a su vez posibilita de acuerdo a los roles definidos, asignar una configuración previa de las funcionalidades, de modo que cuando un usuario se autentique sólo pueda acceder a los datos definidos en su perfil, y a las funcionalidades específicas de su rol.

3.4.1. Requisitos funcionales

3.4.1.1. Requisitos funcionales (RF) del módulo Gestión de Mapas:

RF1: Gestionar Mapfile.

RF2: Exportar Mapfile.

RF3: Configurar Datos Globales.

RF4: Combinar Mapfile.

RF5: Gestionar Agrupaciones.

RF6: Modificar MapObject.

RF7: Modificar Webobject.

RF8: Gestionar Metadatos Webobject.

RF9: Gestionar Outputformats.

[7] Se usa para optimizar la velocidad del Mapserver, cargándose las capas por defecto al iniciar la aplicación y se van modificando según se desee. Fichero sobre el que Mapserver extrae la configuración del mapa.

RF10: Modificar Legend.

RF11: Modificar Scalebar.

RF12: Gestionar Symbols.

RF13: Modificar ReferenceMap.

RF14: Modificar QueryMap.

RF15: Gestionar Layers.

RF16: Modificar Layers.

RF17: Gestionar Metadatos de las Layers.

RF18: Gestionar Clases de una Layer.

RF19: Gestionar Estilos de las Clases de las Layers.

RF20: Mostrar vista previa de las Layers.

RF21: Importar mapfiles.

RF22: Ofrecer servicios web.

3.4.1.2. Requisitos funcionales (RF) del módulo Gestión de Perfiles:

RF23: Crear perfiles de usuario en cuanto a datos.

RF24: Configurar Perfil Datos.

RF25: Crear perfiles de usuario en cuanto a funcionalidades.

RF26: Gestionar usuarios del sistema.

RF27: Autenticar Usuarios.

3.4.2. Requisitos no funcionales (RNF)

3.4.2.1. Usabilidad

- El sistema podrá ser usado por personas con conocimientos básicos en el manejo de computadoras.
- Se emplearán componentes que indiquen al usuario el estado de los procesos que por su complejidad requieran de un tiempo de procesamiento apreciable.
- El software tendrá siempre visible la opción de Ayuda, lo que posibilitará un mejor aprovechamiento por parte de los usuarios de sus funcionalidades.

3.4.2.2. Fiabilidad

- El sistema debe estar disponible todo el tiempo para sus usuarios, descontando el tiempo que se encuentre en mantenimiento.
- El tiempo entre fallos no debe exceder los 3 meses.
- El tiempo medio de reparación en caso de fallos es de 7 días.

3.4.2.3. Eficiencia

- El tiempo de respuesta estará dado por la cantidad de información a procesar, entre mayor cantidad de información, mayor será el tiempo de procesamiento.
- Al igual que el tiempo de respuesta, la velocidad de procesamiento de la información, la actualización y la recuperación dependerán de la cantidad de información que tenga que procesar la aplicación.

3.4.2.4. Soporte

El periodo de soporte así como las restricciones asociadas se manejarán entre el equipo de desarrollo y los clientes.

3.4.2.5. Restricciones de diseño

- Diseño sencillo, con pocas entradas, donde no sea necesario mucho entrenamiento para utilizar el

sistema.

- Se deben emplear los estándares establecidos (diseño de interfaces, base de datos y codificación).

3.4.2.6. Requisitos para la documentación de usuarios en línea y ayuda del sistema.

El software tendrá siempre la posibilidad de ayuda disponible para cualquier tipo de usuario, lo que le permitirá un avance considerable en la explotación de la aplicación en todas sus funcionalidades.

3.4.2.7. Interfaz

Interfaces de usuario:

- El sistema debe tener una apariencia profesional y un diseño gráfico sencillo.
- El sistema debe ser intuitivo.

Interfaces Hardware:

Para las PCs clientes:

- Se requiere tengan tarjeta de red.
- Al menos 64 MB de memoria RAM.
- Se requiere al menos 100 MB de disco duro.
- Procesador 512 MHz mínimo.

Para los servidores:

- Se requiere tarjeta de red.
- El Servidor de Mapas tenga como mínimo 2GB de RAM y 2GB de disco duro.
- El Servidor de BD tenga como mínimo 2GB de RAM y 10GB de disco duro.
- Procesador 3 GHz como mínimo.

Interfaces Software:

Para las PCs clientes:

- Un Navegador como Mozilla Firefox, Zafari u otro navegador que cumpla con los estándares W3C.
- Sistema operativo: GNU/Linux, Windows y Mac OS.

Para los Servidores:

- Sistemas operativos GNU/Linux o Windows Server 2000 o superior.
- Servidor Web Apache 2.0 o superior, con módulo PHP 5 configurado con la extensión pgsql incluida.
- PostgresSQL como Sistema Gestor de Base de Datos.
- PostGis como extensión de PostgresSQL como soporte de datos espaciales.
- PgRouting como extensión de PostgresSQL para análisis de rutas.
- Mapserver 5.2.2 o superior, con extensión PHP mapscript.

3.4.2.8. Requisitos de Licencia

De acuerdo a los tipos de licencias de los componentes y herramientas que se proponen a utilizar para el desarrollo de LiberMaps se puede catalogar legalmente esta arquitectura de modelo libre, permitiendo la utilización, modificación y distribución de las mismas por terceros sin necesidad de obtener la autorización de sus respectivos titulares.

3.4.2.9. Requisitos Legales, de Derecho de Autor y otros.

- El sistema debe ajustarse y regirse por la ley, decretos leyes, decretos, resoluciones y manuales (órdenes) establecidos, que norman los procesos que serán automatizados.
- La mayoría de las herramientas de desarrollo son libres y del resto, las licencias están avaladas.
- Como producto, LiberMaps se distribuye amparado bajo las normativas legales establecidas en el registro comercial emitido por las entidades jurídicas de la Universidad de las Ciencias Informáticas.

3.4.2.10. Estándares Aplicables

El sistema será desarrollado bajo estándares Open GIS como aseguramiento de la parte científica y en el desarrollo se codificará y modelará siguiendo los patrones de las normativas ISO, tanto de codificación como de diseño de bases de datos.

3.5. Descripción de la arquitectura

A continuación se describe la arquitectura de LiberMaps apoyados en la metodología RUP y en sus 4+1 vistas arquitectónicas, Figura 7, que se encargan de reunir los elementos más significativos del desarrollo de un sistema.

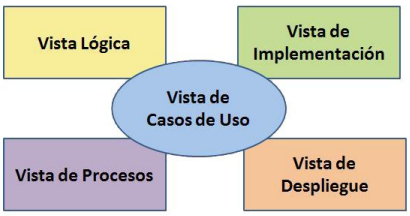

Figura 7-- Vistas arquitectónicas de RUP

3.5.1. Vista de casos de uso

La vista de casos de uso representa un subconjunto del artefacto Modelo de Casos de Uso y se encarga de listar los casos de usos más significativos, con las funcionalidades principales del sistema. Contiene además los casos de usos significativos para la arquitectura, que son aquellos que describen funcionalidades imprescindibles para el sistema, y a través de los cuales se valida la arquitectura propuesta para el mismo, así como los diagramas de casos de uso.
En el sistema se definieron 27 casos de uso, de ellos 20 críticos (C.U.C) y 7 secundarios (C.U.S).
Quedando estructurados los mismos, de la siguiente forma:

Tabla 2 -- Casos de uso del sistema por módulos.

Módulos	C.U.C	C.U.S
Gestión de Mapas	13	7
Gestión de Perfiles	7	-

A continuación se presentan los casos de usos críticos del catálogo de mapas LiberMaps.

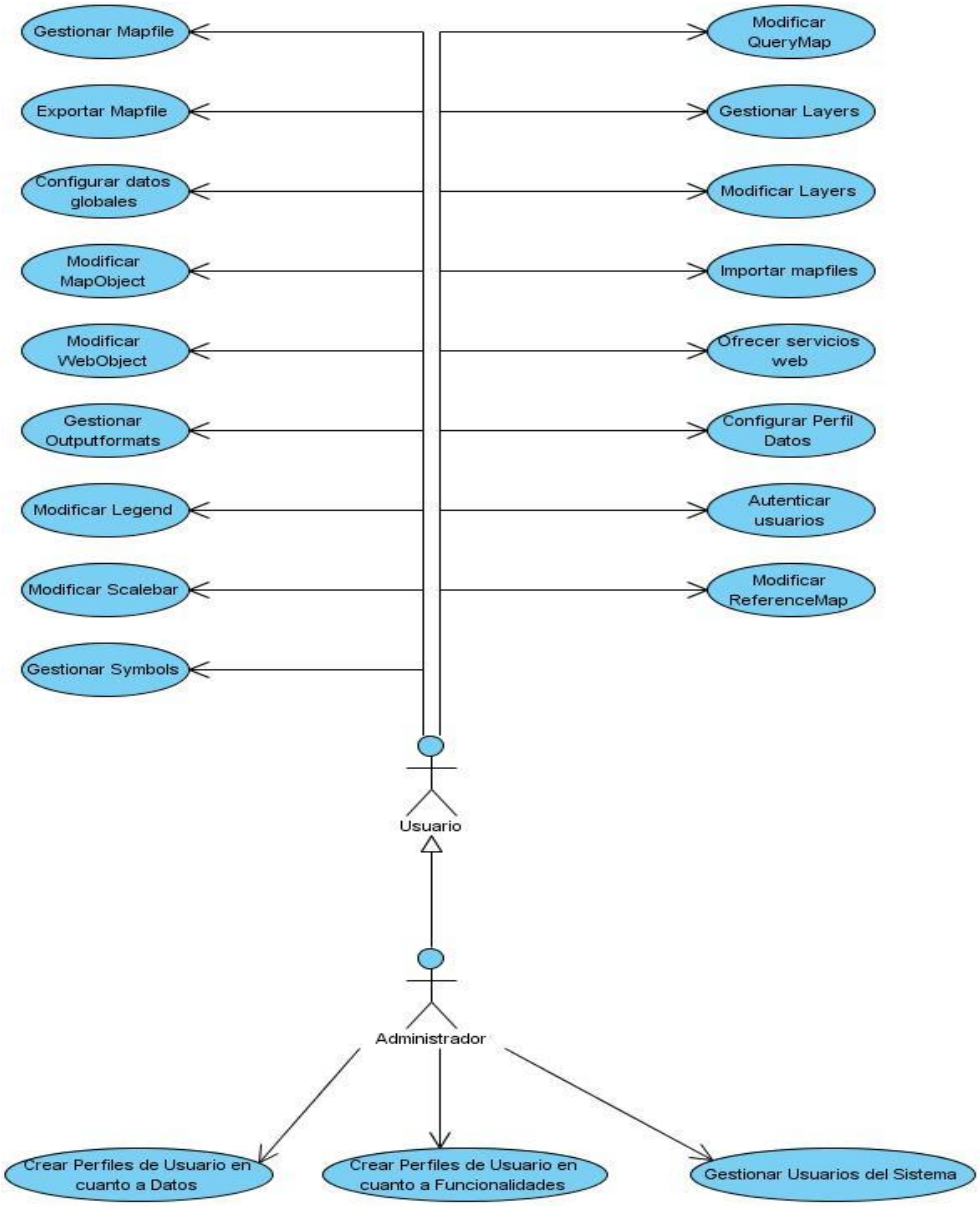

Figura 8 -- Vista de C.U.C LiberMaps

3.5.2. Vista lógica

La vista lógica contiene las clases más importantes por las que se compone el sistema con su explicación detallada, la descomposición de las mismas en subsistemas y paquetes y las relaciones que se establecen entre ellas. Contiene los elementos del diseño más significativos para la arquitectura.

Para un mayor entendimiento, se representa la vista lógica para las aplicaciones Web, empleando el framework Symfony basado en la estructura del MVC.

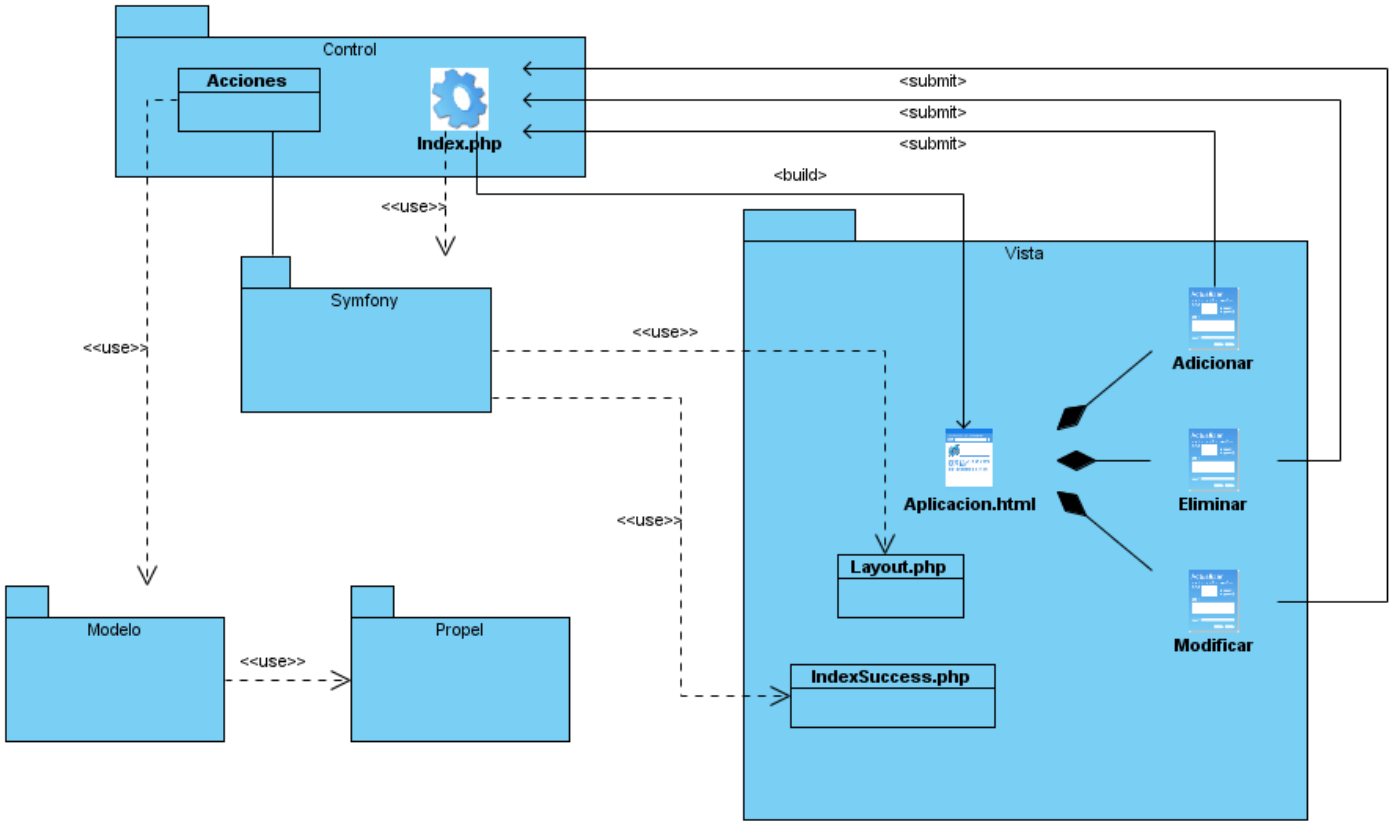

Figura 9 -- Vista lógica aplicaciones Web

La representación del modelo para cada una de las clases de la base de datos quedaría como se muestra en la figura 10.

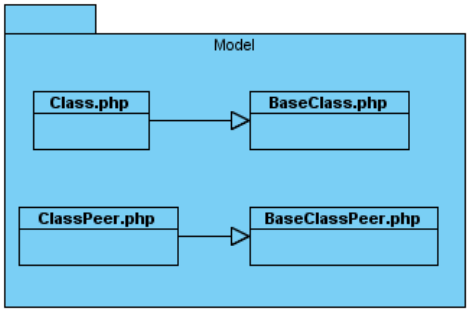

Figura 10 -- Estructura de las clases del Modelo generadas por Symfony

En el diseño de las clases de cada caso de uso se obviará esta estructura interna representándose solamente un paquete con el nombre de la clase, Figura 11.

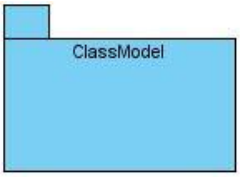

Figura 11 -- Paquete de clases del modelo para un objeto de la BD

3.5.3. Vista de procesos

La vista de procesos proporciona una base para comprender la organización de los procesos de un sistema. La misma solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

El catálogo de mapas LiberMaps, al igual que toda aplicación Web, funcionará bajo los conceptos de arquitectura cliente/servidor sobre la plataforma web, donde cada instancia del sistema en el cliente es independiente de la ejecución de otra instancia en el lado del servidor. Además la concurrencia de utilización del servidor Web Apache y de la Base de Datos se maneja mediante los propios servidores. Es por ello que el sistema no cuenta con tareas que necesiten ejecutarse periódicamente sin la intervención del cliente y por tanto no se requiere de una Vista de Procesos.

3.5.4. Vista de despliegue

La vista de despliegue de un sistema contiene los nodos que forman la topología hardware sobre la que se ejecuta el sistema, la distribución, entrega e instalación de las partes que constituyen el sistema físico. **(Flores, 2003).**

Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Es decir, describe los nodos físicos necesarios para la configuración de la plataforma donde se ejecutará el sistema.

3.5.4.1. Diagrama de despliegue

El diagrama de despliegue se encarga de mostrar las relaciones físicas entre los componentes hardware y software en el sistema, Figura 12. Es la representación de un conjunto de nodos relacionados por conexiones de comunicación.

Para realizar el despliegue de la aplicación se requiere de un servidor Web, donde se ejecutarán todas las tareas propias del mismo, tales como la construcción de interfaces de usuarios, procesamiento de datos, y control de flujo; junto con el servidor Web se encontrará el servidor de mapas. Se deberá contar también con un servidor de base de datos, donde se estará ejecutando el sistema gestor de bases de datos, PostgresSQL.

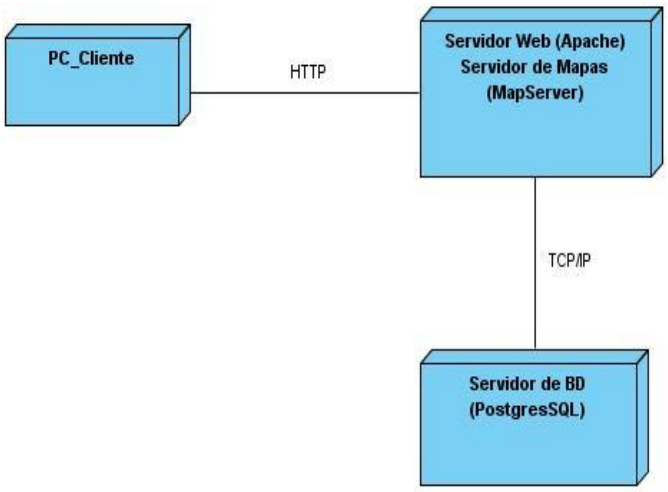

Figura 12 -- Diagrama de despliegue LiberMaps

3.5.5. Vista de implementación

La vista de implementación describe la descomposición del software en componentes y subsistemas de implementación. Un componente de software constituye una parte física de un sistema, ya sea un módulo, una base de datos, etc.
Algunos de los elementos de esta vista son:

- Subsistemas de implementación.
- Diagramas de componentes

A continuación se muestra la vista de implementación para LiberMaps.

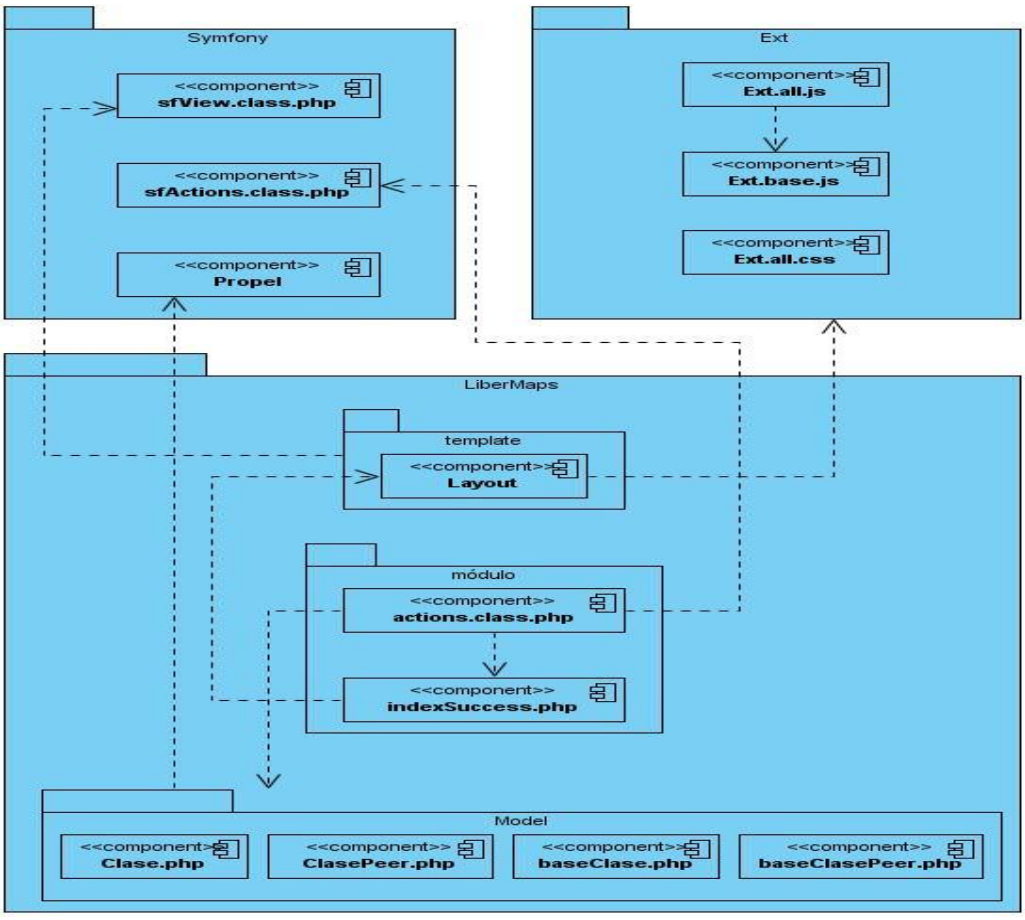

Figura 13 -- Vista de implementación

En la figura anterior se muestra la carpeta LiberMaps que contiene a su vez las carpetas: Template, Módulo y Model. Se tienen además las carpetas Symfony y Ext.

- Template: Aquí se encuentra el componente layout.php que es la plantilla que se le aplica a todas las páginas. Esta hace uso de los ficheros definidos dentro de la carpeta Ext para conformar la vista y del componente sfView.class.php de la carpeta Symfony para mostrar dicha vista.
- Módulo: Esta carpeta está integrada por un componente actions.class.php (representando al controlador) y un componente indexSuccess.php (representando a la vista). El primero se relaciona con la carpeta Model para el acceso a los datos de la base de datos, además hace uso del componente sfAction.class.php que trae Symfony por defecto. Por otro lado, la vista utiliza el componente layout.php que está en la carpeta Template para darle forma a la página.
- Model: Contiene las clases responsables de manejar la información con las tablas de la base de datos. Symfony crea cuatro clases en la BD por cada tabla. Model utiliza el componente Propel para la abstracción a la BD.
- Symfony: Las librerías a usar para el desarrollo de la aplicación teniendo en cuenta el patrón MVC se encuentran en esta carpeta: sfView.class.php para mostrar las vistas, sfAction.class.php para la controladora y Propel para el modelo.
- Ext: Librería Java Script ligera y de alto rendimiento, compatible con la mayoría de los navegadores, usada para trabajar la parte de la presentación de las vistas.

3.6. Conclusiones

Al terminar este capítulo queda establecida finalmente la propuesta de Arquitectura de Software para el catálogo de mapas LiberMaps. En el mismo se abordaron tanto los requerimientos funcionales como los no funcionales con que cuenta el sistema, lo que permite una mejor comprensión de las metas y restricciones que debe cumplir, así como la definición de las 4+1 vistas arquitectónicas de RUP. En resumen, en el capítulo se definió detalladamente la arquitectura candidata, la cual ha sido enfocada a obtener flexibilidad y facilidad para el usuario final a la hora de utilizar el sistema.

CAPÍTULO 4: Evaluación de la arquitectura.

4.1. Introducción

En este capítulo se describirán brevemente algunos de los métodos que se emplean en la evaluación de la arquitectura de software así como otros aspectos relacionados con este tema, para finalmente abordar todo lo referente a la evaluación de la arquitectura propuesta como solución al problema planteado en la investigación realizada. La evaluación de la arquitectura es fundamental para conocer los problemas o debilidades que pueden estar presentes en el sistema y poder corregirlos a tiempo. Esta evaluación no define si una arquitectura determinada es buena o no, expresa simplemente donde se encuentran los riesgos y fortalezas de la misma.

4.2. Necesidad de evaluar la arquitectura

Aunque la arquitectura solamente puede permitir, no garantizar, que un atributo de calidad sea alcanzado, la medida en que un sistema logra alcanzar determinados atributos de calidad depende, sin dudas, de las decisiones que se hayan tomado con respecto a la arquitectura del mismo. Es por ello que la propuesta arquitectónica puede ser evaluada en función del impacto que tiene sobre dichos atributos.

Realizar evaluaciones a la arquitectura es vital para detectar riesgos potenciales en su estructura y propiedades que puedan afectar al sistema a desarrollar y buscar, por tanto, soluciones para corregirlos a tiempo. Además, permite verificar que los Requisitos no Funcionales (RNF) establecidos estén presentes en la arquitectura y determinar el grado en el que se satisfacen los atributos de calidad.

A continuación se especifican algunas de las ventajas que produce la evaluación de la arquitectura:

- Detección temprana de riesgos en la arquitectura.
- Mejora la arquitectura.
- Permite la documentación de las decisiones arquitectónicas tomadas y su fundamento.
- Permite constatar el grado de cumplimiento de una arquitectura con los RNF de un sistema.

4.3. ¿Cuándo realizar la evaluación de la arquitectura?

Aunque generalmente se evalúa la arquitectura de software cuando ya está diseñada pero sin haber comenzado a implementar aún, dicha evaluación puede ser realizada en cualquier momento. No obstante, Kazman propone dos categorías de las diferentes etapas en las que puede realizarse: evaluación temprana y evaluación tardía.

- Evaluación temprana: Para esta primera variante no es necesario que la arquitectura esté completamente especificada. Esto permite que se puedan tomar decisiones referentes a la arquitectura en cualquier nivel, debido a que se pueden efectuar cambios arquitectónicos como resultado de una evaluación en función de los atributos de calidad que se esperan del sistema.
- Evaluación tarde: En la segunda categoría se realiza la evaluación de la arquitectura cuando la misma se encuentra establecida y se ha terminado además la implementación. Esto suele presentarse generalmente cuando se ha adquirido un sistema ya desarrollado y se desea comprobar que éste cumple con los atributos de calidad a los que está asociado.

4.4. Cualidades por las que se puede evaluar la arquitectura

Se considera que la calidad del software está determinada por el grado en el que este posee una combinación deseada de atributos. Estos atributos son características que el sistema debe satisfacer. Estas características o atributos, según Barbacci, se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios **(Barbacci, 1995).**

Bass establece una clasificación para los atributos de calidad que se divide en dos categorías:

- Observables vía ejecución: aquellos atributos que se determinan por el comportamiento del sistema en tiempo de ejecución.
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema **(Bass, 1998)**.

En las siguientes tablas se describen algunos de los principales atributos de calidad, agrupados según la clasificación ofrecida por Bass.

Tabla 3 -- Descripción de atributos de calidad observables vía ejecución.

Atributo de Calidad	Descripción
Disponibilidad *(Availability)*	Es la medida de disponibilidad del sistema para el uso **(Barbacci, 1995).**
Funcionalidad *(Functionality)*	Habilidad del sistema para realizar el trabajo para el cual fue concebido **(Kazman, 2001)**.
Desempeño *(Performance)*	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria **(IEEE 610.12).**
Confiabilidad *(Reliability)*	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo **(Barbacci, 1995).**
Seguridad interna *(Security)*	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos **(Kazman, 2001)**.

Tabla 4 -- Descripción de atributos de calidad no observables vía ejecución.

Atributo de Calidad	Descripción
Configurabilidad *(Configurability)*	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema **(Booch, 1999)**.

Modificabilidad *(Modifiability)*	Es la habilidad de realizar cambios futuros al sistema **(Booch, 1999)**.
Portabilidad *(Portability)*	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos **(Kazman, 2001)**.
Reusabilidad *(Reusability)*	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones **(Bass, 1998)**.
Escalabilidad *(Scalability)*	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental **(Pressman, 2002).**

4.5. Técnicas de evaluación

Las técnicas utilizadas para la evaluación de atributos de calidad requieren grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema **(Bosch, 2000)**.

Las técnicas de evaluación se clasifican en cualitativas y cuantitativas, Figura 14. En la primera clasificación se encuentran las técnicas que son usadas cuando la arquitectura se encuentra aún en construcción y de ellas se obtienen respuestas de sí o no. Entre las mismas se encuentran: escenarios, cuestionarios y listas de chequeo. La segunda clasificación agrupa las técnicas que se utilizan cuando la arquitectura ya ha sido implantada. Estas son: métricas, normas, máximos y mínimos teóricos, simulaciones, prototipos, etc.

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos **(Bosch, 2000).** Las técnicas que cumplen con lo antes planteado son las cualitativas, que son las más empleadas por los arquitectos debido al bajo costo que implica llevar a cabo estas técnicas en comparación con las cuantitativas. Además de que estas últimas tienen la desventaja de depender de los valores máximos y mínimos que se están empleando para realizar la evaluación y estos valores resultan difíciles de predecir.

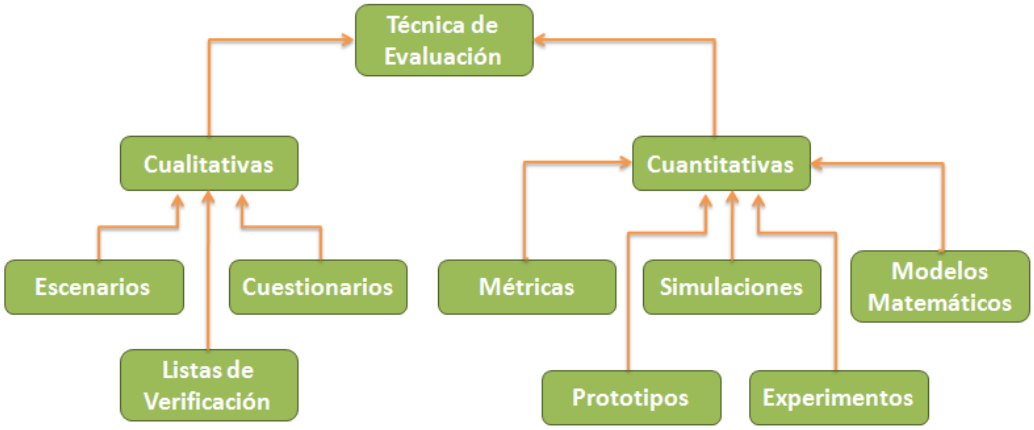

Figura 14 -- Técnicas de evaluación (Salvador, 2007)

4.6. Métodos de evaluación

Actualmente existen múltiples métodos para evaluar la arquitectura de un software, todos ellos con sus características específicas. Estos métodos consisten en una serie de actividades a desarrollar por distintos implicados para analizar la medida en la que cumple la arquitectura diseñada con los atributos de calidad. Para escoger un procedimiento determinado, se deben tener en cuenta sus fortalezas y debilidades así como las particularidades del sistema a desarrollar. Es por ello que a continuación se explican algunos de los métodos más utilizados.

4.6.1. Método de Análisis de Arquitectura de Software (SAAM)

El método de Análisis de Arquitectura de Software (Software Architecture Analysis Method, SAAM), fue el primero en ser ampliamente documentado. Aunque originalmente se diseñó para el análisis de la modificabilidad de la arquitectura, ha demostrador ser muy útil para evaluar otros atributos de calidad de forma rápida, como es el caso de la portabilidad, integrabilidad y escalabilidad. SAAM se enfoca en la enumeración de una serie de escenarios que representan los probables cambios a los que se verá sometido el sistema.

Se establece como entrada principal de este método alguna forma de descripción de la arquitectura a evaluar y se obtienen como resultado, según lo planteado por Kazman, las siguientes salidas:

- Una proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema.

- Entendimiento de la funcionalidad del sistema, e incluso una comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación **(Kazman, 2001).**

Cuando se evalúa una arquitectura con este método se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. En el caso que se estén evaluando varias arquitecturas candidatas, entonces SAAM permite observar cuál de ellas satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

4.6.2. Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM (**Kazman, 2001).**

El nombre de este método se debe al hecho de que muestra la forma en que una arquitectura determinada, satisface atributos específicos de calidad y brinda una visión de cómo esos atributos interactúan con otros, es decir, los tipos de acuerdos que se establecen entre ellos. ATAM se basa en la identificación de los estilos o enfoques arquitectónicos empleados.

Kazman propone el término enfoque arquitectónico dado que no todos los arquitectos están familiarizados con el lenguaje de estilos arquitectónicos, aún haciendo uso indirecto de estos. De cualquier forma, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros **(Kazman, 2001).**

Este método consta de 9 pasos agrupados en 4 fases.

4.6.3. Método de Revisión Intermedio de Diseño (ARID)

El método de Revisión Intermedio de Diseño (Active Reviews for Intermediate Designs, ARID), es conveniente para realizar la evaluación de diseños parciales en etapas tempranas del desarrollo. Es una mezcla entre los métodos ADR y ATAM, analizado anteriormente.

ADR (Active Design Review) es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto (**Kazman, 2001).**

Estos dos métodos (ADR y ATM) aportan características útiles para la evaluación de diseños preliminares. ADR brinda a los involucrados en el desarrollo la documentación detallada que necesitan, así como cuestionarios que deben llenar. ATAM por su parte está orientado a la evaluación de toda la arquitectura. Es por ello que, ante la necesidad de evaluación en las fases tempranas del diseño, los autores del método ARID (Kazman, Clements y Klein) proponen utilizar las particularidades que proveen los métodos mencionados anteriormente y de esta combinación surge ARID.

4.7. Evaluación de la arquitectura definida para LiberMaps

Para el proceso de evaluación de la Arquitectura de Software del catálogo de mapas LiberMaps, se propone el uso de la Metodología de Evaluación de Arquitectura de Software en los proyectos productivos de la Universidad de las Ciencias Informáticas, elaborado por Yamila Vigil Regalado **(Regalado, 2009)**. Se escoge esta metodología porque fue pensada precisamente para aplicarse en la universidad debido a los problemas que existen actualmente en el centro con la evaluación de la arquitectura y su documentación. Este método cuenta con dos áreas de procesos fundamentales: el proceso de evaluación (base) y el proceso de documentación. Consta además de cuatro fases: concepción, presentación, desarrollo y exposición de los resultados. En estas fases se desarrollan un conjunto de actividades, las cuales van a generar artefactos (planillas) que van a contribuir al proceso de documentación.

A continuación se enuncian las fases y actividades que conforman esta metodología:

- **Fase de Concepción.**

1. Selección de los momentos en que se realizará la evaluación arquitectónica a lo largo del ciclo de desarrollo del software.

2. Realización del cronograma de evaluación.

3. Selección de los involucrados.

4. Selección del equipo de evaluación.

5. Preparación de la información.

- **Fase de Presentación.**

6. Descripción del desarrollo del proceso evaluativo.

7. Presentación de los objetivos del negocio.

8. Presentación de la arquitectura de software.

- **Fase de Desarrollo.**

9. Selección de los atributos de calidad.

10. Descripción del comportamiento esperado de cada atributo.

11. Determinación del grado de prioridad de cada atributo.

12. Identificación de los escenarios.

13. Descripción de los escenarios.

14. Establecimiento de la prioridad de los escenarios.

15. Determinar puntos críticos.

- **Fase de Exposición de Resultados.**

16. Valorar impacto de los posibles cambios arquitectónicos en los atributos de calidad.

17. Realización de cambios arquitectónicos.

18. Exposición de los resultados del proceso de evaluación.

Para la evaluación de la arquitectura base de LiberMaps, solamente se tuvieron en cuenta aquellas actividades relacionadas directamente con el arquitecto y se omitieron aquellas que no se consideraban necesarias por ser el mismo arquitecto el responsable de la evaluación. Las actividades realizadas fueron:

1. Selección de los momentos en que se realizarán las evaluaciones.

- Objetivo: Identificar las fases del ciclo de desarrollo adecuadas para evaluar la arquitectura, en correspondencia con el objetivo específico que se persiga con cada evaluación.
- Responsable: Grupo de arquitectura
- Participantes: Involucrados internos, es decir, líder del proyecto, grupo de arquitectura y analistas del sistema.
- Artefacto de entrada: No se requiere de ninguno para realizar esta actividad.
- Artefacto de salida: Definición de Evaluaciones de Arquitectura de Software.

Esta planilla contiene el número de evaluaciones que se realizarán, las fases del ciclo de desarrollo en que se efectuarán las mismas, una breve descripción del motivo de la evaluación y la justificación del momento escogido para realizarla.

2. Selección de los Atributos de Calidad.

- Objetivo: Seleccionar los atributos de calidad cuyo comportamiento dependa directamente del diseño arquitectónico.
- Responsable: Líder del proyecto.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefactos de entrada: Valoraciones de los Objetivos del Negocio, Valoraciones de la Arquitectura Presentada.
- Artefacto de salida: Definición de los Atributos de Calidad.

Este artefacto abarca los atributos necesarios para el logro de la calidad requerida por los clientes, y que impactan directamente en el diseño arquitectónico.

3. Descripción del Comportamiento Esperado de cada Atributo.

- Objetivo: Describir el comportamiento que deben lograr cada uno de los atributos de calidad.
- Responsable: Equipo de arquitectura.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefacto de entrada: Definición de los Atributos de Calidad.
- Artefacto de salida: Comportamiento Esperado de los Atributos de Calidad.

Este artefacto recoge el estado ideal de cada atributo, los elementos que influyen en su comportamiento y el grado de conflicto de cada atributo con el resto.

4. Determinación del Grado de Prioridad de cada Atributo.

- Objetivo: Definir la prioridad de los atributos.
- Responsable: Equipo de arquitectura.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefactos de entrada: Definición de los Atributos de Calidad, Comportamiento Esperado de los Atributos de Calidad.
- Artefacto de salida: Grado de Prioridad de los Atributos de Calidad.

Este artefacto recoge la organización de los atributos según su relevancia.

5. Identificación de los Escenarios.

- Objetivo: Identificar escenarios que permitan evaluar el comportamiento de los atributos de calidad definidos.

- Responsable: Equipo de arquitectura.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefactos de entrada: Definición de los Atributos de Calidad, Comportamiento Esperado de los Atributos de Calidad, Grado de Prioridad de los Atributos de Calidad.
- Artefacto de salida: Identificación de los Escenarios.

Este artefacto abarca una breve descripción de los escenarios definidos así como los atributos asociados al mismo.

6. Descripción de los Escenarios.

- Objetivo: Describir detalladamente los elementos que componen los escenarios definidos.
- Responsable: Equipo de arquitectura.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefacto de entrada: Identificación de los Escenarios.
- Artefacto de salida: Descripción Detallada de los Escenarios.

Este artefacto contiene las características de los escenarios teniendo en cuenta elementos como: origen del estímulo, estímulo, ambiente, componente, respuesta y medida de la respuesta.

7. Establecimiento de la Prioridad de los Escenarios.

- Objetivo: Establecer prioridad de los escenarios definidos.
- Responsable: Equipo de arquitectura.
- Participantes: Clientes, usuarios finales, administradores del sistema, arquitectos, analistas, asesores de calidad y equipo de evaluación.
- Artefactos de entrada: Identificación de los Escenarios, Descripción Detallada de los Escenarios.

- Artefacto de salida: Grado de Prioridad de los Escenarios.

Este artefacto contiene la organización de los escenarios según su relevancia.

4.7.1. Evaluación realizada a LiberMaps

Para realizar la evaluación arquitectónica de LiberMaps, no se consideró necesario realizar algunas de las planillas que genera esta metodología, pues solo se tuvieron en cuenta, aquellas que estuvieran relacionadas directamente con el arquitecto. Además se decide omitir la Fase de Presentación debido a que la misma tiene como objetivo informar a todas las personas que estarán involucradas en el proceso de evaluación, los detalles más significativos de la arquitectura definida y en el caso particular de esta aplicación, la evaluación será realizada por el propio arquitecto. Los artefactos generados en este proceso aparecen reflejados en los anexos del presente trabajo.

4.8. Conclusiones

Después de haber realizado la evaluación de la arquitectura propuesta según la metodología escogida, se puede concluir que la misma cumple con los atributos de calidad definidos: Funcionalidad, Eficiencia, Seguridad, Portabilidad y Usabilidad. La arquitectura se diseñó con el propósito de permitirle al sistema cumplir con todas las funcionalidades establecidas, así como permitir el acceso solamente a los usuarios con permisos. Contará además con interfaces sencillas y tendrá siempre visible la opción Ayuda posibilitando un mayor aprovechamiento de la aplicación por parte de los usuarios. El sistema será multiplataforma pues funcionará tanto en Windows como en Linux y el tiempo de respuesta del mismo dependerá de la cantidad de información a procesar, tratando de optimizar al máximo este proceso. Por otro lado se puede decir que no se encontraron punto críticos, pues como ya se mencionó los atributos de calidad están presentes. Además tanto el framework Symfony como el estilo MVC le permiten a la aplicación ser escalable, por lo que se le pueden incorporar funcionalidades y realizar cambios en el futuro sin que esto implique grandes complicaciones. Dándole la posibilidad de cumplir además con los atributos de Escalabilidad y Modificabilidad.

CONCLUSIONES GENERALES

En la investigación realizada se abordaron los aspectos fundamentales relacionados con la Arquitectura de Software, analizando las características de los principales estilos arquitectónicos y patrones de diseño, definiéndose el estilo Modelo Vista Controlador como parte de la solución del problema. Se analizaron además las características de un conjunto de tecnologías y herramientas para seleccionar las más adecuadas para el desarrollo del catálogo de mapas LiberMaps. Una vez definida la arquitectura del sistema se realizó la evaluación de la misma. Finalmente se han arribado a las siguientes conclusiones generales:

- El estilo arquitectónico empleado en LiberMaps contribuyó a lograr un sistema más organizado y un entorno de trabajo bien definido, lo que representa una gran ventaja a la hora de identificar los problemas que puedan surgir durante el desarrollo.
- Los patrones de diseño incluidos en la propuesta arquitectónica para LiberMaps, son vitales en la definición de una correcta arquitectura de software, pues son soluciones a problemas que pueden darse mientras se realiza el diseño de la aplicación. Estos patrones permiten por tanto, refinar los componentes del software o los subsistemas del mismo.
- Symfony, a pesar de no ser uno de los mejores frameworks de PHP en cuanto a rendimiento, ofrece numerosas ventajas a la hora de desarrollar aplicaciones con un alto grado de calidad en menos tiempo, ahorrándole al programador tener que generar sentencias SQL y logrando sistemas más seguros.
- La evaluación realizada a la propuesta arquitectónica fue fundamental para comprobar si la misma cumplía con los atributos de calidad definidos así como generar la documentación asociada.

RECOMENDACIONES

Se recomienda:

- Perfeccionar la arquitectura continuamente a través del ciclo de desarrollo.
- Realizar la evaluación de la arquitectura una vez concluido el proceso de implementación del sistema.

REFERENCIAS BIBLIOGRÁFICAS

- Bravo, Javier Domínguez**.** ***"Breve Introducción a la Cartografía y a los Sistemas de Información Geográfica (SIG)"*****. Octubre de 2000.**
- Álvarez de Sayaz, Carlos. ***Metodología de la Investigación Científica.*** **Santiago de Cuba: Universidad de Oriente, 1995.**
- Wolf, Dewayne E. Perry y Alexander L. ***"Foundations for the study of software architecture".*** **Octubre de 1992.**
- Clements, Paul. ***"A Survey of Architecture Description Languages".*** **Alemania: s.n., 1996.**
- Garlan, David. ***"Software Architecture: A Roadmap".*** **s.l. : Anthony Finkelstein, 2000.**
- Pressman, Roger. ***Ingeniería del Software, un enfoque práctico.*** **2002.**
- Kicillof, Carlos Reynoso y Nicolás. ***Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** **s.l.: UNIVERSIDAD DE BUENOS AIRES, 2004.**
- Garlan, Robert Allen y David. ***"The Wright Architectural Description Language".*** **Carnegie Mellon University : Technical Report, 1996.**
- Bass, L., Barbacci, M., Carriere, J., Kazman, R., Klein, M., y Lipson, H. ***"Attribute-based architectural styles".*** **Carnegie Mellon University: Technical Report, 1999.**
- Doberkat, Ernst-Erich. ***"Pipes and filters: Modelling a software architecture through relations".*** **Universidad de Dortmund: s.n., 2002.**
- Shaw, David Garlan y Mary. ***Software Architecture: Perspectives on an emerging discipline.*** **s.l. : Prentice Hall, 1996.**
- Shaw, David Garlan y Mary. ***"An introduction to software architecture".*** **1994.**

- Burbeck, Steve. ***"Application programming in Smalltalk-80: How to use Model-View-Controller (MVC)".*** **University of Illinois in Urbana-Champaign: Smalltalk Archive, 1992.**
- Szyperski, Clemens Alden. ***Component software: Beyond Object-Oriented programming.*** **s.l. : Addison-Wesley, 1998.**
- Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. **"*Pattern – Oriented Software Architecture. A System of Patterns."* Inglaterra. : John Wiley & Sons, 1996.**
- Camacho, Erika., Cardeso, Fabio., Núñez, Gabriel. ***Arquitecturas de Software, Guía de estudio.*** **2004.**
- Reynoso, Carlos. ***Introducción a la Arquitectura de Software.*** **2004.**
- Gamma, Erich., Helm, Richard., Johnson, Ralph., John Vlissides. ***Design Patterns: Elements of Reusable Object-Oriented Software.*** **s.l. : Addison-Wesley, 1995.**
- Rozo, Fabio Andrés Herrera. ***Informe 4. Servidores de Mapas.*** **Santiago de Cali. : s.n., 2007.**
- Larman, C. ***UML y PATRONES. Introducción al análisis y diseño orientado a objetos.*** **s.l.: P. Hall, Editor., 1999.**
- Flores, Amaya., Estrada Elvira., Espinosa Carlos., Manrique Joel. 2003. ***Resumen del libro "Lenguaje Unificado de odelado".*** **s.l. : Addison Wesley, 2003.**
- M Barbacci, M Klein, TLongstaff, C Weinstock . **Quality Attributes. Carnegie Mellon University : Technical Report, 1995.**
- Bass, L., Clements, P., & Kazman, R. **Software Architecture in practice. s.l. : Addison-Wesley., 1998.**
- Kazman, R., Clements, P., Klein, M. **Evaluating Software Architectures. s.l. : Addison Wesley, 2001.**

- 610.12, IEEE Std. **IEEE Standard Glossary of Software Engineering Terminology - Description. 1990**
- Booch, G., Rumbaugh, J., & Jacobson, I. T**he UML Modeling Language. s.l. : Adisson-Wesley, 1999.**
- Bosch, J. **Design & Use of Software Architectures. s.l. : Addison-Wesley., 2000.**
- Regalado, Yamila Vigil. ***Metodología de Evaluación de Arquitecturas de Software.*** **Ciudad de la Habana : s.n., 2009.**
- Salvador, Omar Gómez. ***Evaluando Arquitecturas de Software. Parte 1. Panorama General.*** **México : Brainworx S.A, 2007. 1870-0888.**
- Tristan Ravoallan, Xavier Lacot, Vincent Lamaire. **Frameworks PHP pour le' Enterprise. París : Claver Age, 2005.**

ANEXOS

Anexo 1:

Tabla 5 -- Definición de las evaluaciones de la Arquitectura

Nombre de la Evaluación:	Evaluación Inicial			
No. De la Evaluación:	1	**Responsable:**	Arquitecto	
Participantes:	Equipo de Arquitectura del proyecto			
Momento del ciclo de Desarrollo del Sistema				
Flujos de Trabajo	**Fases**			
	Inicio	**Elaboración**	**Construcción**	**Transición**
Modelación del Negocio				
Requerimientos				
Análisis y Diseño		**x**		
Implementación				
Prueba				
Despliegue				
Configuración y Cambios				
Gestión de Proyecto				
Ambiente				
Breve descripción del motivo de la Evaluación:	Verificar la existencia de los atributos de calidad definidos			
Justificación del momento seleccionado:	Es el momento donde se tiene la lista de requisitos funcionales y no funcionales, y es una primera aproximación a la ASW ha utilizar.			

Anexo 2:

Tabla 6 -- Definición de los atributos de calidad

Atributo de Calidad:	Funcionalidad	**Tipo de Atributo:**	Dinámico
Breve Descripción del Atributo:			
Habilidad del sistema para realizar el trabajo para el cual fue concebido.			
Atributo de Calidad:	Eficiencia	**Tipo de Atributo:**	Dinámico
Breve Descripción del Atributo:			

El sistema responde con la rapidez apropiada a las exigencias del usuario.			
Atributo de Calidad:	Seguridad	**Tipo de Atributo:**	Dinámico
Breve Descripción del Atributo:			
Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.			
Atributo de Calidad:	Portabilidad	**Tipo de Atributo:**	Estático
Breve Descripción del Atributo:			
Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.			
Atributo de Calidad:	Usabilidad	**Tipo de Atributo:**	Dinámico
Breve Descripción del Atributo:			
Facilidad con la cual las personas que interactúan con la aplicación en forma efectiva, y sobre como el sistema provee soporte al usuario en este sentido.			

Anexo 3:

Tabla 7 -- Comportamiento Esperado de los Atributos de Calidad

Atributo de Calidad:	Funcionalidad	**Tipo de Atributo:**	Dinámico
Descripción de su Comportamiento		**Elementos que debe tener la Arquitectura para logar satisfacer el Atributo:**	
Alto	**Medio**	**Bajo**	
X			Una BD con alta seguridad y funcionalidad. Un servidor de mapas con alta funcionalidad. Debe tener tratamiento de errores para la entrada de datos en la BD. Debe permitir la estabilidad del sistema ante errores.
Atributo de Calidad:	Eficiencia	**Tipo de Atributo:**	Dinámico
Descripción de su Comportamiento		**Elementos que debe tener la Arquitectura para logar satisfacer el Atributo:**	
Alto	**Medio**	**Bajo**	
X			Debe asegurar el acceso a los datos contenidos en la BD en el momento solicitado de la manera solicitada en breve tiempo.
Atributo de Calidad:	Seguridad	**Tipo de Atributo:**	Dinámico
Descripción de su Comportamiento		**Elementos que debe tener la Arquitectura para logar satisfacer el Atributo:**	
Alto	**Medio**	**Bajo**	
X			Una BD con alta seguridad para el acceso.

Atributo de Calidad:	Portabilidad		Tipo de Atributo:	Estático
Descripción de su Comportamiento			Elementos que debe tener la Arquitectura para logar satisfacer el Atributo:	
Alto	Medio	Bajo		
	X		El sistema debe poder desplegarse y usarse sobre las principales plataformas existentes.	
Atributo de Calidad:	Usabilidad		Tipo de Atributo:	Dinámico
Descripción de su Comportamiento			Elementos que debe tener la Arquitectura para logar satisfacer el Atributo:	
Alto	Medio	Bajo		
	X		El sistema debe brindar una interfaz amigable, intuitiva y entendible por los clientes. Debe tener siempre visible la opción de Ayuda, para posibilitar un mejor aprovechamiento por parte de los usuarios de sus funcionalidades.	

Anexo 4:

Tabla 8 -- Grado de Prioridad de los Atributos de Calidad

Atributos de Calidad	Prioridad		
	Alta	Media	Baja
Funcionalidad	X		
Seguridad	X		
Eficiencia	X		
Usabilidad		X	
Portabilidad		X	

Anexo 5:

Tabla 9 -- Escenario # 1. Acceso al sistema

Nombre del Escenario:	Acceso al sistema			
Origen del Estímulo:	Usuario			
Estímulo	Ambiente	Componente	Respuesta	Medida de la Respuesta
Solicita acceso al sistema.	En cualquier momento	Frameworks Symfony y EXTJS, BD.	Acceso al sistema y otorgar privilegios.	2.05 Segundos.
Comportamiento de los Atributos de Calidad:				
Atributo:	¿Presente Cómo?		¿Ausente Por qué?	
Seguridad	El sistema permite el			

	acceso a la información requerida. A partir de que se verifica su validez como usuario, se carga la aplicación según su perfil.	
Eficiencia	Lo hace en 2.05 segundos.	
Funcionalidad	El sistema permite el acceso del personal autorizado.	
Usabilidad	El sistema es intuitivo y fácil de entender.	
Interacción entre los Actores del Sistema:		
No hay en este caso		
Interacción entre Componentes Arquitectónicos:		
1. EXTJS muestra la interfaz de autenticación.	2. El Framework Symfony mapea la BD.	
	3. La BD busca y valida los datos.	
	4. La BD otorga privilegios y acceso al sistema.	
5. EXTJS muestra la interfaz del sistema.		

Anexo 6:

Tabla 10 -- Grado de Prioridad de los Escenarios

Escenarios	Prioridad		
	Alta	Media	Baja
Acceso al Sistema	X		
Creación de perfiles de usuario	X		
Intento de acceso no autorizado al sistema	X		