Universidad de las Ciencias Informáticas

Título: "Desarrollo de la Arquitectura del Sistema Automatizado de Control de Gestión de Indicadores de Refinación, SACGIR"

Trabajo de Diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autores:

Carlos Enrique Hernández Reyes.

Julio Alberto Leyva Durán.

Tutor:

Lic. David Silva Barrera.

Consultor:

Ing. Manuel Alejandro Gil Martín.

Ciudad de la Habana, Julio, 2008

DECLARACIÓN DE AUTORÍA

Ciudad de la Habana, Julio del 2008
"Año 50 de la Revolución"

Nosotros, Carlos Enrique Hernández Reyes y Julio Alberto Leyva Durán declaramos que somos los únicos autores de la presente investigación y reconocemos a la Universidad de las Ciencias Informáticas (UCI) a hacer uso de la misma en su beneficio.

Para que así conste firmo la presente a los __ días del mes de __ del año 2008.

Carlos E Hernández Reyes

Julio A Leyva Durán

Lic. David Silva Barrera

PENSAMIENTO

"Seamos realistas y hagamos lo Imposible"

Ché

Agradecimientos

Primeramente a La Revolución cubana y a nuestro Comandante en Jefe Fidel Castro Ruz, por su brillante idea de crear esta Universidad.

A nuestros padres y familiares que tanto esfuerzo y empeño han puesto por ver este sueño hecho realidad.

A nuestrso compañeros y amigos por estar juntos estos maravillosos 5 años…

A los que se van, a los que se quedan y a los que no llegaron, suerte en la vida…

A nuestro turor, al equipo de trabajo de SACGIR y todos los que de una forma u otra hicieron posible la realización de este trabajo.

Dedicatoria

A mis padres y hermana por ser mis guias en todo momento.

Mami te agradezco por todo lo que has hecho en la vida por mí, solo de darme la posibilidad de venir al mundo fue lo más hermoso que cualquier ser humano puede pedir.
Muchas gracias, siempre estarás conmigo.

A mi padre por engendrarme todo el respeto, el amor y el cariño que me dió.
Este es un sueño hecho realidad para él, a tí te lo dedico.

A mi hermanita del alma, mi tata, más que una hermana, mi amiga, mi todo, para tí también va este momento de felicidad, te quiero mucho.

A mi sobrinito, Daniel Enrique, espero que seas un buen hombre en la vida, tienes una bella familia, cuida a tu madre, que ellas son el ser más hermoso del mundo.

A todos mis amigos de la UCI en especial a Julio A por ser mi hermano en estos 5 años, gracias man!!!!, jajaja.....

Carlos E.

Dedicatoria

A mis padres y familiares por ser mis guias y ejemplo en todo momento.

A mi madre por la educación que siempre me ha enseñado, y estar en los momentos cuando más la he necesitado. A mi padre, gracias por su respeto y guía en las decisiones que he tomado.

A mis abuelos por ser el gran ejemplo en mi vida, en especial a mi abuela Delmis por apoyarme y ayudarme en todo momento, por su ejemplo de sacrificio y sabiduría.

A mis tios por ayudarme y aconsejarme siempre, en especial a la memoria de mi tia Nuris,en donde quieras que estes...

A mis hermanos y primos por estar junto a mi, les deceo que sigan con el ejemplo que siempre le han dado nuestra familia.

En especial también a mis amistades de la UCI, por acompañarme estos 5 maravillosos años, a mi gran amigo luis Adonis, Yadisnel, Ernesto y Yuniesky por su confianza y apoyo incondicionalidad en todo momento, a mi compañero de tesis y amigo Carlito por apoyarme durante estos 5 años...muchas gracias!!!

A todos muchas gracias...

Julio A.

DATOS DE CONTACTO

Síntesis del Tutor: Lic. David Silva Barrera

Profesión: Lic. Ciencias de la Computación

Categoría docente: Instructor

Años de graduado: 5

Correo: dsilva@uci.cu

Síntesis del Consultante: Ing. Manuel Alejandro Gil Martín

Profesión: Ing. Informática

Categoría docente: Especialista de la dirección de informatización y Arquitecto principal de la Universidad de las Ciencias Informáticas.

Años de graduado: 2

Correo: mag@uci.cu

Resumen

El presente trabajo de diploma está enmarcado en el proyecto Productivo SACGIR de la Universidad de las Ciencias Informáticas, en la facultad 9; debido al convenio Cuba-Venezuela en la rama de la industria del petróleo.

La solución propuesta es diseñar una Arquitectura de Software, línea base para el proyecto SACGIR, el cual tiene como objetivo la gestión de datos de Refinación, debe cumplir con los requerimientos del negocio y facilitar los procesos claves del mismo, además, de permitir el control de acceso a datos, funcionalidades de consulta, reporte de información, brindar vistas de indicadores para la evaluación de los mismos y almacenar los datos históricos.

Este trabajo contiene un estudio de los principales elementos que constituyen la Arquitectura de Software, partiendo de sus principales conceptos, tendencias de los patrones y estilos arquitectónicos, además de las herramientas de modelado y desarrollo, así como los Frameworks de desarrollo para la Web, Entorno de Desarrollo (IDE), lenguaje de Programación y Gestor de Base de Datos.

El sistema SACGIR cuenta con cinco módulos, con responsabilidades cada uno.

A través del Documento de Descripción de la Arquitectura se describe la solución arquitectónica del sistema, incluyendo además otros artefactos como son las vistas arquitectónicas definidas por la metodología RUP.

Situación Problemática

Vivimos en un mundo unipolar donde los grandes monopolios, encabezados por el gobierno de los Estados Unidos, poseen las mayores riquezas del mundo; enriquecidos a costa de los países subdesarrollados, que sin maquinarias necesarias para la explotación de los yacimientos minerales, son saqueados por estos grandes países industrializados.

Un ejemplo de este saqueo, era la situación que afrontaba la República Venezolana a finales del siglo XX, que era un abastecedor seguro del llamado oro negro de los Estados Unidos, sin recibir un precio justo, ni ser dueño absoluto de sus refinerías, este país era títere de los antojos consumistas estadounidenses.

Con la llegada del Presidente Hugo Chávez Frías a Venezuela la situación hoy en día es diferente, aplicó leyes socialistas a favor de su país, siendo una de las más importantes: La Nacionalización de las Industrias Petroquímicas, trayendo esta una nueva tarifa de adquisición del combustible para los EEUU.

La respuesta del gobierno estadounidense fue una política sucia, apoyando a la oposición del país venezolano para crear un caos que llevara al colapso del nuevo gobierno. Como resultado de esta política en el año 2002 se produce el paro de las industrias del petróleo (PDVSA), saboteándose los sistemas de comunicaciones, sistemas de gestión y control de la compra y venta de los derivados del petróleo.

Esta situación no tuvo grandes acontecimientos, muy pronto el gobierno venezolano retomó el control de todo, incluyendo las refinerías. Pero algo grave había ocurrido, varias personas que mediaban con los equipos de almacenamientos de información, habían abandonado sus puestos de trabajos sin dejar las claves de accesos a estos servidores.

El gobierno venezolano toma medidas urgentes y la principal es pedir ayuda al gobierno cubano debido a sus estrechas relaciones que existen entre los hermanos países, la cual fue cambiar la plataforma de software propietario al software libre.

El gobierno cubano le asignó la responsabilidad de asumir este importante proyecto a la Universidad de las Ciencias Informáticas (UCI), con el objetivo de darle solución con la calidad requerida en el menor tiempo posible.

Con la ayuda brindada por los especialistas se detecta que en la Gerencia de Planificación y Gestión de Refinación de PDVSA existen varios sistemas propietarios (SIMP, START) entre otros, que proporcionan información de forma separada y que son almacenadas en diferentes bases de datos, esta información es brindada a los directivos de la empresa en diversos formatos, lo cual dificulta el análisis y procesamiento de la información. La solución a esta problemática es crear un sistema

automatizado único para la gestión de datos de Refinación que cumpla con los requerimientos del negocio y facilite los procesos claves del mismo, además, que permita el control de acceso a los datos, funcionalidades de consulta, reporte de información, vistas de indicadores para la evaluación de los mismos y almacene los datos históricos.

A partir de toda esta información recopilada, se crea a finales del 2006 en la facultad nueve de la universidad, el proyecto llamado “Sistema Automatizado de Control de Gestión de Indicadores de Refinación”, SACGIR.

Problema científico

¿Cómo diseñar una arquitectura robusta y flexible a la vez, que permita responder a las necesidades del control de indicadores de refinerías?

Objeto de estudio

Proceso de diseño de una arquitectura basada sobre plataforma de software libre para el proyecto Sistema Automatizado de Control y Gestión de Indicadores de Refinería (en lo adelante SACGIR).

Objetivo general

Diseñar una arquitectura de software para el proyecto SACGIR, que permita al grupo de desarrollo tomar decisiones sobre los aspectos dinámicos y estáticos más significativos de la organización del sistema, la selección de elementos estructurales mediante los cuales se compone dicho sistema, sus interfaces, comportamiento; y que permita a los desarrolladores conocer la estructura del sistema que se está desarrollando.

Objetivos específicos

- Definir un marco arquitectónico que permita el desarrollo, de forma paralela, de la aplicación en cada una de las capas lógicas.
- Definir patrones de diseño útiles para el proyeto, según los problemas recurrentes que aparezcan durante el desarrollo del mismo.

Campo de acción

El diseño y descripción de la arquitectura base para el desarrollo del sistema SACGIR.

Hipótesis

Si se define un marco arquitectónico con la selección de patrones adecuados se logrará el diseño de una buena arquitectura robusta y flexible que responda a las necesidades del sistema a implementar.

Tareas de investigación:

- Realizar un estudio del negocio que permita identificar algunas propuestas de los estilos arquitectónicos.
- Investigar las diferentes propuestas arquitectónicas a partir de consultar la bibliografía especializada.
- Comparar las distintas propuestas de arquitecturas consultadas, teniendo en cuenta aspectos positivos, negativos.
- Revisar bibliografía científica teórica usada en el desarrollo de arquitecturas para sistemas sobre plataforma Web.
- Identificar patrones arquitectónicos a utilizar, así como fundamentación y aplicación de estos patrones.
- Definir las herramientas y tecnologías a utilizar para el desarrollo.
- Diseñar una propuesta arquitectónica que cumpla con los requerimientos de la aplicación y que garantice el desarrollo paralelo y la reutilización de componentes de la misma.

Para realizar dichas tareas de investigación se tiene en cuenta varios métodos:

Métodos teóricos:

- Análisis y Síntesis: Para el procesamiento de la información y arribar a las conclusiones de la investigación, así como precisar las características del modelo arquitectónico propuesto.

- Histórico – Lógico: Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos.

Índice

Figuras

Tablas

Anexos

Capítulo 1

Estado del Arte y Fundamentación teórica

1.1 Introducción

En este capítulo se abordará sobre el análisis del estado del arte de la Arquitectura de Software(AS en lo adelante), partiendo del estudio de sus principales conceptos, tendencias de los patrones y estilos arquitectónicos.

1.2 La Arquitectura de Software

La AS (1) es uno de los grandes temas de hoy en día en la Ingeniería de Software, es la organización fundamental de un sistema encarnado en sus componentes[1], las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución.

La AS aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. (2)

Edsger Dijkstra (3) propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera.

Fred Brooks utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa. (4)

De forma general una arquitectura de software se selecciona y diseña con base en objetivos y restricciones. Los objetivos son aquellos prefijados para el sistema de información, pero no solamente los de tipo funcional, también otros objetivos como la mantenibilidad, adaptabilidad, flexibilidad e interacción con otros sistemas de información. Las restricciones son aquellas limitaciones derivadas de las tecnologías disponibles para implementar sistemas de información. Algunas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas.

Se puede decir también que la AS establece los fundamentos para que los desarrolladores de un proyecto de software trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades.

Con lo anteriormente expuesto se puede inferir que la AS es el diseño de más alto nivel de la estructura de un sistema, brinda una vista general del mismo que se puede

[1] La idea de “componente” no es la de la correspondiente tecnología de desarrollo (COM, CORBA Component Model, EJB), sino la de elemento propio de un estilo (los estilos se abordará más adelante).

ver como un todo antes de comenzar el análisis del sistema. Está compuesta por componentes, definiendo el ¿qué? de la aplicación, posee relaciones y restricciones que definen el ¿cómo? realizar la aplicación, posee también requerimientos tanto funcionales (RF) como no funcionales (RNF) ligada más a este último que junto a las decisiones significativas definen el ¿por qué? de las decisiones a tomar. Además, permite a los miembros del grupo de desarrollo encaminar el producto por una línea de trabajo común, logrando alcanzar los objetivos propuestos.

1.3 Patrones

1.3.1 Definición de Patrones

Los desarrolladores con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, y se les da un nombre, podrían llamarse “Patrones”.

De manera más simple un patrón es un par problema/solución con nombre, que se puede aplicar en nuevos contextos; con consejos acerca de como aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

Un patrón es una descripción de un problema bien conocido que suele incluir:

- Descripción.
- Escenario de Uso.
- Solución concreta.
- Las consecuencias de utilizar este patrón.
- Ejemplos de implementación.
- Lista de patrones relacionados.

1.3.2 Categorías de patrones

Según la escala o nivel de abstracción:

- **Patrones de arquitectura:** Aquéllos que expresan un esquema organizativo estructural fundamental para sistemas de software.
- **Patrones de diseño**: Aquéllos que expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.
- **Idiomas**: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

También es importante reseñar el concepto de Antipatrón de Diseño (5), que con forma semejante a la de un patrón, intenta prevenir contra errores comunes de diseño en el software. La idea de los Antipatrones es dar a conocer los problemas que acarrean ciertos diseños muy frecuentes, para intentar evitar que diferentes sistemas acaben una y otra vez en el mismo callejón sin salida, por haber cometido los mismos errores.

1.4 Estilos y Patrones Arquitectónicos

Un estilo arquitectónico define a una familia de sistemas en términos de un patrón de organización estructural, según Mary Shaw y David Garlan (6). Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado, así como un conjunto de restricciones de cómo pueden ser combinados. Robert T. Monroe (7) plantea que un estilo arquitectónico provee una colección de elementos edificadores del diseño en bloque, reglas y restricciones para componer los bloques constructivos, y las herramientas para analizar y manipular los diseños creados en el estilo. Los estilos generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado.

1.4.1 Ejemplo de algunos estilos Arquitetónicos

- **Estilos de Flujo de Datos**
  - Tubería y filtros.
- **Estilos Centrados en Datos**
  - Arquitecturas de Pizarra o Repositorio.
- **Estilos de Código Móvil**
  - Arquitectura de Máquinas Virtuales.
- **Estilos Peer-to-Peer**
  - Arquitecturas Basadas en Eventos.
  - Arquitecturas Orientadas a Servicios (SOA).
  - Arquitecturas Basadas en Recursos.
- **Estilos de Llamada y Retorno**
  - Model-View-Controller (MVC).
  - Arquitecturas en Capas.
  - Arquitecturas Orientadas a Objetos.
  - Arquitecturas Basadas en Componentes.

A continuación se detallarán los patrones referentes más conocidos por los AS:

1.4.1.1 Arquitectura en Capas (8)

Los sistemas o arquitecturas en capas constituyen uno de los patrones que aparecen con mayor frecuencia mencionados, o, por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente. 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 algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

Casos representativos de este estilo son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware. El ejemplo más característico es el modelo OSI con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación. El estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos.

1.4.1.2 Arquitecturas Orientadas a Objetos (AOO) (8)

Los nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. En la semblanza de estos autores curiosamente no se establece como cuestión definitoria el principio de herencia. Ellos piensan que, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Resumiendo las características de las arquitecturas OO, se podría decir que:

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

1.4.1.3 Arquitecturas Orientadas a Servicios (SOA)

Lo que SOA es:

Este estilo de Arquitectura, es una metodología o una estrategia en la cual las aplicaciones hacen uso de los servicios disponibles en la red, estos servicios representan procesos de negocio y que se combinan entre sí para ofrecer soluciones adecuadas a las diferentes necesidades de negocio, ofreciendo un marco de trabajo para alinear los procesos de negocio con los sistemas de las Tecnologías de la Información.

Lo que SOA no es:

Una tecnología.
Una herramienta.
Servicios Web.
Arquitectura para todo tipo de aplicaciones.
Una arquitectura de rápida implementación.

¿Por qué Utilizar SOA?

- **Abstracción del lenguaje:** Antes de que surgiera SOA los desarrolladores hablaban de interfaz, clases, tablas, consultas y los directivos de negocio sólo hablaban de procesos de negocio y para entonces, lograr un entendimiento entre ellos era muy difícil. Mientras con SOA un servicio de negocio para ambas partes es lo mismo. La posibilidad de que ambas partes tantos los directivos de negocio como los desarrolladores hablen un lenguaje común posibilita que puedan entenderse y tomar decisiones juntos.

- **Abstracción de tecnología:** Las empresas actuales están muy vinculadas a la habilidad que esta posea para adaptar sus tecnologías a los frecuentes cambios y desafíos del negocio. Una SOA permite que las tecnologías evolucionen a la par del negocio.
- **Flexibilidad de la funcionalidad:** Los bloques que se componen para armar una solución, responden a las necesidades del cliente, deja de ser una solución genérica para convertirse en algo hecho a la medida. Al observar la estructura antigua del software, se asemeja a un bloque armado, funcional pero estático. ¿Qué pasaría si cambia el día de mañana algunas de las funcionalidades que presenta? Si el software es estático e inflexible. Pues inevitablemente para mejorar es necesario cambiarlo todo.

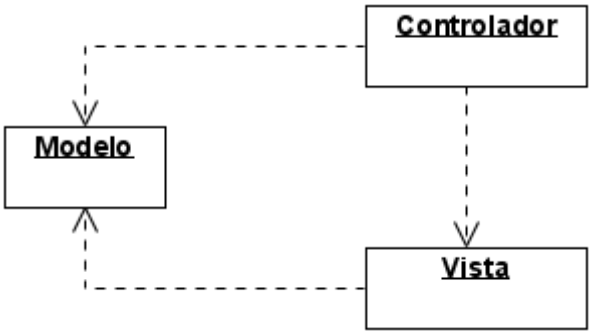
1.4.1.4 El patrón conocido como Modelo-Vista-Controlador (MVC) (8)

Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres partes diferentes:

- **Modelo.** El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

- **Vista.** Maneja la visualización de la información.

- **Controlador**. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.



Figura# 1 Patrón MVC

1.5 Patrones de Diseño

1.5.1 Definición de Patrones

Los desarrolladores con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, y se les da un nombre, podrían llamarse “Patrones”.

De manera más simple un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de como aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

Un patrón es una descripción de un problema bien conocido que suele incluir:

- Descripción.
- Escenario de Uso.
- Solución concreta.
- Las consecuencias de utilizar este patrón.
- Ejemplos de implementación.
- Lista de patrones relacionados.

1.5.2 Patrones generales de software para asignar responsabilidades (GRASP)

Lo patrones de GRASP, no compiten con los patrones de diseño, sino que sirven de guía para encontrar los patrones de diseño (que son más concretos).

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

En cuanto a las responsabilidades UML define una responsabilidad como “un contrato u obligación de un clasificador”.

Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento.

Básicamente, estas responsabilidades son de los siguientes dos tipos:

Conocer*:*

- Conocer los datos privados encapsulados.
- Conocer los objetos relacionados.
- Conocer las cosas que puede derivar o calcular.

Hacer*:*

- Hacer algo él mismo, como crear un objeto o hacer un cálculo.
- Iniciar una acción en otros objetos.
- Controlar y coordinar actividades en otros objetos.

1.5.2.1 Principales Patrones GRASP (9)

Dentro de los patrones GRASP se destacan cinco 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**: Debe haber pocas dependencias entre las clases.
- **Controlador**: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.

Además existen cuatro patrones GRASP adicionales:

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

1.5.3 Patrones de Diseño

Los patrones de diseño (design patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

1.5.3.1 Objetivos de estos patrones

Los patrones de diseño pretenden:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Así mismo, no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.
- No es obligatorio utilizar los patrones siempre, sólo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable. Abusar o forzar el uso de los patrones puede ser un error.

1. 5.4 Principales patrones GoF (Gang of Four) (10).

1.5.4.1 Patrones creacionales

- **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.
- **Builder (Constructor virtual)**: Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- **Factory Method (Método de fabricación)**: Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- **Prototype (Prototipo)**: Crea nuevos objetos clonándolos de una instancia ya existente.

- **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.

1.5.4.2 Patrones Estructurales

- **Adapter (Adaptador)**: Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- **Bridge (Puente)**: Desacopla una abstracción de su implementación.
- **Composite (Objeto compuesto)**: Permite tratar objetos compuestos como si de uno simple se tratase.
- **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.
- **Proxy**: (**Apoderado**) Mantiene un representante de un objeto.

1.5.4.3 Patrones de Comportamiento

- **Chain of Responsibility (Cadena de responsabilidad)**: Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- **Command (Orden)**: Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- **Interpreter (Intérprete)**: Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- **Iterator (Iterador)**: Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- **Mediator (Mediador)**: Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- **Memento (Recuerdo)**: Permite volver a estados anteriores del sistema.
- **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.

- **Strategy (Estrategia)**: Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- **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.
- **Visitor (Visitante):** Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

1.6 Frameworks

1.6.1 Algunas definiciones

A continuación se muestran algunas definiciones de los framework de desarrollo (11):

- Conjunto de clases que cooperan y forman un diseño reutilizable para un tipo específico de software. Un framework ofrece una guía arquitectónica partiendo el 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. (10)
- Infraestructura software que crea un entorno común para integrar aplicaciones e información compartida dentro de un dominio dado. (12) (13)
- Es una aplicación semicompleta que contiene componentes estáticos y dinámicos que pueden ser personalizados para obtener aplicaciones de usuario específicas. (14)

Con lo anterior expuesto, se puede arribar a la conclusión de que un framework es una base, donde se representa o define una arquitectura adaptada a las particularidades de un determinado dominio de aplicación, conteniendo o definiendo, un conjunto de clases, de forma abstracta una serie de componentes y sus interfaces, estableciendo las reglas y mecanismos de interacción entre ellos, los frameworks son diseñados con el intento de facilitar el desarrollo de software, incluyen la implementación de algunos componentes o incluso varias implementaciones alternativas, algunos pueden soportar un tipo de AS en específico (se abordará en el capítulo 2).

1.6.2 Framework para el desarrollo de aplicaciones web

Conjunto de clases que cooperan y forman un diseño reutilizable formando una infraestructura que facilita el desarrollo de aplicaciones web.

A continuación se muestra una tabla con algunos de los frameworks utilizados por la comunidad de desarrolladores web:

Framework	PHP4	PHP5	MVC	MúlDB's	ORM	DB Obj	Caching	Validación	Ajax
Zend	-	✔	✔	✔	-	✔	✔	✔	-
CakePHP	✔	✔	✔	✔	✔	✔	✔	✔	✔
Symfony	-	✔	✔	✔	✔	✔	✔	✔	✔
Seagull	✔	✔	✔	✔	✔	✔	✔	✔	-
WACT	✔	✔	✔	✔	-	✔	-	✔	-
Prado	-	✔	-	✔	-	-	✔	✔	✔
PHP on TRAX	-	✔	✔	✔	✔	✔	-	✔	✔
ZooP	✔	✔	✔	✔	-	✔	✔	✔	✔
eZ Components	-	✔	-	✔	-	✔	✔	✔	-
CodeIgniter	✔	✔	✔	✔	-	✔	✔	✔	-
Kumbia	-	✔	✔	✔	✔	✔	✔	✔	✔

Tabla #1 framework de desarrollo Web.

PHP: Lenguaje de programación ***P****HP* ***H****ypertext* ***P****re-processor* (inicialmente PHP Tools, o, *P*ersonal *H*ome *P*age Tools).

- PHP 4: versión no Orientada a Objeto.
- PHP 5: versión avanzada Orientada a Objeto.

MVC: Patrón Modelo Vista Controlador.

MúlDB's: Soporte múltiple a varios Gestores de Bases de Datos.

ORM: Mapeo Objeto Relacional (Object-Relational Mapping), permite transformar un registro en objeto y viceversa.

DB Obj: Conseguir un objeto de la base de datos.

Caching: Los tipos de caché que están integrados al framework.

Validaciones: Validación de información.

Ajax: JavaScript asíncrono (Asynchronous JavaScript), mejora la interacción del usuario con la aplicación.

Como se puede apreciar en la tabla anterior, existen varios frameworks que proporcionan mucha de las funcionalidades requeridas para el desarrollo de la aplicación, en este caso particular sólo se puede contar con cuatro de ellos, debido a que están aprobados por la Universidad de las Ciencias Informáticas, estos son: CakePHP, Symfony, PHPPrado y Kumbia.

1.7 Metodología de Desarrollo

1.7.1 Proceso Unificado de Desarrollo (RUP)

RUP es un proceso de desarrollo de software, las principales características de ser : Iterativo e Incremental, Dirigido por Casos de Uso y Centrado en la Arquitectura, en esta última se involucra los elementos más significativos del sistema, influenciados principalmente por plataformas de software, sistemas operativos, manejadores de Bases de Datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales, todos estos elementos se resumen en varias vistas, denominadas como el modelo de 4+1 de la Arquitectura, recibiendo el nombre(como se representó anteriormente en UML), vistas lógicas, Implementación, proceso y despliegue, más la rectora , vista de casos de usos.

RUP no es una herramienta para construir software, es una metodología que brinda al equipo de desarrollo una idea de como realizar el sistema. Es una metodología Orientada a Objeto.

1.7.2 Fases de RUP

Presenta cuatro fases por la cual transita el software, las cuales son:

- **Inicio o Conceptualización.**
- **Elaboración.**
- **Construcción.**
- **Transición.**

En las fases del RUP (ya mencionadas previamente) la arquitectura tiene una gran participación, ejemplo:

Fase de Inicio: Se presenta una arquitectura candidata, en la cual ya se sacan algunos de los casos de uso arquitectónicamente significativos que incluyen los que son más necesarios para el cliente en esta versión o quizás en versiones futuras.

Fase de elaboración: Se realiza por flujos, en el primero (aproximadamente), se determinó un diseño de alto nivel para la arquitectura. Después se formó una arquitectura en un par de construcciones dentro de la primera iteración.

Fase de construcción: En el primer flujo de construcción se trabaja con las partes generales de la aplicación, que son generales en cuanto al dominio, y que no son específicas del sistema que se va a desarrollar(es decir, seleccionar el software del sistema, los sistemas heredados, los estándares y las políticas de uso).
También se decide cómo manejar los requisitos generales no funcionales, así como la disponibilidad de estos requisitos.

En el segundo flujo de **construcción**, se trabaja con los aspectos de la arquitectura específicos de la aplicación. Se escogen un conjunto de casos de uso relevantes en cuanto a la arquitectura, se capturan los requisitos, se analizan, se diseñan, se implementan y se prueban. Los resultados serán nuevos subsistemas implementados como componentes de desarrollo que soportan los casos de uso seleccionados. Este mecanismo se realiza una y otra vez hasta terminar con las iteraciones. Si este final de las iteraciones tiene lugar en el final de la fase de elaboración, entonces se habrá conseguido una arquitectura estable.

Es necesaria una AS que describa los elementos del modelo más importantes para el sistema. Tiene una gran importancia ya que guía el equipo de trabajo con el sistema, tanto en este ciclo como a través del ciclo de vida del sistema completo. Estos elementos significativos, arquitectónicamente hablando, incluyen algunos de los subsistemas, dependencias, interfaces, colaboraciones, nodos y clases activas. Describen los cimientos del sistema, que son necesario como base para comprenderlo, desarrollarlo y producirlo económicamente. Además de abarcar decisiones importantes sobre el sistema.

1.7.3 Decisiones que abarca la AS (2)

- ✓ La organización del sistema software.
- ✓ Los elementos estructurales que compondrán el sistema y sus interfaces, junto con sus comportamientos, tal y como se especifican en las colaboraciones entre estos elementos.
- ✓ La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- ✓ El estilo de la arquitectura que guía esta organización: los elementos y sus interfaces, sus colaboraciones y su composición.

Sin embargo, la AS está afectada no sólo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos, tecnológicos y la estética.

Resumiendo, una buena arquitectura permite obtener los casos de uso correctos, de manera económica. (2)

1.7.4 El rol de Arquitecto del Software

El arquitecto crea la arquitectura junto con otros desarrolladores. Trabajan para conseguir un sistema que tendrá un alto rendimiento y una calidad, y será completamente funcional.

El arquitecto posee la responsabilidad técnica más importante en estos aspectos y selecciona patrones de arquitectura y entre productos para establecer las dependencias entre subsistemas para cada uno de esos distintos intereses.

El verdadero objetivo es cumplir con las necesidades de la aplicación de la mejor forma posible con el estado actual de la tecnología y un coste que la aplicación pueda soportar, en otras palabras, ser capaz de implementar la funcionalidad de la aplicación(es decir, los casos de uso) de manera económica, ahora y en el futuro.

1.7.4.1 Principales Artefactos a desarrollar por el Arquitecto (15)

- Documento Descripción de la Arquitectura (4+1 Vistas)
- Modelo de Despliegue.
- Modelo de Implementación
- Protocolos.
- Interfaces.

1.7.4.2 Actividades Principales realizadas por el Arquitecto

- Análisis de la arquitectura.
- Priorizar los Casos de Uso.
- Identificar mecanismos de diseño.
- Estructurar el modelo de implementación.
- Reutilización de elementos de diseño existentes.
- Identificar los elementos de diseño.
- Describir la arquitectura en tiempo de ejecución.

1. 8 Lenguaje Unificado de Modelado (UML)

Un Proceso de Desarrollo de Software es la definición del conjunto de actividades que guían los esfuerzos de las personas implicadas en el proyecto, a modo de plantilla que explica los pasos necesarios para terminar el proyecto (2). Destacar que UML no es un lenguaje de programación, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto los representados por diagramas estáticos (Casos de Uso, diagrama de clases, etc.) como los dinámicos (Diagramas de actividades, interacción, etc.)

La representación en UML de un software está formado por 4+1 vistas o modelos parciales separados, relacionados entre sí, estas vistas son:

- **Vista de casos de uso**.
- **Vista lógica**.
- **Viste de implementación**,
- **Vista de procesos.**
- **Vista de despliegue**.

1.9 Herramientas CASE

Computer Aided Software Engineering (CASE) o Ingeniería de Software Asistida por Computación, traducido al español, estas herramientas permiten organizar y manejar la información de un proyecto informático. A medida que los sistemas que hoy se construyen se tornan más y más complejos, las herramientas CASE de modelado con UML ofrecen muchos beneficios para un proyecto, logrando aplicar la metodología de análisis y diseño orientados a objetos y abstraernos del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y

modificar, permite también la generación de la documentación y reutilización de componentes, logrando de forma general una mayor productividad y calidad.

1.9.1 Descripciones de algunas herramientas CASE propuestas

1.9.1.1 Visual Paradigm

Soporta notación UML 2.x, capacidades de ingeniería inversa y directa, generación de código, importación desde Racional Rose, generación de código e ingeniería inversa a la vez de los lenguajes: Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python. Adicional, soporta la generación de código en: C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C, and Ruby. Además del soporte de ingeniería inversa de: Java class, .NET (dll, exe), JDBC y ficheros mapeados de Hibernate.

- **Principales características**:
  - Licencia: Gratuita y Comercial.
  - Producto de calidad.
  - Soporta aplicaciones Web.
  - Varios idiomas.
  - Generación de código para Java y exportación como HTML.
  - Fácil de instalar y actualizar.
- **Integraciones IDE:**
  - MS Visio.
  - Plug-in.
  - Visual Studio.
  - IntelliJ IDEA.
  - Eclipse.
  - NetBeans.

Entre las nuevas características, incluye el modelado colaborativo con Concurrent Versions System (CVS) y Subversión, interoperabilidad con modelos UML a través de XMI, etc.

- **Plataformas de Sistemas Operativos:**
  - Linux.
  - MacOs.
  - Windows.

1.9.1.2 Rational Rose Interprise

Unifica todos sus equipos de desarrollo a través del modelamiento el cual está basado en UML. Soporta la generación de código a partir de modelos en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++® y Visual Basic®.

- **Integraciones IDE:**
  - Borland JBuilder (versiones 7.0 a 10.0).
  - Microsoft Visual Studio (versiones 2003 en adelante).
- **Diagramas:**
  - Clases, Componentes, Deployment, Secuencia, Statechart, Caso de Uso, Colaboración.
- **Características adicionales incluidas:**
  - Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
  - Capacidad de crear definiciones de tipo de documento XML para el uso en la aplicación.
  - Integración con otras herramientas de desarrollo de Rational.
  - Provee visualización, modelado y las herramientas para desarrollar aplicaciones de Web.
- **Plataformas de Sistemas Operativos apropiadas:**
  - Windows 2000.
  - Windows NT.
  - Windows XP.

Como se ha podido apreciar, las dos herramientas son bastante parecidas; pero se puede presumir que una candidata podría ser el Visual Paradigm, porque posee una característica fundamental: soporte multiplataforma. Además otra de las ventajas que posee el Visual Paradigm es que se adecua al entorno de desarrollo permitiéndoles a los diseñadores, analistas y a todo aquel que trabaje con el mismo, una mayor organización y claridad en el trabajo.

1.10 Gestores de Bases de Datos

1.10.1 MySQL

MySQL es un sistema de gestión de bases de datos relacionales, multi–hilo y multi-usuario. Una base de datos relacional almacena datos en tablas separadas en lugar de poner todos los datos en un gran almacén, lo que añade velocidad y flexibilidad. La parte SQL de "MySQL" se refiere a "Structured Query Language". SQL es el lenguaje estandarizado más común para acceder a bases de datos. MySQL Server trabaja en entornos cliente/servidor o incrustados. Además, funciona en diferentes plataformas y fue escrito en C y en C++. Tiene como una de sus principales ventajas la velocidad en la lectura de datos, pero a costa de eliminar un conjunto de facilidades que presentan otros SGBD: integridad referencial, bloqueo de registros, procedimientos almacenados.

MySQL es muy popular en aplicaciones web y actúa como un componente de bases de datos para las plataformas LAMP, MAMP y WAMP (Linux/MAC/Windows-Apache-MySQL-PHP/Perl/Python), también trabaja en numerosas plataformas como AIX, HP-UX, GNU/Linux, Mac OS X, Novell NetWare, OpenBSD, OS/2, Solaris, SunOS, y todas las versiones de Windows. Su mayor desempeño se logra cuando se combina con el lenguaje de programación PHP.

1.10.2 Postgres SQL

Postgres SQL, fue desarrollado por el equipo de Oracle, es totalmente libre. Se recomienda la utilización de Postgres SQL para la elaboración de un sistema robusto y para lograr mayor escalabilidad.

Es un Gestor de Bases de Datos objeto-relacional altamente extensible, soporta operadores y tipos de datos definidos por el usuario y es capaz de manejar complejas rutinas y reglas de modo que su avanzada funcionalidad se pone de manifiesto con las consultas SQL declarativas, el control de concurrencia multiversión, soporte multiusuario, transacciones, optimización de consultas, herencia y valores no atómicos (atributos basados en vectores y conjuntos). Además cuenta con un mejor soporte para subselects, triggers, vistas y procedimientos almacenados en el servidor, además tiene ciertas características orientadas a objetos. Este sistema cuenta con una API (Application Program Interface) flexible lo cual ha permitido dar soporte para el desarrollo con PostgreSQL en diversos lenguajes de programación entre los que se incluyen: Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.

Se hace necesario señalar que a diferencia de la mayoría de otros sistemas de bases de datos que usan bloqueos para el control de concurrencia, Postgres utiliza un modelo multiversión MVCC o Control de Concurrencia Multi-Versión (Multi-Version Concurrency Control), mediante el cual evita el problema de los bloqueos por completo. Esto significa que mientras se consulta una base de datos, cada transacción ve una imagen de los datos (una versión de la base de datos) como si fuera tiempo atrás, sin tener en cuenta el estado actual de los datos que hay por debajo.

1.11 Lenguajes de Programación

1.11.1 PHP

PHP es un acrónimo recursivo para "PHP Hypertext Pre-processor" (inicialmente PHP Tools o Personal Home Page Tools), su creador Rasmus Lerdorf ha recibido muchas contribuciones de otros desarrolladores debido a su política de código abierto. PHP es usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web, ofreciendo soluciones simples y universales para las paginaciones dinámicas web de fácil programación.

Soporte para bases de datos: MysSQL, PostgreSQL, Oracle, MS SQL Server, Sybase mSQL, Informix, entre otras.
Como producto de código abierto, también es multiplataforma y está ampliamente difundido en el mundo entre las comunidades de programadores.

1.11.2 Python

Python es un lenguaje de programación fácil de aprender y potente. Tiene eficaces estructuras de datos de alto nivel y una solución de programación orientada a objetos simple pero eficaz. La elegante sintaxis de Python, su gestión de tipo dinámica y su naturaleza interpretada hacen de él un lenguaje ideal para guiones (scripts) y desarrollo rápido de aplicaciones, en muchas áreas y en la mayoría de las plataformas.

Una de las virtudes de Python es la cantidad de alternativas existentes para el desarrollo de aplicaciones web, existen servidores de aplicaciones complejos y ligeros, plantillas para el desarrollo web y la posibilidad de usar python embebido en un documento html como si fuese PHP o ASP.

Soporte para bases de datos: MySQL, PostgreSQL, SQLite, Firebird, Sybase, MaxDB y MSSQLServer.

1.11.3 Tecnología AJAX

En realidad, el término de AJAX es un acrónimo de *Asynchronous JavaScript + UML,* que se puede que se puede traducir como "JavaScript asíncrono + UML".

El Libro (16) define AJAX de la siguiente forma:
"AJAX no es una tecnología en sí mismo. En realidad, se trata de varias tecnologías independientes que se unen de forma nuevas y sorprendentes."

Las tecnologías que forman AJAX son:

- XHTML y CCS, para crear una presentación basada en estándares.
- DOM, para la interacción y manipulación dinámica de la presentación.
- XML, XSLT y JSON, para el intercambio y manipulación de la información.
- XMLHttpResquest, para el intercambio asíncrono de información.
- JavaScript, para unir todas las demás tecnologías.

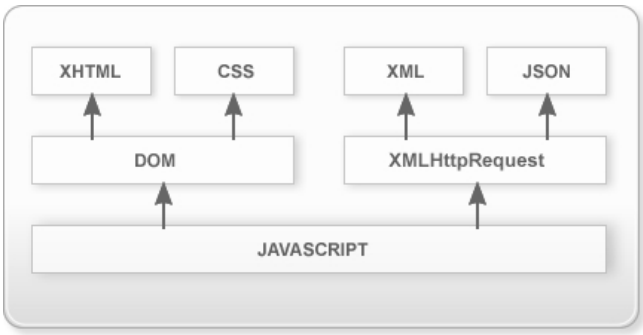
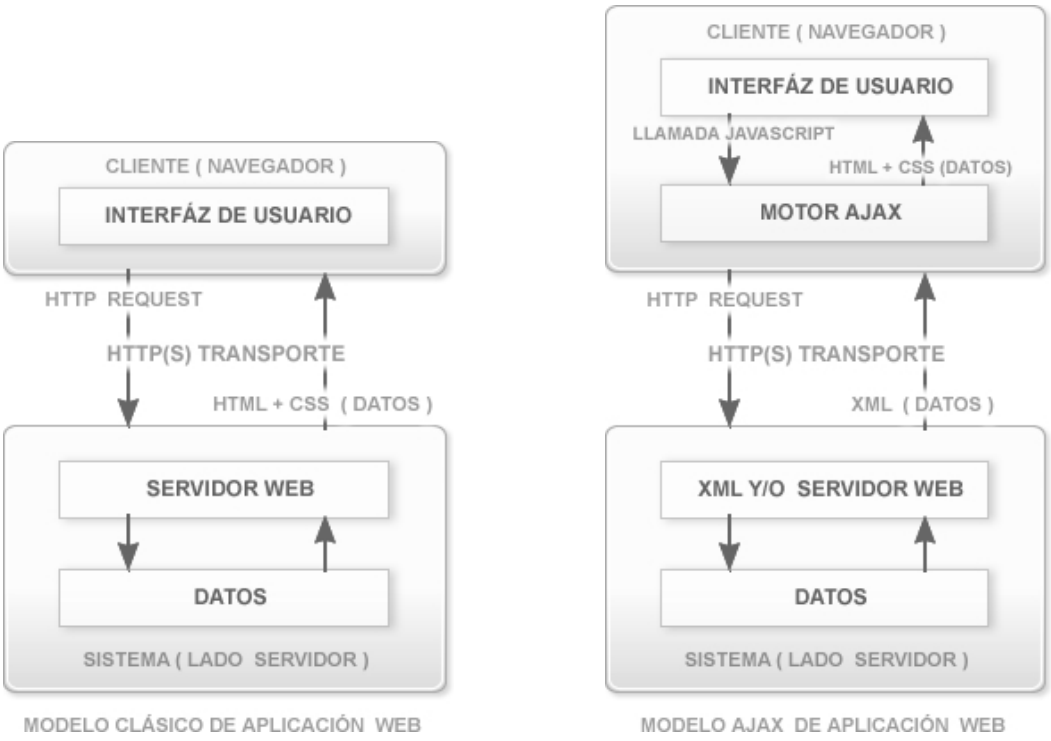


Figura #2. Tecnologías agrupadas bajo el concepto AJAX

Desarrollar aplicaciones en AJAX requiere un conocimiento avanzado de todas y cada una de las tecnologías anteriores.



Figura#3 Comparación gráfica del modelo tradicional de aplicación web y del nuevo modelo propuesto por AJAX.

En las aplicaciones web tradicionales, las acciones del usuario en la página (pinchar en un botón, seleccionar un valor de una lista, etc.) desencadenaban llamadas al servidor. Una vez procesada la información del usuario, el servidor devuelve una nueva página HTML al navegador del usuario.

En el siguiente esquema, la imagen de la izquierda muestra el modelo tradicional de las aplicaciones web. La imagen de la derecha muestra el nuevo modelo propuesto por AJAX:

AJAX permite mejorar completamente la interacción del el usuario con la aplicación evitando las recargas constantes de la página, ya que el intercambio de información con el servidor se produce en un segundo plano.

Las aplicaciones construidas con AJAX eliminan la recarga constante de páginas mediante la creación de un elemento intermedio entre el usuario y el servidor. La nueva capa intermedia de AJAX mejora la respuesta de la aplicación, ya que el usuario nunca se encuentra con una ventana del navegador vacía esperando la respuesta del servidor.

Las peticiones HTTP al servidor se sustituyen por peticiones JavaScript que se realizan al elemento encargado de AJAX. Las peticiones más simples no requieren intervención del servidor, por lo que la respuesta es inmediata. Si la interacción

requiere una respuesta del servidor, la petición se realiza de forma asíncrona mediante AJAX. En este caso, la interacción del usuario tampoco se ve interrumpida por recargas de páginas o largas esperas por la respuesta del servidor.

Es necesario decir que por la inmensidad de datos que la aplicación deberá procesar, brindar y por la cantidad de usuarios que soportará esta, se decidió utilizar esta mezcla de tecnologías que se hacen llamar AJAX.

1.12 Ambiente de desarrollo (IDE)

El ambiente de desarrollo (Development Environment) es algo imprescindible en la producción de software. Es donde se definen el conjunto de herramientas y tecnologías (frameworks), versiones a usar y su integración, que intervienen en un proceso de desarrollo de software.

1.12.1 Eclipse

Eclipse es como una tienda donde no solamente se hacen productos, sino que además se hacen las herramientas para hacer los productos. Eclipse contiene un equipo de instrumentos para desarrollo en Java o Java Development Toolkit (JDT) para escribir y depurar programas en Java; además se obtiene un ambiente de desarrollo de plugin Plug-in Development Enviroment para heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada además que el JDT. Esto es para lo que la mayoría las personas usan Eclipse.

Aunque Eclipse es escrito en Java y su principal uso es como IDE para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plug-in, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#, PHP. En principio permite ejecutar un programa sobre cualquier plataforma. Es una extensible plataforma de código abierto para desarrollar herramientas.

1.12.2 Zend Studio

Zend Studio se ha diseñado para una amplia gama de programadores y existen dos ediciones: Standard y Professional. Zend Studio, concebido con el fin de crear aplicaciones altamente fiables, proporciona una facilidad de uso inigualable, escalabilidad, fiabilidad, y la extensión que los programadores profesionales y de empresas requieren para desarrollar, distribuir, depurar y administrar aplicaciones PHP críticas de negocios.

Zend Studio proporciona la visualización, edición y la capacidad de ejecución para bases de datos populares SQL incluyendo MySQL, Oracle, IBM DB2 y Cloudscape, Microsoft SQL Server, SQLite y PostgreSQL.

Principales Características (17)**:**

- IDE Líder para PHP.
- Desarrollo y despliegue de Código rápido.
- Control de calidad/ herramientas de prueba.
- Depurador local y remoto.
- Conectividad de base de datos y herramientas SQL.
- Entorno de equipo colaborativo.
- Soporte CVS y Subversión.
- Intercambio PHP4/PHP5 completo.
- Soporte PHPDOcs /PHPDocumentor.
- SFTP/FTP sobre SSL para conexiones seguras.
- 100+ plantillas de código.
- Integración de Zend Framework.
- Soporte de servicios web (SOAP).
- Plataforma de etapas.
- PHP/Java Bridge.
- API de Rendimiento.
- Integración con Zend Platform™.
- SFTP/FTP sobre SSL para conexiones seguras.

1.13 Conclusiones Parciales

Con todos los temas en este primer capítulo abordados y fundamentados, se concluye lo siguiente:

A lo largo de este capítulo se han expuesto las principales características y definiciones de la AS dentro de la Ingeniería de Software, así como su importancia para cualquier sistema que se desee implementar. Se abordaron las clasificaciones de los estilos y patrones tanto arquitectónicos como de diseño para el desarrollo de una AS específica y las Herramientas de Modelado y Desarrollo.

Capítulo 2

Tecnologías presentes en la solución

2.1 Introducción

Este capítulo es el resultado de la búsqueda y el análisis de la información correspondiente al proceso de diseño de la arquitectura del sistema. Definiendo los Estilos Arquitectónicos asociados y de Diseño, que se ajustan a la solución adecuada; también se definirá el marco de trabajo o framework que dará soporte a la aplicación, así como el lenguaje de programación y el gestor de Base de Datos, además se tendrá en cuenta las herramientas CASE automáticas y versión UML soportado.

2.2 Patrón de Arquitectura MVC

La base arquitectónica del sistema (SACGIR) se ha modelado haciendo uso del Patrón (MVC) con el objetivo de utilizar la separación de las responsabilidades de cada una de las capas que lo conforman y así lograr facilidades de desarrollo.

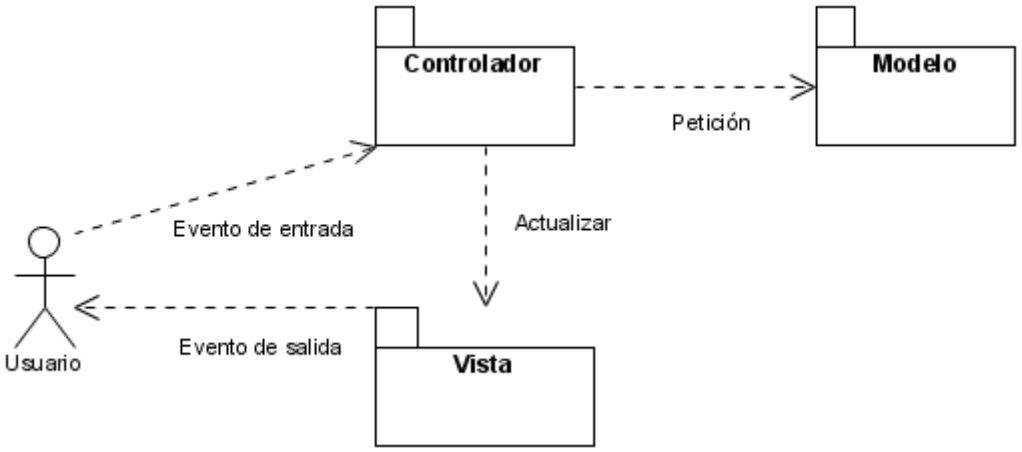


Figura #4 Flujo de mensajes del patrón MVC, empleado en SACGIR.

2.2.1 Modelo (Model): Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.

2.2.2 Vista (View): Intercambia la información con el usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

2.2.3 Controlador (Controller): Recibe las entradas, traducidas a solicitudes de servicio para el modelo.

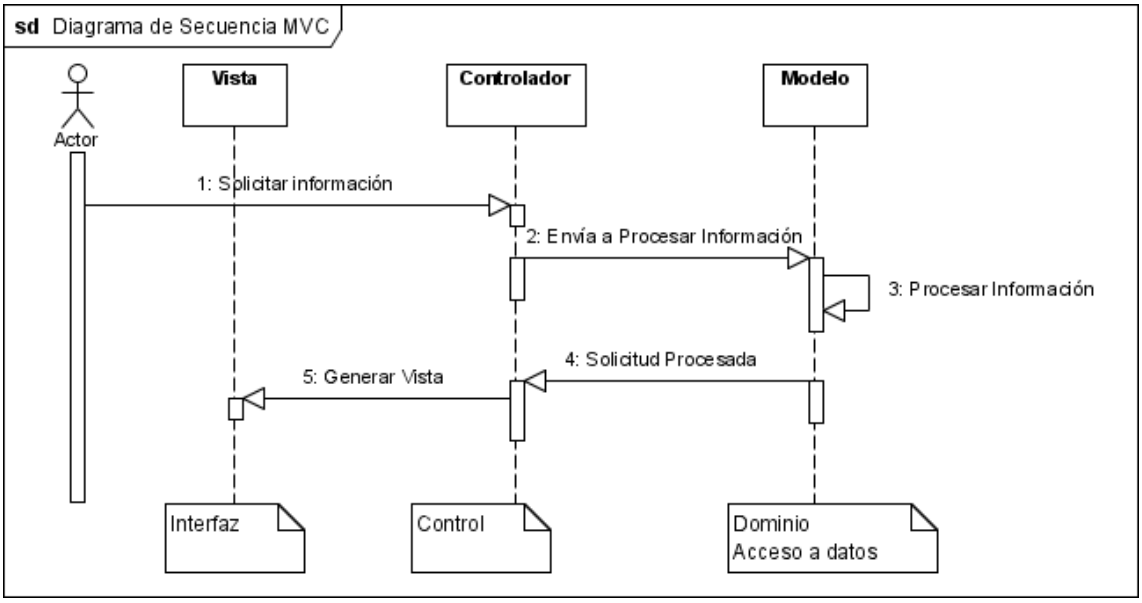


Figura #5 Diagrama de Secuencia del MVC.

2.2.4 MVC2

El modelo de patrón que se escogió para lograr la independencia de las Vista con el Modelo, ha sido el MVC2, el mismo elimina la eventualización del Modelo hacia las Vista, aqunque este aún tiene acceso a formularios que se encuentran en el Modelo. En el sitema la comunicación de las vista con el modelo se realiza solamente por medio de el Controlador, como se ilustra en la Figura # 4.

La arquitectura de MVC2 (18) realmente es una implementación del MVC modificada, el mayor cambio es que el Modelo ya no dispara los eventos a sus Vistas,Figura #5.

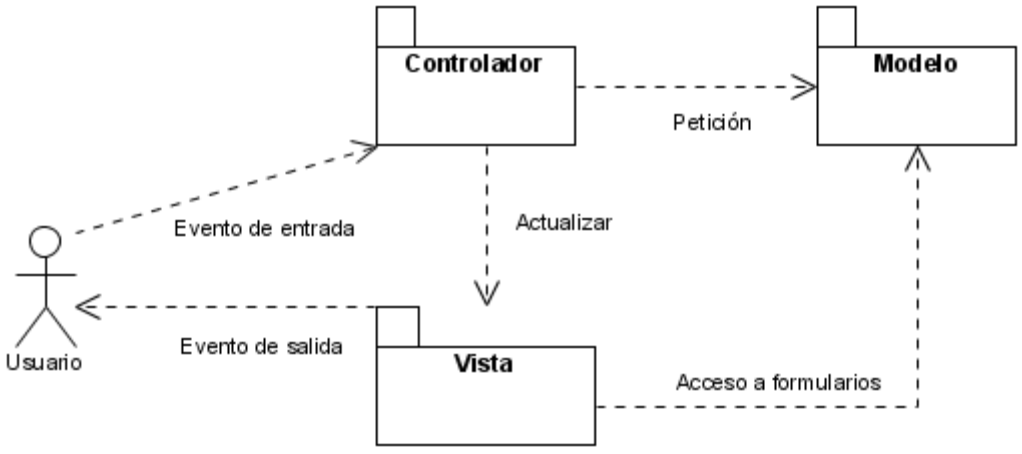


Figura #6 Flujo de mensajes del patrón MVC.

El MVC2 presenta responsabilidades análogas al MVC común, estas son:

El modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos, además de definir las reglas de negocio.

El controlador es responsable de:

- Recibir los eventos de entrada y comunicar las acciones que ellos derivan a la capa de Modelo.

Las vistas son responsables de:

- Recibir datos del modelo a través del Controlador para mostrar al usuario.

2.3 FrameWork CakePHP

CakePHP es un framework para programar aplicaciones Web que sigue la arquitectura MVC, propuesta en el proyecto. Para su funcionamiento requiere un servidor Web Apache, con PHP (versión 4 o 5) y un servidor de base de datos, que pude ser: MySQL, SQLite' o 'ADODB' y fundamentalmente PostgreSQL, por su licencia libre, el cual se utilizará en el proyecto, para la persistencias de los datos. Existen herramientas, como XAMPP, que integran ambos servicios en una única instalación. La versión de cakePHP con la que se ha trabajado es la 1.1.14.4797 de abril de 2007.

Seguidamente se representará la jerarquía de carpetas, al desempacar el framework; inicialmente existen tres principales carpetas: app, cake, docs, vendors y el fichero index.php. Dentro de la carpeta app es donde se desarrollara el sistema, las principales carpetas que aparecen dentro de app son:

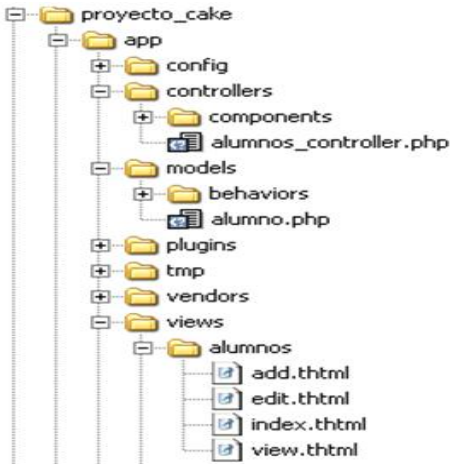


Figura #7 Carpetas y ficheros de un ejemplo de una aplicación.

Config: Contiene ficheros de configuración de la aplicación, un ejemplo es la configuración a la Base de Datos que el framework empleará.
Controllers: Contiene las clases controladoras de la aplicación y sus componentes.
Models: Contiene las clases modelos de la aplicación.
Views: Contiene los ficheros de la capa presentación o vista.

Existen otros ficheros y carpetas importantes para que la aplicación pueda funcionar, la cuales se pueden encontrar en la ayuda del framework (19) (20) .

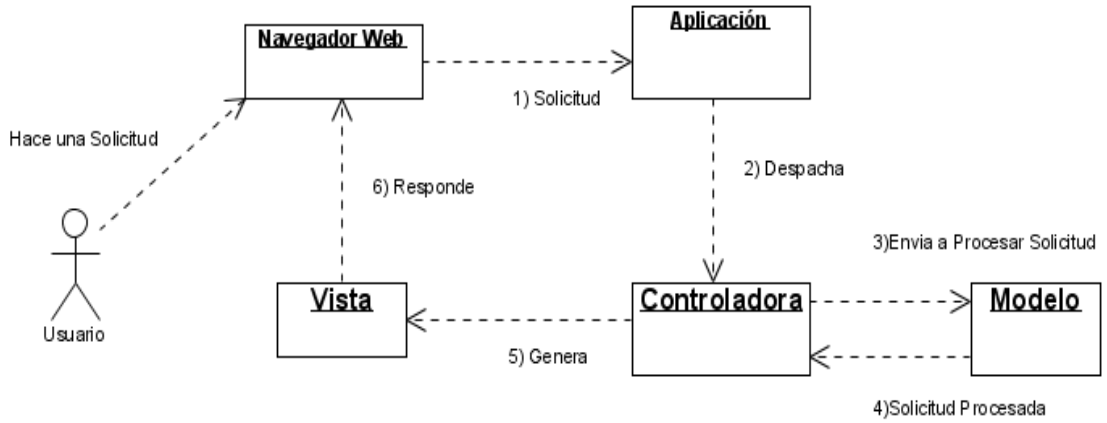


Figura #8 Estructura del paso de mensaje entre las capas vista, Modelo y Controlador.

2.4 Patrones de Diseño seleccionados

Los Patrones de Diseño son buenas soluciones que pueden ser aplicadas a problemas recurrentes que surgen durante el diseño de un sistema o aplicación. Para hacer sitios mejores en la Web, más funcionales y usables, es necesario romper el proceso de diseño web en pequeños pasos independientes basados en los detalles importantes dentro de los requerimientos, con el objetivo de tener una visión de cómo se pueden aplicar los Patrones en el sistema. Los patrones que se seleccionaron se tuvieron en cuenta en el marco de desarrollo o framework CakePHP.

2.4.1 Singleton (solitario)

Propósito
Garantizar que una clase sólo tiene una única instancia, proporcionando un punto de acceso global a la misma.

Estructura

Figura #9 Patrón de Diseño Solitario (Singlenton).

Cuando usarlo

- Cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes.
- Cuando la "InstanciaÚnica" debe ser especializable mediante herencia y los clientes deben poder usar la instancia extendida sin modificar su código (el de los clientes).

Ventajas

- El acceso a la "InstanciaÚnica" está más controlado.
- Se reduce el espacio de nombres (frente al uso de variables globales).
- Permite refinamientos en las operaciones y en la representación, mediante la especialización por herencia de "Solitario".
- Es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable).
- Es más flexible que la alternativa de las "operaciones de clase", además de que éstas (en C++), al ser funciones miembro estáticas, no son virtuales, luego no pueden ser especializadas mediante herencia ni redefinidas polimórficamente.

2.4.2 Command (acción)

Propósito: Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer la operaciones.

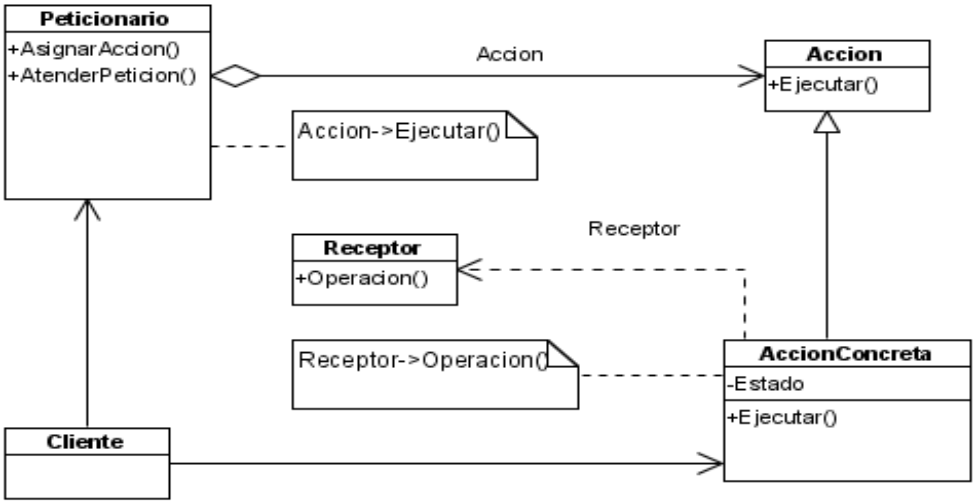


Figura #10 Patrón de Diseño Acción (Command).

Aspectos de interés

- El "Cliente" es el responsable de crear la "AcciónConcreta" correspondiente a una petición y de configurarla con el "Receptor" que sabe resolverla.
- "Acción" suele ser una clase abstracta que define la interfaz para la ejecución de operaciones.
- Para dar soporte a operaciones que se puedan deshacer y/o rehacer es preciso almacenar la información de estado que permita hacerlo y si además se pretende hacer esto en múltiples niveles habrá que mantener una "lista de históricos" con las acciones ya ejecutadas.
- Cada "AcciónConcreta" establece una relación entre el receptor de la petición y la acción que éste realiza para satisfacerla.
- Para permitir deshacer y/o rehacer operaciones, el "estado" debe contener información relativa a los receptores, las acciones y las operaciones involucradas en el proceso.
- Las habilidades de cada "AcciónConcreta" van desde la simple invocación de la correspondiente "Operación" del "Receptor" hasta la implementación interna de toda la acción que satisface la petición.

2.4.3 Front-Controller (Controlador Frontal)

Es recomendable utilizar este patrón debido a que obliga a todas las peticiones hechas a la aplicación pasen por un servlet Controlador.

- El controlador proporciona un punto de entrada único que controla y gestiona las peticiones Web realizadas por los clientes.
- Teniendo este único punto de entrada se evita tener que repetir la misma lógica de control en todos los .jsp.

Normalmente se utiliza junto con un Dispatcher[2] que es el responsable de redirigir el flujo de ejecución hacia el .jsp adecuado. Este Dispatcher puede ser realizado por el propio controlador o estar en una clase a parte.

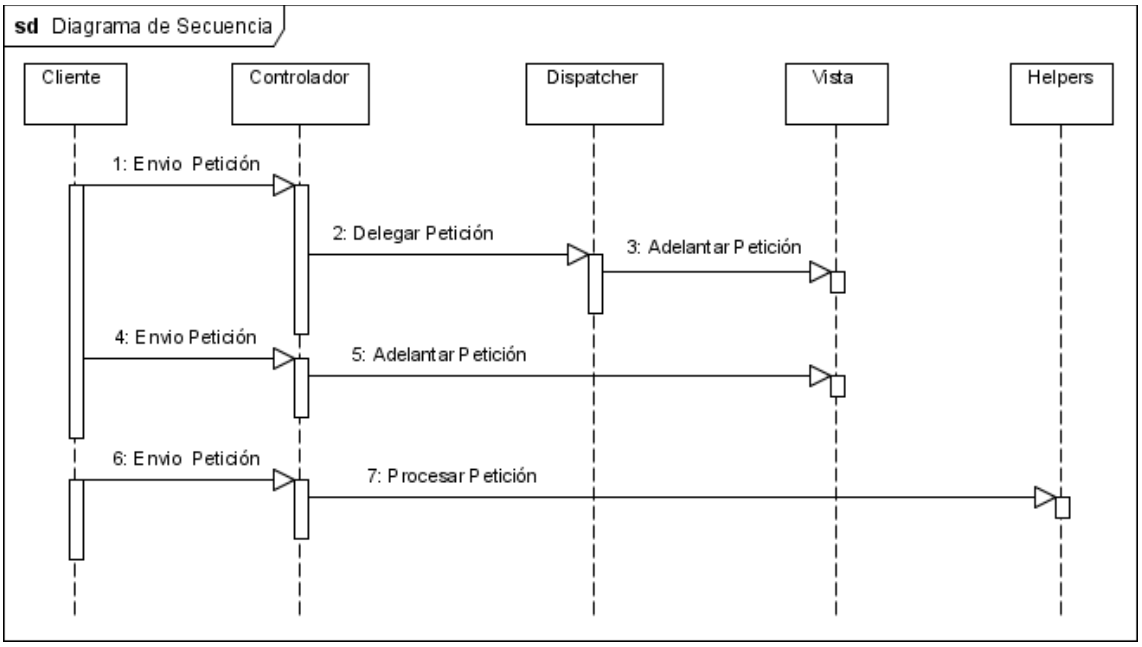


Figura #11 Diagrama de secuencia del patrón Front Controller.

Aspectos de interés

El Patrón de Diseño Front Controller funciona como fichero de punto de entrada único a la aplicación.

2.4.4 Experto

La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada.

[2] La aplicación delega a la capa controladora la petición del usuario desde el navegador Web.

Hay que tener en cuenta que es aplicable mientras se consideren los mismos aspectos del sistema:

- Lógica de negocio
- Persistencia a la base de datos
- Interfaz de usuario

2.4.5 Controller (Controlador)

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. Un error muy común es asignarle demasiada responsabilidad y alto nivel de acoplamiento con el resto de los componentes del sistema.

2.4.6 Active Record(Registro Activo)

Un objeto que hace de wrapper a una fila de una tabla o vista de la base de datos encapsulando el acceso a los datos y agregando la lógica del dominio a sus datos.

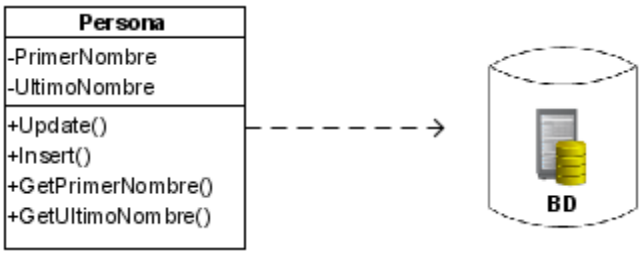


Figura # 12 Representación del Patrón Active Record

Un objeto que posee tanto datos como comportamiento y muchos de sus datos son persistentes y necesitan ser almacenados en una base de datos. Active Record pone la lógica de acceso a datos en el objeto de dominio, permitiendo que cada objeto de dominio sepa como cargarse y guardarse desde y hacia la base de datos.

2.4.7 Registry (Registro)

Un objeto conocido que otros objetos pueden utilizar para encontrar objetos y servicios comunes.

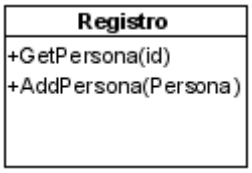


Figura #13 Representación del patrón Registry.

Registry es en esencia un objeto global o al menos parece uno, (singleton) el cual es accedido directamente por otros objetos.

2.4.8 Fabricación Pura

Solución

Asignar un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema: inventada para dar soporte a una alta cohesión, un bajo acoplamiento y reutilización.

Esa clase es una fabricación de la imaginación. En teoría, las responsabilidades que se asignan brindan soporte a una alta cohesión y a bajo acoplamiento, de modo que el diseño de la fabricación sea muy limpio, o puro. De ahí el nombre: fabricación pura.

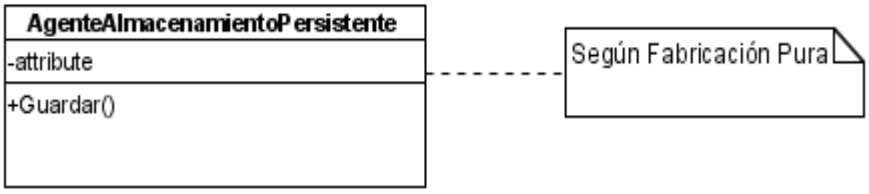


Figura #14 Representación del patrón Fabricación Pura.

2.4.9 Caché

El patrón Administración de Caché (21) propone asignar a los intermediarios de bases de datos la responsabilidad de dar mantenimiento a su caché. Si se utiliza un intermediario diferente con cada clase de objeto persistente, el intermediario habrá de darle mantenimiento a su propia caché.

Cuando se materializan los objetos, se colocan en la caché con su identificador como clave. Si después se le pide al intermediario un objeto, primero buscará la caché y con ello evitará una materialización innecesaria.

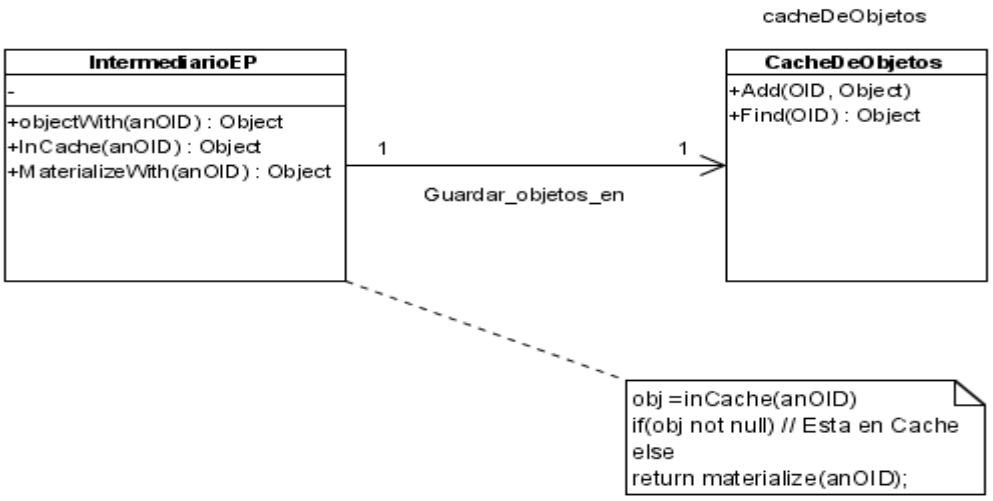


Figura # 15 Los intermediarios conservan una caché de los objetos materializados.

2.6 Patrones de Idiomas

Estos se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como estándares de codificación y proyecto, describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y predictibilidad.

2.6.1 Patrones de idiomas seleccionados

- **Mayúsculas y Minúsculas Pascal**: La primera letra del identificador y la primera letra de las siguientes palabras concatenadas están en mayúsculas.
- **Instrucciones de Nomenclatura de Clases**: Utilizar un sustantivo o un sintagma nominal para asignar un nombre a una clase, el estilo de Mayúsculas y minúsculas Pascal y las abreviaturas con moderación.
- **Instrucciones de Nomenclatura de Interfaces**: Asignar nombres a interfaces utilizando sustantivos, sintagmas nominales o adjetivos que describan su comportamiento. Utilizar el estilo de Mayúsculas y minúsculas Pascal y las abreviaturas con moderación. Incluir un prefijo con la letra I en los nombres de interfaces para indicar que el tipo es una interfaz.
- **Instrucciones de nomenclatura de tipos de enumeración**: Utilizar el estilo de Mayúsculas y minúsculas Pascal en los nombres de valores y tipos Enum, las abreviaturas con moderación, no utilice el sufijo Enum en nombres de tipo

Enum, Utilice un nombre en singular para la mayoría de los tipos Enum; pero utilice un nombre en plural para los tipos Enum que son campos de bits, agregue siempre FlagsAttribute a un tipo Enum de campo de bits.

- **Instrucciones de nomenclatura de parámetros**: Utilizar el estilo de Mayúsculas y minúsculas Camel para los nombres de parámetros. también utilizar nombres de parámetros descriptivos, los nombres de parámetros deben ser lo suficientemente descriptivos como para que el nombre y el tipo del parámetro se puedan utilizar para determinar su significado en la mayoría de los escenarios.
- **Instrucciones de nomenclatura de métodos**: Utilizar verbos o sintagmas verbales al asignar nombres a los métodos, y el estilo de Mayúsculas y minúsculas Pascal.
- **Instrucciones de nomenclatura de campos estáticos**: Utilizar sustantivos, sintagmas nominales o abreviaturas de nombres al asignar nombres a campos estáticos, el estilo de Mayúsculas y minúsculas Pascal, no debe utilizarse un prefijo de notación húngara en nombres de campos estáticos. Se recomienda utilizar propiedades estáticas en lugar de campos estáticos públicos cada vez que sea posible.

2.7 Lenguaje de Modelado UML 2.x

El Lenguaje Modelado Unificado (UML) es una norma ampliamente usada en la industria del software para el software modelado. Ayuda a los practicantes a visualizar, comunica, y lleva a cabo sus planes. UML ha evolucionado durante los años y está ahora en su versión 2.x.

La versión de UML para la herramienta de modelado Visual Paradigm se ha sincronizado con el nuevo desarrollo de UML 2.1 para proporcionar un ambiente modelado visual que satisface la tecnología del software de hoy y necesidades de comunicación.

2.7.1 Ventajas que ofrece UML 2.1 para Visual Paradigm

UML 2.1 ofrece herramientas de modelado fácil de usar y fiable, en la cual se pueden citar las siguientes:

- Diagrama de Clases.
- Diagrama de Secuencia.
- Diagrama de Estados.
- Diagrama Componente.
- Diagrama de Paquetes.
- Diagramas de estructura compuestos.
- Disgrama de Interaccion.
- Diagrama de flujo de eventos.
- Diagrama de Casos de Uso.
- Diagrama de Comunicación.
- Diagrama de Secuencia.
- Diagrama de Objetos.
- Diagrama de Despliegue.
- Diagramas de Actividades.
- Modelo Caso de Uso de Negocio.
- Editor de Casos de Uso.
- Diagrama de Estimación de Tiempo.

2.8 Herramienta CASE Seleccionada

2.8.1 Visual Paradigm

Visual Paradigm es una poderosa herramienta CASE que al igual que el Rational Rose utiliza UML para el modelado, es la herramienta por excelencia para ser utilizada en un ambiente de software libre. Permite crear tipos diferentes de diagramas en un ambiente totalmente visual. Es muy sencillo de usar, fácil de instalar y actualizar. Genera código para varios lenguajes. Tiene integrado el MS Visio y es compatible con otras ediciones, posibilita un entorno de creación de diagramas para UML 2.x. Posibilita la representación gráfica de los diagramas permitiendo ver el sistema desde diferentes perspectivas, como el de componentes, despliegue, secuencia, casos de uso, clase, actividad, estado, entre otros.

Además, identifica requisitos y comunica información, se centra en cómo los componentes del sistema interactúan entre ellos, además, permite ver las relaciones entre los componentes del diseño y mejora la comunicación entre los miembros del equipo usando un lenguaje gráfico.

En el proyecto la versión a utilizar es la Enterprise Edition, debido a las características que brinda:

- Modelamiento Proceso de Negocio.
- Modelamiento de Base de Dato.
- Modelo Relacional de Objetos.
- Modelamiento Visual.
- Integración a Entornos de Desarrollo.
- Ingeniería Directa e Inversa.
- Generación de Código.
- Inter Operabilidad.

2.9 Lenguaje de Programación seleccionado

El lenguaje de programación seleccionado para el desarrollo de la aplicación es PHP5, al ser un lenguaje libre dispone de una gran cantidad de características que lo convierte en la herramienta ideal para la creación de páginas Web dinámicas.

2.9.1 Características que proporciona la utilización del lenguaje

- Es un lenguaje multiplataforma.
- Capacidad de conexión con la mayoría de los manejadores de base de datos que se utilizan en la actualidad (Ejemplo PostgreSQL, es el que se utiliza en el proyecto, se mencionará más adelante).
- Capacidad de expandir su potencial utilizando la enorme cantidad de módulos llamados ext's o extensiones.
- Es libre, se presenta como una alternativa de fácil acceso para todos.
- Posee una amplia documentación de ayuda para un mayor entendimiento del lenguaje.
- Soporte sólido para Programación Orientada a Objetos con PHP Data Objects.
- Está acompañado de una excelente biblioteca de funciones como son: acceso a base de datos, encriptación, envío de correo, creación de PDF, XML.
- PHP 5 incorpora la extensión *php_soap* para el manejo de protocolos SOAP (Protocolo de Acceso Simple a Objetos, protocolo popular para la creación y consumos de servicios web).

2.10 Gestor de Base de Datos seleccionado

Como Gestor de Bases de Datos a utilizar para el desarrollo de la base de datos se seleccionó el PostgreSQL, su principal característica es que es código abierto, además de permitir soporte en el lenguaje de programación seleccionado (PHP).

2.10.1 Principales Ventajas

- Soporta transacciones y desde la versión 7.0, llaves foráneas (integridad referencial).
- Escala muy bien al aumentar el número de CPUs y la cantidad de RAM.
- Utiliza un Control de Concurrencia Multi-Versión (Multi-Version Concurrency Control), mediante el cual evita el problema de los bloqueos por completo.
- Permite copias de seguridad en caliente desde *pg_dump* mientras la base de datos permanece disponible para consultas.
- Los bloqueos de tabla han sido sustituidos por el control de concurrencia multiversión, el cual permite a los accesos de sólo lectura continuar leyendo datos consistentes durante la actualización de registros.
- Cuenta con un mejor soporte para *subselects, triggers, vistas y procedimientos almacenados* en el servidor.
- Soporte multiusuario, transacciones, optimización de consultas, herencia y valores no atómicos (atributos basados en vectores y conjuntos).

2.11 Herramienta IDE seleccionada

Según la propuesta del lenguaje de programación anteriormente expuesta (PHP 5), se seleccionó el IDE Eclipse, una de las herramientas de Desarrollo Integradas utilizadas por la comunidad de desarrolladores web. La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plug-ins que van conformando la funcionalidad final. La forma en que los plug-ins interactúan es mediante interfaces o puntos de extensión; así, las nuevas aportaciones se integran sin dificultad ni conflictos.

Para la utilización de este IDE se le incluye el plug-in PDT (Developer PHP Tools) version1.0, para permitir ejecutar el programa o el sistema (en la implementación) sobre la plataforma (Linux, Windows, etc.) utilizada.

PDT está dirigido hacia la plataforma de integración del IDE Eclipse, suministra componentes necesarios para el desarrollo de aplicaciones web con el lenguaje de programación PHP, multiplataforma y es el resultado de la unión de Zend y la Fundación Eclipse.

2.11.1 Herramientas del PDT

- **Editor**: brinda un resaltado en la sintaxis y completamiento de código.
- **Inspection**: Contiene un explorador con los ficheros o el proyecto como tal desarrollado en PHP.
- **Debug**: Permite debugear los script de PHP.

2.11.1.1 Características en la versión 1.0 del PDT

1. Editor sensible al contexto, el cual provee de resaltamiento de código, asistente de código y autocompletado de código.
2. Integración con el modelo del proyecto Eclipse, que permite para inspeccionar el uso de las vistas del contorno del fichero y del proyecto, así como la nueva vista PHP Explorer.
3. Soporte para el debug incremental del código de PHP.
4. Extensos framework y APIs que permiten a los desarrolladores e ISVs (vendedores de software independientes) fácilmente extender PDT para crear nuevas e interesantes herramientas orientadas al desarrollo de PHP.

2.12 Conclusiones Parciales

En este capítulo se han definido, luego de la búsqueda y análisis de la información del estado del arte de todos los aspectos mencionados en el primer Capítulo, las herramientas que se ajustan a la solución del problema.

Capítulo 3

Línea Base de la Arquitectura

3.1 Introducción

Este Capítulo tiene como propósito lograr una mejor comprensión del sistema, organizar el desarrollo, fomentar la reutilización, contribuyendo de esta forma a una evolución del sistema más rápida y eficaz. A través del Documento de Descripción de la Arquitectura se describe la solución arquitectónica del sistema, incluyendo además otros artefactos como son las vistas arquitectónicas, definidas por la metodología RUP.

3.2 Estructura del Equipo de desarrollo

3.2.1 Herramientas de Desarrollo

A modo de resumen se mencionarán las herramientas tanto de modelación como de desarrollo que se definieron en el proyecto SACGIR, de esta forma se conformaron los puestos de trabajo por roles.

Herramientas de Modelado.

- Lenguaje de Modelado: UML 2.1
- Herramientas CASE: Visual Paradimg, versión: Enterprise Edition

Herramientas de Desarrollo.

- Lenguaje de Programación PHP5 y framework CAKEPHP.
- Gestor de Base de Datos: PostgreSQL.
- IDE Eclipse incluido el plug-in PDT.

3.2.2 Estructura del Equipo de Desarrollo

El equipo de trabajo está distribuido de la siguiente forma[3]:

- Líder Proyecto.
- Líder de Desarrollo.
- Arquitecto Principal.
- Arquitecto de Información.
- Analista Principal.
- Implementador.
- Integrador.
- Planificador.
- Diseñador de Base de Datos.
- Diseñador de Interfaz de Usuario.
- Analista de Riesgo.
- Calidad.
- Revisor Técnico.
- Económico.
- Administrador de Configuración y Control de Cambios.

3.2.3 Configuración de los puestos de trabajos por roles

- **Analista**
  1. PC, con mause y teclado.
  2. Instalación de Visual Paradigm.
  3. Instalación del paquete Oficce/ Open Oficce.

- **Implementador**
  1. PC, con mause y teclado.
  2. Instalación del IDE Eclipse, Servidor Apache y framework CakePHP.

- **Documentador**
  1. PC, con mause y teclado.
  2. Instalación de Visual Paradigm.
  3. Instalación del paquete Oficce/ Open Oficce.

- **Diseñador BD**
  1. PC, con mause y teclado.
  2. Instalación de Visual Paradigm.
  3. instalación de postgreSQL

[3] Consultar Anexo No. 1 Organigrama del Equipo de Desarrollo.

3.3 Organigrama de la Arquitectura

3.3.1 Visión General de la Arquitectura

El siguiente diagrama representa la estructura del sistema SACGIR:

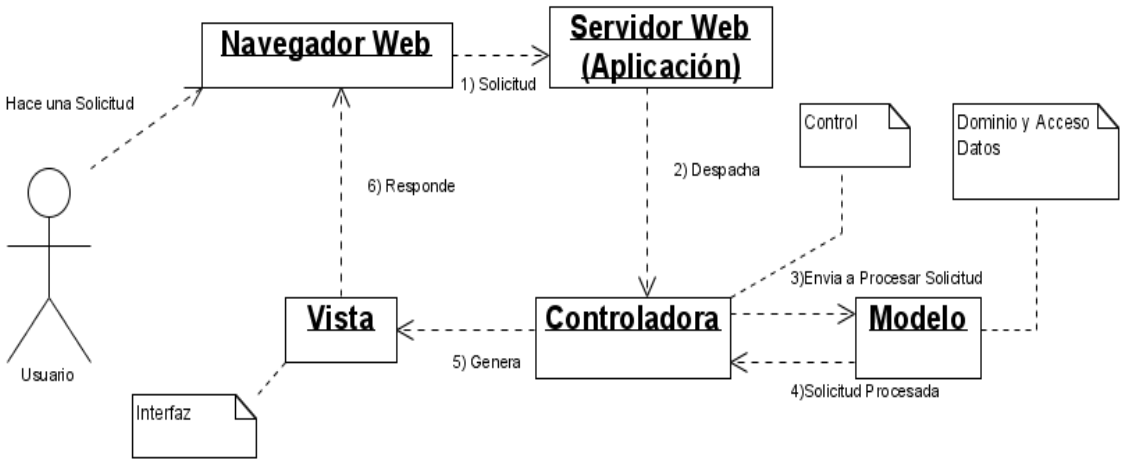


Figura #16: Estructura de capas del patrón MVC aplicado al Sistema.

Descripción de responsabilidades de las áreas propuestas:

Área Vista:

Interfaz: *(El usuario interactúa con el sistema a través de la interfaz)*

- Control de los roles de usuario para la aceptación de peticiones.
- Captura de datos para la conformación de solicitudes al servidor.
- Reajuste basado en los datos que llegan desde la capa de Control.

Área Controlador:

Control: *(la capa controladora recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario)*

- Control de los roles de usuario para la ejecución de peticiones.
- Recepciona peticiones realizadas por la capa Interfaz y las entrega a la capa de Negocio.

Área Modelo:

Dominio: *(contiene la lógica y las reglas de Negocio)*

- Control de los roles de usuario para en contraste con las reglas del negocio.
- Ejecuta las peticiones realizadas por la capa de Control.
- Realiza peticiones a la capa de acceso a datos.

Acceso a Datos: *(incorpora los procedimientos necesarios para el acceso y control de los datos)*

- Control de los roles de usuario para la consulta de datos del sistema.
- Almacenamiento de la información del sistema.

En la siguiente tabla se ilustra las Tecnologias que se emplearan en cada Área:

Área	PHP	CakePHP	DHTML	Java script	PostgresSQL
Interfaz	X	X	X	X	
Control	X	X			
Dominio Acceso a Datos	X	X			X

Tabla # 2: Tecnologías utilizadas por capa.

3.3.2 Módulos del sistema

- ✓ Módulo Administración
- ✓ Módulo de Gestión
- ✓ Módulo de Reportes
- ✓ Módulo de Control
- ✓ Módulo de Planificación

Responsabilidades de cada módulo:

- ✓ Módulo Administración :
  - Es el encargado de la gestión de nomencladores. Ejmp. Lista Refinerías, Lista de crudos. etc
  - Obtiene e inserta información al sistema.
  - Control de Roles de usuarios (autenticación según el nivel de privilegios).
- ✓ Módulo de Gestión :
  - Gestiona toda la información que entra, se procesa, modifica, elimina y sale del sistema.
  - Obtiene e inserta información al sistema.
- ✓ Módulo de Reportes :
  - Muestra la información que se genera de estos reportes.
  - Solamente obtiene información, no tiene permisos para modificarla.
- ✓ Módulo de Control :
  - Evaluar el comportamiento de los indicadores.
  - Obtiene e inserta información al sistema.

- ✓ Módulo de Planificación
  - Es el encargado de gestionar la planificación en el área de refinación.
  - Obtiene e inserta información al sistema.

3.3.3 Restricciones de acuerdo a la estrategia de diseño

1. El diseño de la aplicación, se hará aplicando el paradigma de la programación orientada a objetos.
2. Se utilizarán las tecnologías que brinda el framework definido(CakePHP).

3.4 Descripción de la Arquitectura

Para lograr una mejor comprensión del sistema, organizar el desarrollo, fomentar la reutilización y una comprensión arquitectónica global del sistema, utilizando las vistas arquitectónicas definidas por la metodología RUP.

- Vista de casos de uso.
- Vista lógica.
- Vista de implementación.
- Vista de despliegue

3.4.1 Vista de Caso de Uso

El sistema cuenta con 48 Casos de Uso, todos arquitectónicamente significativos[4]; los cuales comenzarán a desarrollarse en la primera iteración del proyecto. Están representados por módulos de la siguiente forma:

Módulo	Cantidad de Casos de Uso
Gestión	27
Control	2
Reportes	11
Administración	8

Tabla #3: Cantidad de Casos de Uso del sistema por módulos.

[4] En la documentación digital se encontran los modelos “Casos de Uso”.

3.4.2 Vista Lógica

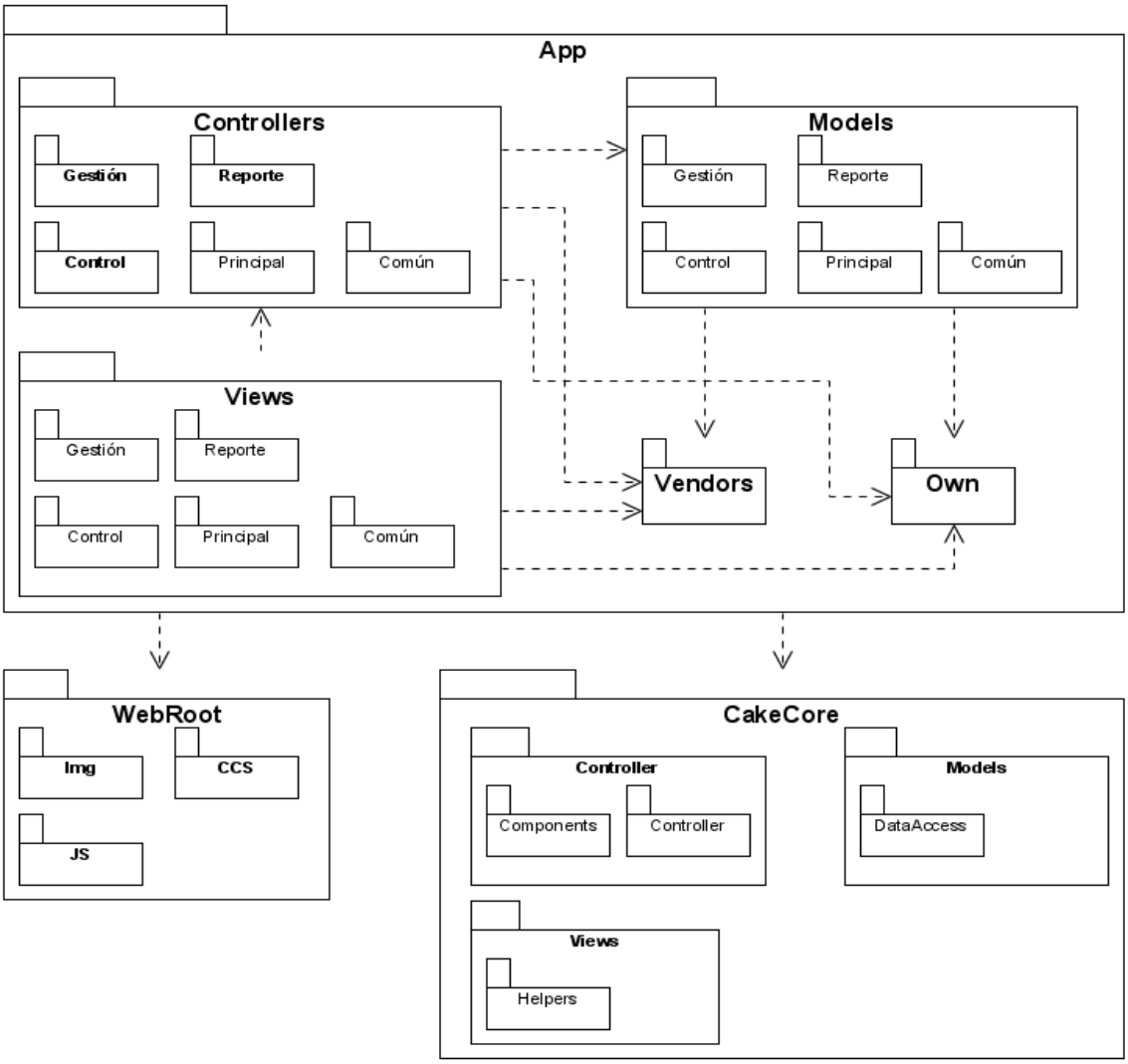


Figura #17: Vista Lógica.

Paquete	Descripción
App	El paquete contiene las clases del negocio.
Views	El paquete contiene las clases Interfaz (vistas) del sistema separadas en los sub paquetes “Gestión”, “Control”, “Reporte”, “Principal”, “Común”.
Controllers	El paquete contiene las clases controladoras del sistema separadas en los sub paquetes “Gestión”, “Control”, “Reporte”, “Principal”, “Común”.
Models	El paquete contiene las clases Modelo del sistema, separadas en los sub paquetes “Gestión”, “Control”, “Reporte”, “Principal”, “Común”. Se incluye un conjunto de clases para la representación de reglas y lógica del negocio.

Vendors	El paquete contiene las clases (definidas por el framework). de apoyo al sistema que no forman parte de la lógica del negocio
Own	El paquete contiene otras clases (definidas por el equipo de desarrollo) de apoyo al sistema que no forman parte de la lógica del negocio.
WebRoot	El paquete contiene los archivos de apoyo necesarios para el diseño gráfico de la interfaz de la aplicación web.
Img	El paquete contiene todas las imágenes que se utilizan en las interfaces del sistema.
CSS	El paquete contiene todos los estilos que se aplican en las interfaces del sistema.
JS	El paquete contiene las clases desarrolladas en Java script que se utilizan en las interfaces del sistema.

Tabla #4.1: Descripción de los paquetes del sistema.

CakeCore	El paquete contiene las clases base del framework CakePHP.
Views	El paquete contiene las clases base que ofrece el framework para el trabajo con la capa de interfaz del patrón Modelo-Vista-Controlador. Se incluyen las clases “Helpers” (facilitadores de diseño web)
Controllers	El paquete contiene las clases base que ofrece el framework para el trabajo con la capa de Control del patrón Modelo-Vista-Controlador. Se incluye la clase Controller y otras Componentes (clases) de apoyo al proceso de control.
Models	El paquete contiene las clases base que ofrece el framework para el trabajo con la capa de Acceso a Datos del patrón Modelo-Vista-Controlador. Se incluyen las clases de acceso a datos de PostgresSQL y clases con funcionalidades de acceso a datos característicos y relevantes del framework CakePHP

Tabla #4.2: (continuación)

Diagrama de Clases del sistema.

Las clases del sistema, desarrolladas todas en el lenguaje php5, se encuentran distribuidas a una por archivo. Localizadas atendiendo con su responsabilidad en el modelo de paquetes, bajo una estructura de carpetas del sistema de archivos similar al diagrama. La descripción de las clases se realiza a través del Modelo de Diseño del Sistema, separado en un documento por cada módulo[5].

3.4.3 Vista de Procesos

El sistema se basa en la arquitectura cliente-servidor sobre la plataforma Web, donde cada instancia del sistema en el cliente es independiente de la ejecución de otra. La concurrencia de utilización del servidor Web y la utilización de la Base de Datos se maneja a través de los propios servidores.

El sistema no cuenta con tareas que requieran ejecutarse periódicamente sin la intervención del cliente. Por lo anterior no se requiere una Vista de Procesos.

3.3.5 Vista de despliegue

Describe los nodos físicos necesarios para las configuraciones de la plataforma donde se ejecutará el sistema y la asignación de las tareas de la Vista del Proceso a los nodos físicos. Esta vista se realiza sólo si el sistema es distribuido a través de más de un nodo, por lo tanto es opcional.

3.3.5.1 Diagrama de Despliegue

Un diagrama de despliegue muestra las relaciones físicas entre los componentes hardware y software en el sistema. Es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos.

[5] En la documentación digital se encuentra el “Modelo de Diseño del sistema”.

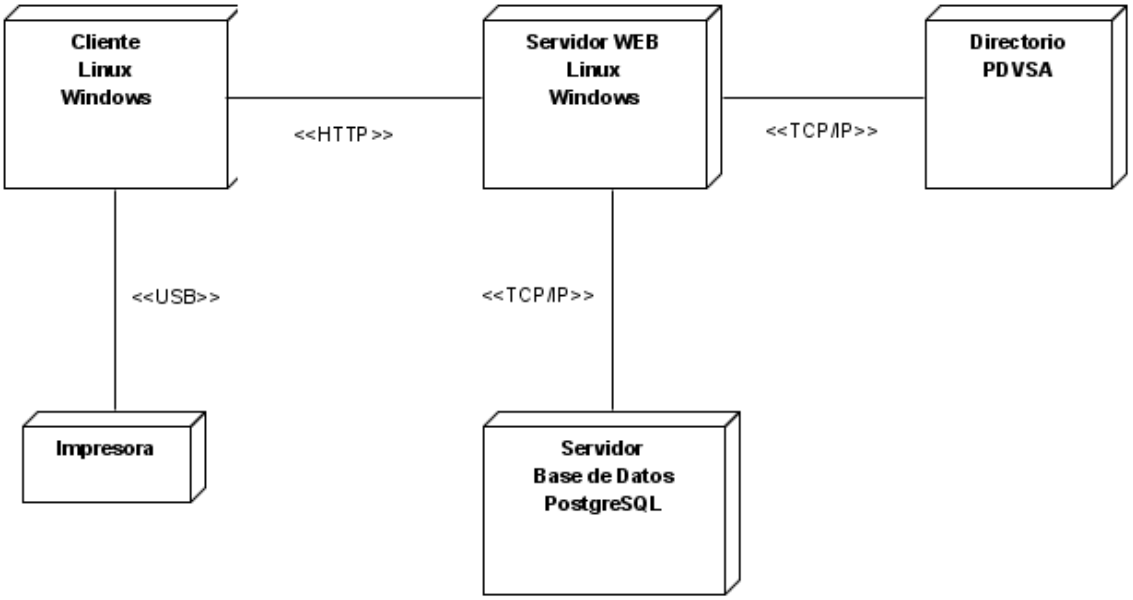


Figura # 18: Diagrama de despliegue

A continuación se describirá los nodos físicos representados en el diagrama de despliegue:

Servidor Web

Figura #18.1 Servidor Web.

Subsistemas de Implementación.

En este nodo se ejecutarán todas las funcionalidades del servidor Web, entre ellas se encuentra la construcción de interfaces de usuarios, el procesamiento de datos, y el control de flujo.

- Capa Presentación.
- Capa de Control.
- Capa de Modelo.

Servidor de Bases de Datos

Figura #18.2 Servidor de Bases de Datos.

Subsistemas de Implementación.

En este nodo estará ejecutándose el servidor PostgreSQL. La lógica del tratamiento de los datos no se implementará aquí sino en el servidor Web en la misma aplicación.

- Capa de Datos

Cliente Web

Figura #18.3 Clientes Web.

Subsistemas de Implementación.

- Representación de la Capa Presentación en el cliente a través de Javascript (AJAX).

Directorio PDVSA

Figura #18.4 Clientes Web.

Subsistemas de Implementación.

Este nodo es el servidor de Dominio, el cual contendrá todos los usuarios de PDVSA.

Descripción de los elementos e interfaces de comunicación

En el caso del proyecto, el cliente ya tiene establecido la configuración de las redes de datos en la empresa. Por lo que el sistema solamente haría uso de las mismas para su ejecución.

Protocolos TCP/IP y HTTP

Se utilizarán los protocolos de comunicación mencionados debido a que el sistema se desarrolla sobre tecnología Web siguiendo los estándares para la misma.

Cliente/Servidor

Como servidor Web se utiliza Apache 2.x, por sus características de Software Libre y compatibilidad con el lenguaje de desarrollo utilizado, PHP.

El cliente podrá ser Netscape 3 (o superior), Internet Explorer 4.2 (o superior) y compatibles; ejecutándose tanto en Sistema Operativo Windows como Linux.

3.3.6 Vista de Implementación

El diagrama apunta principalmente a la distribución de los componentes comunes que integran el sistema.

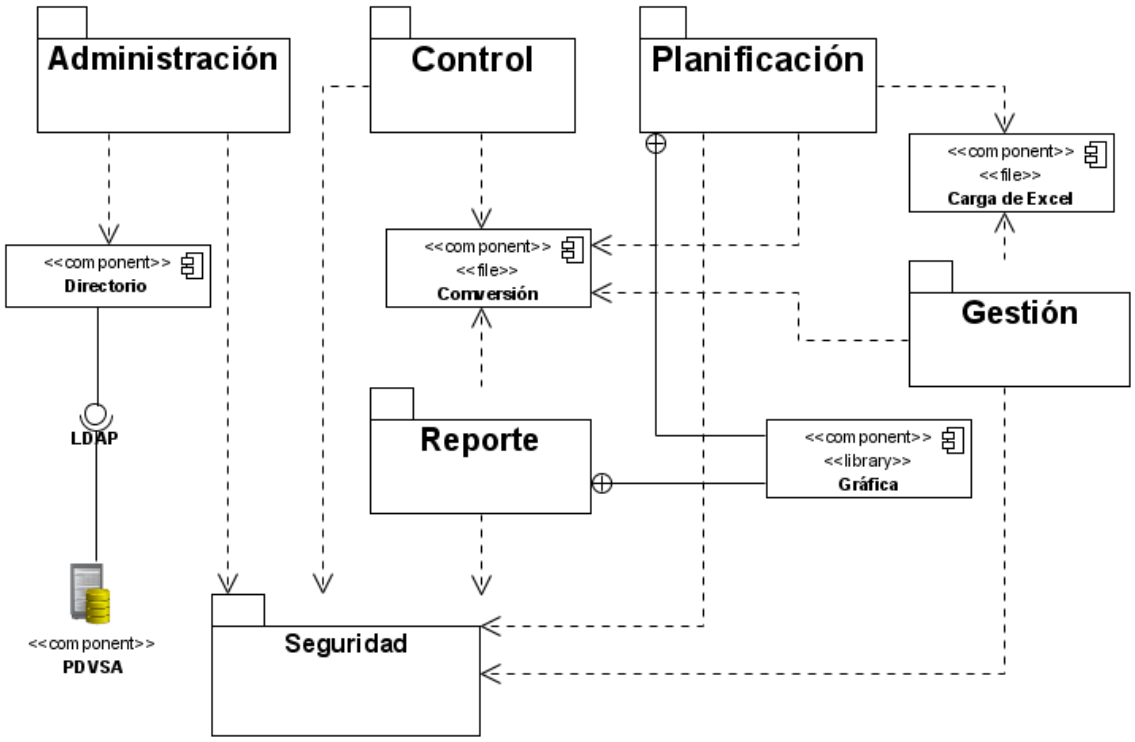


Figura # 19 Vista de Implementación.

3.3.7 Vista de Datos

El sistema cuenta con 53 tablas[6] que permiten la persistencia de datos para los módulos de forma común. Todas las tablas son arquitectónicamente significativas para el sistema.

Como Gestor de Bases de Datos se utiliza PostgreSQL. Es Software Libre, con funcionalidades para el tratamiento y seguridad de grandes volúmenes de datos.

La utilización del Framework de desarrollo CakePHP ofrece facilidades en el manejo de los datos, aunque restringe la utilización de Vistas, y Triggers declarados en la Base de Datos.

3.4 Requerimientos No funcionales

3.4.1 Usabilidad

Preparar a los Administradores en la gestión de Roles y Permisos.		
	Descripción	Esta acción específica del sistema llega un alto grado de complejidad, se debe preparar un curso para instruir a los que trabajaran con estas funcionalidades.
	Prioridad	Secundario
	Estimado	Fin de la segunda versión

Tabla # 5.1 Requerimiento No Funcional: Usabilidad.

Preparar a los Administradores en la gestión de los indicadores		
	Descripción	Esta acción específica del sistema llega un alto grado de complejidad, se debe preparar un curso para instruir a los que trabajaran con estas funcionalidades. En el caso específico del trabajo con la definición de la fórmula del indicador, aunque se tratará de que la interfaz sea lo más amigable posible, es necesario mediante un curso adiestrar a este a rol a que pueda definir cualquier indicador mediante la definición de fórmulas.
	Prioridad	Secundario
	Estimado	Fin de la segunda versión

Tabla # 5.2 (Continuación).

[6] La descripción detallada de la Vista de Datos se puede encontrar en el documento “Descripción del Modelo Entidad Relación”

3.4.2 Rendimiento

El sistema debe responder en un tiempo relativamente rápido a las peticiones del usuario (menos de 5 segundos)		
	Descripción	Teniendo en cuenta que el sistema no es de tiempo real y no necesita de respuestas de milisegundos se plantea este como un tiempo máximo promedio estimado, el cual debe ser mucho menor; aunque si se alcanza valor no afectará el funcionamiento del sistema.
	Prioridad	Alta
	Estimado	Durante el ciclo de desarrollo, a prueba después del despliegue del sistema definitivo.

Tabla # 6.1 Requerimiento No Funcional: Rendimiento

El sistema necesita en los clientes con 256 mb de RAM y un navegador WEB que soporte javascript		
	Descripción	Hay funcionalidades que están desarrolladas en javascript por lo que es imprescindible que los clientes lo soporten.
	Prioridad	Alta
	Estimado	Durante el ciclo de desarrollo, probado durante las actividades de despliegue de la 1era versión.

Tabla # 6.2 Requerimiento No Funcional: Rendimiento

El sistema debe permitir conectados trabajar concurrentemente 200 usuarios como mínimo.		
	Descripción	La cantidad de usuarios que se encuentran en PDVSA que usarían el sistema no deben parar esa cifra.
	Prioridad	Alta
	Estimado	Durante el ciclo de desarrollo, probado durante las actividades de despliegue de la 1era versión.

Tabla # 6.3 Requerimiento No Funcional: Rendimiento

El sistema necesita un servidor de base de datos de 8 gigas de RAM		
	Descripción	Para ganar en velocidad a la hora del trabajo con datos
	Prioridad	Alta
	Estimado	Durante el despliegue.

Tabla # 6.4 Requerimiento No Funcional: Rendimiento

3.4.3 Soporte

Es necesario de que los documentos Excel se elaboren en un mismo formato estándar.		
	Descripción	De los documentos Excel se importa gran parte de la información de los planes y precios y costos, por lo que es necesario que mantengan un formato estándar.
	Prioridad	Alto
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 7 Requerimiento No Funcional: Soporte.

3.4.4 Restricciones de diseño

El sistema debe guiarse por las pautas de diseño de la Intranet de PDVSA.		
	Descripción	El sistema va a ser accedido desde la intranet, además que como política de PDVSA todos los sistemas deben estar integrados a la intranet.
	Prioridad	Secundario
	Estimado	Durante el ciclo de desarrollo.

Tabla # 8 Requerimiento No Funcional: Restricciones de diseño.

3.4.5 Adquisición de Componentes

El sistema debe de conectarse a una Base de Datos externa.		
	Descripción	El sistema necesita de una conexión con la base de datos del SIM en una primera versión. El sistema necesita de una conexión con la base de datos del SIMP en una segunda versión. El sistema necesita de una conexión con la base de datos del STARS en la segunda versión.
	Prioridad	Alta
	Estimado	Durante el ciclo de desarrollo.

Tabla # 9 Requerimiento No Funcional: Componentes

3.4.6 Interfaz

3.4.6.1 Interfaces de Usuarios

Interfaz de interacción con el usuario.		
	Descripción	Interfaces amigables, fáciles de interactuar con ellas.
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo.

Tabla # 10.1 Requerimiento No Funcional: Interfaz Usuario.

3.4.6.2. Interfaces con otros Software

Interfaz de interacción con otros sistemas.		
	Descripción	Interfaz con el SIM, por el protocolo de red TCP Interfaz con el SIMP por el protocolo de red TCP Interfaz con el STARS por el protocolo de red TCP
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 10.2 Requerimiento No Funcional: Interfaz con otros Software.

3.4.7 Requerimientos de Hardware

3.4.7.1 Servidor Web

Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema.		
	Descripción	Procesador Pentium D 2x2 cache. 3.2 GHz. Memoria RAM >= 4 – 8 GB.
	Prioridad	Alta
	Estimado	Durante el Despliegue.

Tabla # 11.1 Requerimiento No Funcional: Requerimientos de Hardware Servidor Web.

3.4.7.2 Servidor de Bases de Datos

Hardware de la estación de trabajo servidor de Base de Datos.		
	Descripción	Procesador Pentium D 2x2 cache. 3.2 GHz. Memoria RAM 4 – 8 GB. Almacenamiento en discos SCSI en espejo y capacidad igual o superior a los 120 GB cada disco. Tecnología de respaldo de datos históricos.
	Prioridad	Alta
	Estimado	Durante el Despliegue.

Tabla # 11.2 Requerimiento No Funcional: Requerimientos de Hardware Servidor de Bases deDatos.

3.4.7.3 Cliente Web

Hardware de la estación de trabajo del cliente.		
	Descripción	Procesador Pentium 3 (o superior). Memoria RAM mínima de 256 MB. Procesador Pentium 3 (o superior). Memoria RAM mínima de 256 MB.
	Prioridad	Alta
	Estimado	Durante el Despliegue.

Tabla # 11.3 Requerimiento No Funcional: Requerimientos de Hardware Cliente Web.

3.4.8 Requerimientos de Software

3.4.8.1 PC Cliente

Software instalado en la estación de trabajo del cliente.		
	Descripción	Sistema Operativo tanto Windows (win9.x o versión superior) como Linux (cualquiera de sus distribuciones). El Navegador Web compatible con HTML 2.0 y CSS, podrá ser Netscape 3 (o superior), Internet Explorer 4.2 (o superior) y compatibles.
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 12.1 Requerimiento No Funcional: Requerimientos de Software PC Cliente

3.4.8.2 PC Servidor Web

Software instalado en el Servidor Web.		
	Descripción	Servidor Web Apache 2.2.X o superior.
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 12.2 Requerimiento No Funcional: Requerimientos de Software PC Cliente.

3.4.8.3 Servidor de Bases de Datos

Software instalado en el Servidor de Base de datos.		
	Descripción	Servidor de Bases de Datos PostGreSQL.
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 12.3 Requerimiento No Funcional: Requerimientos de Software Servidor de Bases de Datos.

3.4.9 Seguridad

Seguridad del sistema.		
	Descripción	**Confidencialidad:** La información manejada por el sistema está protegida de acceso no autorizado y divulgación. **Integridad:** la información manejada por el sistema es objeto de cuidadosa protección contra la corrupción y estados inconsistentes, de la misma forma es considerada igual a la fuente o autoridad de los datos. Se incluye también mecanismos de chequeo de integridad y realización de auditorias por personal calificado de la entidad. **Disponibilidad:** los usuarios autorizados (autenticados por dominio y según su roll) se les garantizará el acceso a la información, los dispositivos o mecanismos utilizados para lograr la seguridad, no ocultarán o retrasarán a los usuarios para obtener los datos deseados en un momento dado.
	Prioridad	Alta
	Estimado	Ciclo de Desarrollo. Despliegue del Sistema.

Tabla # 13 Requerimiento No Funcional: Seguridad.

3.5 Conclusiones Parciales

En este capítulo se abordó la propuesta de la Arquitectura de Software para el proyecto SACGIR, definiendo las 4+1 vistas y los requerimientos no funcionales, que son muy importantes para la AS. Así como la Estructura del Equipo de Desarrollo para una mejor organización del personal.

Se definieron los módulos que conforman al SACGIR, de estos: sus responsabilidades y funciones.

Conclusiones Generales

En el presente trabajo se hizo un estudio detallado de los principales aspectos de la descripción arquitectónica, seleccionando el estilo arquitectónico adecuado para la solución, así como sus componentes, configuraciones y restricciones. Además se realizó un estudio detallado de las principales categorías de patrones (arquitectónicos, diseño e Idioma), se fundamentó la elección de los mismos para su aplicación en el sistema.

Se realizó una valoración de las principales tareas que debe llevar a cabo el rol del arquitecto de software según la metodología de desarrollo RUP.

Dada las restricciones impuestas por algunos requisitos no funcionales del cliente y la Infraestructura de Producción (IP) de la UCI, se definieron las principales tecnología y herramientas a utilizar en el desarrollo del sistema.

Se cumplieron los objetivos trazados en esta investigación y a la vez que se dio respuesta a la pregunta planteada en el problema científico a partir de definir un marco arquitectónico para el proyecto SACGIR.

Recomendaciones

- El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.

- Realizar un estudio de la Arquitectura SOA, debido a su implementación futura en PDVSA, así como la incorporación de un componente de Servicios Web que tendrá la responsabilidad de brindar y/o consumir información por las demás aplicaciones.

- Incorporación de la librería ext., para el mejoramiento de la interfaz de usuario (GUI).

- El estudio de cualquier método de evaluación de la arquitectura existente, con el objetivo de identificar las principales debilidades de la misma, tanto en la descripción arquitectónica como en el propio diseño arquitectónico.

Bibliografía Referenciada

1. **1471-2000(2000), IEEE Std.** *Recommended Practice for Architectural Description of Software Systems.* s.l. : IEEE Computer Society, September, 2000.

2. **Jacobson, I, Booch, G y Rumbaugh, J.** *El Proceso Unifeicado de Desarrollo de Software.* 84-7829-036-2.

3. **Dijsktra, E.** *Co-coperating sequencial* processes . New York : F. Genuys , 1968.

4. **Brook, FP.** *The Mythical Man Month: Essays on Software Engineering.* 1975. 0201835959.

5. **Brown, Malveau y McCorrmick Mowbray, Wiley.** *AntiPatterns. Refactoring Software, Architecture and Projects in Crisis.*

6. **Garlan, D y Shaw, M.** *An Intruduction to Software Architecture.* New Jersey : World Scientific Publishing Company , 1993.

7. **Monroe, R. T, y otros.** *Stylized Architecture, Desing Patterns, and Objects .* 1996.

8. **Reynoso, Carlos y Kicillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft .* Universidad de Buenos Aires : s.n.

9. **Larman, Craig.** *UML y Patrones. Introducción al Análisis y Diseño Orientado a Objeto.*

10. **Gamma, Erich, y otros.** *Desing Patterns. Elements of Reusable Object-Oriented Software(GoF-Group of Four).* 0201633612.

11.*Desarrollo de Aplicaciones Web.* [Enlínea] http://personales.ya.com/juriver/pfc/htmlhelp/ch1s03.html.

12. **SEMATECH, Austin.** *Computer Integred Manufacturing(CIM) Framework Specification- Version 2.0.* 1998 : s.n. 93061697j.

13. SEMATECH World Wide Web . [En línea] http://ismi.sematech.org/publications.htm

14. **Fayad, Mohamed, Scmidt, Douglas y Johndson, Ralph.** *Building Application Frameworks: Objects- Oriented Foundations of Framework Desing .* 1999.

15. *Tomado de la Ayuda del Rational 2003.*

16. **Pérez, Javier Eguíluz.** *Introducción a AJAX.* pág. 282 . Programadores web. Conocimientos básicos de JavaScript .

17. 2007 Zend Corporation. Zend & Zend Studio, MR x Zend Technologies Ltd . [En línea] http://www.zend.com.

18. *PRESENTATION FRAMEWORKS, Sun ONE Architecture Guide .*

19. *Tomado de la Ayuda del Frameworks CakePHP.*

20. CakePHP. [En línea] http://www.cakephp.org and http://www.book.cakephp.org.

21. **Brown, K y Whitenack, B.** *Pattern Languages of ProgramDesing vol.2. Reading, MA.* s.l. : Addison-Wesley, 1996.

22. CakePHP. [En línea] http://www.cakephp.org

http://www.book.cakephp.org.

Bibliografía Consultada

1. Alexander, Christopher. A Timeless. Oxford, University Press : s.n., 1979.

2. Christopher, **Alexander.** *A Pattern Language.* Oxford, Universuty Press : s.n., 1977.

3. **Cueva Lovelle, Juan Manuel.** *Tecnología de Objetos: Patrones de Diseño.* 2004.

4. **Evitts, Paul.** *A UML Pattern Language.* s.l. : SAMS Publishing , 2000.

5. **Flower, Martin.** *Enterprise Application Architecture Patterns .* s.l. : Addison- Wesley , 2003.

6. **Flower, Martin.** *Analysis Patterns:Reusable Object Models.* s.l. : Addison- Wessley, 1997.

7. **M. Fowler, K. Beck, J. Brant, W. Opdyke, and D. Roberts**. *Refactoring: Improving the Desing of Exist Code.* s.l. : Addison-Wesley , 1999.

8. **Group, Hillside.** *Home of the Patterns Library .* 2003.

9.**Cardacci, Darío Guillermo**.Home of the Patterns Library . [En línea] http://hillside.net .

10.**Hophe, Gregor y Woolf, Robert.** *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions.* s.l. : Addisson- Wessley, 2003.

11. **Kerievsky, Joshua.** *Refactoring to Patterns .* s.l. : Addison- Wesley , 2004.

12. **McCormik, Hays.** *Antipatterns Tutorial .* 1998.

13.**McCormick, Hays.** Anttipaterns Tutorial . [En línea] 1998. http://www.antipatterns.com/briefinq/sld001.htm.

14.**Corp, Microsoft.** *Enterprise Solution Patterns.* s.l. : MIcrosoft Press, 2003.

15. HYPERLINK "mailto:lwelicki@hotmail.com" **Welicki , León**. *Enterprise Development Reference Architeture .* s.l. : Microsoft Press,2004.http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/MTJ_2864/default.aspx#authorbrief.

16.**Shalloway, Alan.** *Design Patterns Explained: A New perspective on Object Oriented Desing.* s.l. : Pearson Education , 2001.

17. **Bosch, J.** *Design Patterns as Language Constructs: Journal of Object Oriented Programming(JOOP).* 1998.

18. **Clements, P y Northrop, L.** *Software Product Lines: Practices and Patterns .* s.l. : Addison- Wesley , 2001. 02011703327.

19. **Porter, M.** *Competitive Advantage .* New York : The Free Press, 1985.

20. **Weiss, D.M y Robert Lai, C.T.** *Software Product-Line Engeneering .* s.l. : Addison-Wessley , 1999.

21. *PHP Architect The Magazine for PHP Professionals.* **Coates, Sean, [ed.].** Chicago : s.n., 16-18 de Mayo de 2007, 2007, Vol. 6 Issuse 3. http://phparch.com. 1709-7169.

22. *PHP Architect The Magazine for PHP Professionals, Symfony, Modelo-Vista-Controlador.* **Coates, Sean, [ed.].** Atlanta : s.n., 13-14 de Septiembre de 2007, Vol. 6 Issue 5. http://paharch.com. 1709-7169.

23. *Beginning Ajax with PHP: From Novice to Professional.* **Gilmore, Jason, [ed.].** New York : s.n., 2007. Copyright 2007 by Lee Babin . 13(pbk): 978-1-59059-667-8 - 10(pbk): 1-59059-667-6.

24. *PHP/architect´s Guide to PHP Desing Patterns.* **Streicher, Martin, [ed.].** Julio de 2005, Vol. First Edition . Producida en Canada, Impresa en EEUU. 0-9735898-2-5.

25. **Folmer, E y Bosch, J.** Architectting for usability: a survey. *Journal of Systems and Software.* [En línea] Febrero de 2004. http://is.ls.fi.ump.es/status/results/survey.pdf.

26. **Hernandez Hernandez, Ma. E, Alvarez Carrión, G y MUñoz Arteaga, J.** Patrones de Interacción para el Diseño de Interfaces Web Usables. [En línea] 9 de Abril de 2003. http://ccc.inaoep.mx/~grodrig/Descargas/com10017.pdf .

27. **Hassan, Y, Martín Fernández, F.J y Iazza, G.** Diseño Web Centrado en el Usuario. *Usabilidad y Arquitectura de la Informacion.* [En línea] 2, 2004. http://www.hipertext.net/web/pag206.htm.

28. **Christopher, Alexander y Lea, D.** An Introduction for Object-Oriented Designers. *Software Engineering Notes.* [En línea] 1993. http://q.oswego.edu/dl/ca/ca/ca.html.

29. **Muñoz Arteaga, J, Gonzalez Calleros, J.M y Aguilar Cisneros, J.** Diseño de aplicaciones Interactivas utilizando el Paradigma de Patrones. [En línea] 2004. http://www.mor.itesm.mx/~omayora/TallerHCI-04/CameraReady/JMunoz.pdf .

30. **Welie, M, Veer, G, C y Eliëns, , A.** Patterns as Tools for User Interface Design. [En línea] 2000. http://www.cs.vu.nl/~martijn/gta/docs/TWG2000.pdf .

31. **Alonso, G, y otros.** *Web Services Concepts, Architectures and Aplications.* 2004. 3-540-44008-9.

32. **Buschmann, F.** *Pattern- Oriented Software Architecture.* s.l. : John Wiley & Sons , 1996. 0471958697.

33. **McConnell, Steve.** *Rapid Development.* s.l. : Microsoft Press , 1996. pág. 672. 1-55615-900-6.

34. **Thomas, J, Mowbray y Raphael Malveau, C.** *CORBA Desing Patterns.* [ed.] John Willey. 1997. pág. 334. CD-ROM . 0-471-15882-8.

35. **Pressman, Roger.** *Putnam y Myers: Five core metrics. Dorset House.* 2003.

36. **Bass, L, Clements, P y Kazman, R.** *Software architecture in practice.* 2. s.l. : Addison-Wesley, 2003.

37. **Gutiérrez, Gallardo y Diego, Juan.** *Desarrollo web con PHP5 y MySQL .* s.l. : Anaya Multimedia- Anaya Interactiva , 2004. 84-415-1774-6.

38. **Pérez, César.** *Curso de formación "PHP de la enseñanza".* s.l. : MInisterio de Educación, Cultura y Deporte. Secretaría General Técnica. Centro de Publicaciones, 2003. 84-369-3763-5.

39. *PHP 5.* s.l. : Anaya Multimedia- Anaya Interactiva , 2004. 84-415-1770-3.

40. **Stopford, Andrew.** *Proyectos profesionales con PHP .* s.l. : Anaya Multimedia - Anaya Interactiva , 2002. 84-415-1418-6.

Glosario de Términos

Apache: Servidor de páginas Web de código abierto para diferentes plataformas (UNIX, Windows, etc.).

APIs (Application Programming Interface): Interfaz de Programación de Aplicaciones.

Asíncrona: Sistema que no responde a un reloj constante.

Base de datos: Una base de datos es una colección de información organizada de forma que un programa de ordenador pueda seleccionar.

CGI (Common Gateway Interface) : Interfaz de entrada común (en español) es una importante tecnología de la World Wide Web que permite a un cliente (explorador web) solicitar datos de un programa ejecutado en un servidor web.

Clase(s): Abstracciones de objetos. Una clase es una definición de un objeto. Un objeto es una instancia de una clase.

Cliente/Servidor: Método de distribución de información o de archivos en el cual la agrupación central, servidor, almacena los archivos y los hace disponibles para solicitudes de aplicaciones cliente.

Consultas: Una consulta SQL es tipo de consulta a una base de datos empleando lenguaje SQL.

CSS (Cascading Style Sheets): Las hojas de estilo en cascada contienen un conjunto de etiquetas que definen el formato que se aplicará al contenido de las páginas de una Web.

Dispatcher: Parte de un programa encargada de lanzar un proceso en el servidor de un entorno cliente/servidor.

Enum: Es un objeto de cadena cuyo valor normalmente es escogido entre una lista de valores.

FlagsAttribute: Indica que la enumeración debe tratarse como un conjunto de campos de bits.

Frameworks: Plataforma, entorno, marco de trabajo.

Herramientas CASE (Computer Aided Software Engineering): Se incluyen una serie de herramientas, lenguajes y técnicas de programación que permiten la generación de aplicaciones de manera semiautomática.

Hipertexto (Hypertext): Es un sistema para escribir y mostrar texto que enlaza a información adicional sobre ese texto.

HTML: Lenguaje de marcado de hipertexto, es el lenguaje autoritario para crear documentos en la World Wide Web. Define la estructura de un documento web usando etiquetas y atributos.

HTTP (HyperText Transfer Protocol): Protocolo de Transferencia de Hipertexto.

Interfaz: En software, parte de un programa que permite el flujo de información entre un usuario y la aplicación.

ISVs: Vendedores de software independientes.

JavaScript: Lenguaje de programación interpretado, o sea, no requiere compilación. Es utilizado especialmente en páginas web.

JSP: JavaServer Pages (JSP) es una tecnología Java que permite generar contenido dinámico para web, en forma de documentos HTML, XML o de otro tipo

Lenguaje de Programación C y C++: Un lenguaje de programación es aquel elemento dentro de la informática que permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis.

Managers: Responsable.

Modelo OSI (Open System Interconnection): Interconexión de Sistemas Abiertos.

Multiusuario: Es un tipo de configuración hard-soft que permite soportar a varios usuarios o puestos de trabajo al mismo tiempo.

Objeto wrapper: Objetos envolventes (wrapper objects) los cuales se encargan de obtener y grabar los datos.

OO: Orientado a Objeto.

Open Source : Código abierto o código libre.

PDF (Portable Document Format): Formato de documento portable es el formato de archivos desarrollado por Adobe Systems.

Plug-ins: Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande.

RAM (Random Access Memory): Un tipo de memoria de ordenador a la que se puede acceder aleatoriamente.

Refactorizar clases y tipos: Refactorizar el código consiste en cambiar su estructura interna, especialmente el diseño de sus objetos.

RUP: Metodología de desarrollo de software basada en UML. Organiza el desarrollo de software en 4 fases.

SCSI (Small Computer System Interface): Es un interfaz estándar para la transferencia de datos entre distintos dispositivos del bus de la computadora.

Servlet: Los servlets son programas que funcionan como los CGIs convencionales atendiendo peticiones de un cliente, teniendo al servidor como el encargado.

SIM: Término empleado al Sistema que maneja la Mano Factura.

SIMP: Término empleado al Sistema que maneja Crudos Y Petróleos.

SOAP (Simple Object Access Protocol): Protocolo para el intercambio de mensajes sobre redes de computadoras, generalmente usando HTTP.

SQL: Lenguaje estandarizado de consultas utilizado para consultar bases de datos.

STARS: Término empleado al Sistema que maneja los Precios, Economía y Finanzas.

Subclases: Clase hijo de una clase padre. Algunos lenguajes de programación permiten que una clase tenga múltiples padres.

Subselects: Se utiliza en lugar de la cláusula VALUES, la cual puede contener JOIN, llamadas funciones, y puede incluso consultar en la misma TABLA los datos que se inserta.

Tablas: Estructura que permite almacenar una entidad.

TCP/IP: Son las siglas de Protocolo de Control de Transmisión/Protocolo de Internet (en inglés Transmission Control Protocol/Internet Protocol).

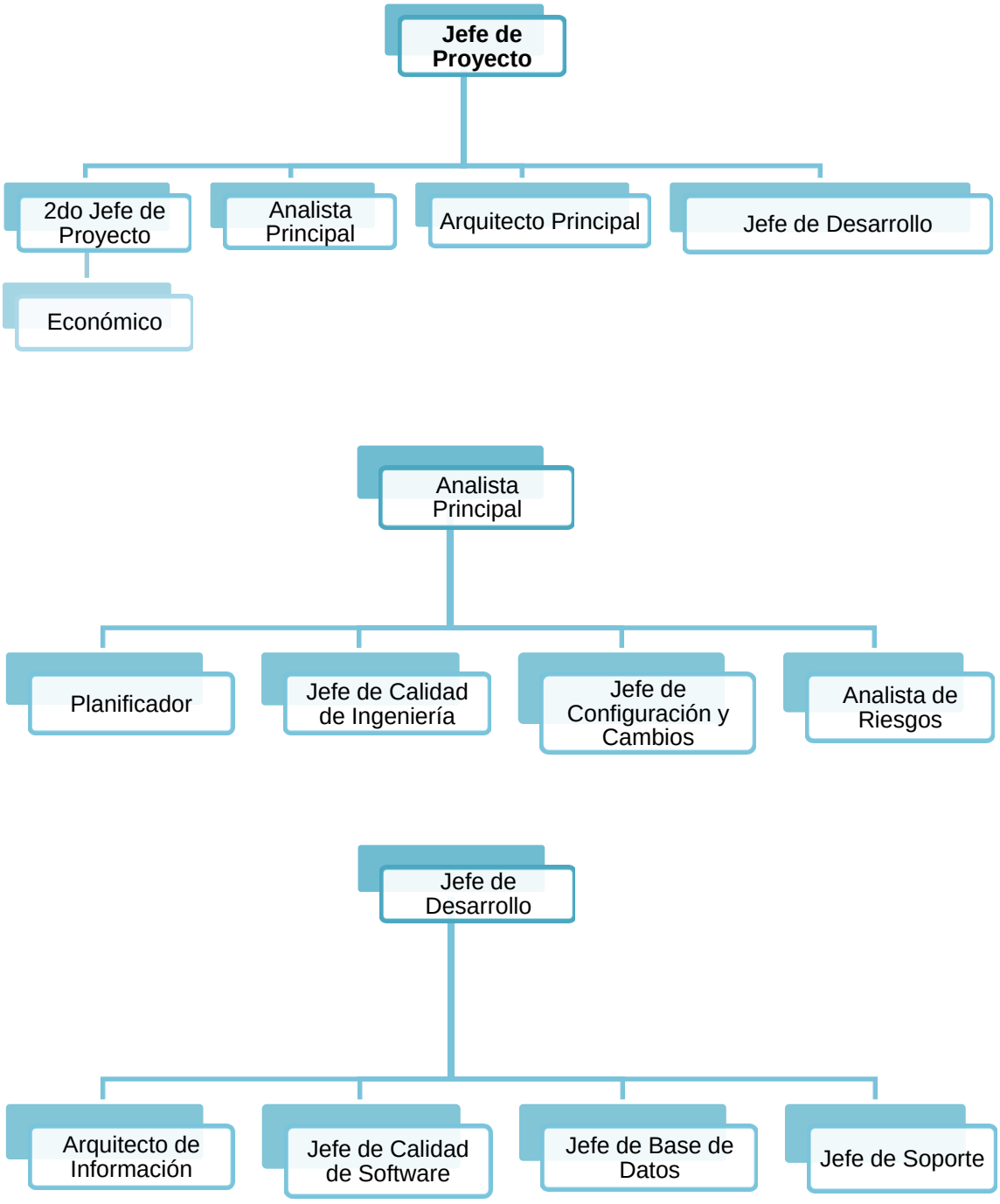
Triggers (Trigger o disparador): Se defina así a una subrutina que es ejecutada de manera automática cuando se produce algún tipo de transacción (inserción, borrado o actualización) en la tabla de una base de datos.

Valores no atómicos: Atributos basados en vectores y conjuntos.

XAMPP: Es un paquete que combina el servidor Apache,base de datos y el lenguaje de programación PHP.

XML (eXtensible Markup Language): Es un lenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

Anexos



Anexo #1 Organización del Equipo de Desarrollo.

Front Controller: /app/webroot/index.php
Command : /cake/distpatcher.php
Registry: /cake/registry.php
Cache : /cake/cache.php
Controller : /cake/controller/controller.php
Factory: /cake/model ConnectionManager::getDataSource()
Active Record: /cake/model/model_php5.php y /cake/model/model_php4.php
Singleton: En la mayoría de las clases del núcleo.

Anexo # 2 Patrones de Diseño del frameworks CakePHP.