Universidad de las Ciencias Informáticas
Facultad 9

Universidad de las Ciencias Informáticas

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

Tema:

PROPUESTA DE ARQUITECTURA DE SOFTWARE PARA EL SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS (Módulo Web).

Autor: Michel Buzón Tur

Tutor: Ing. Dennis Meriño Menadier

Ciudad de La Habana, enero de 2009
"Año 50 de la Revolución"

SIPP

"End, No, the journey doesn't end here. Death is just another path, one that we all must take"

Gandalf the White

SIPP

Agradecimientos

En primer lugar a la Revolución a Fidel y a Raúl, por permitirme ser partícipe de su idea.

A mis padres Magalys y Lázaro por la dedicación, la confianza y el amor que me han brindado durante todos estos años de sacrificio.

A toda mi familia por su apoyo.

A mis amigos que me acompañaron durante todos estos años, en especial a Ángel que ha sido lo más parecido a un hermano que he tenido aquí en la UCI.

A Pimienta y Antonio por su colaboración y apoyo en el lab 103.

A mis profesores que inculcaron en mí ese mar de sabiduría que me ayudó a hoy estar en el lugar que estoy, en especial a mi tutor Dennis por todo el apoyo brindado.

A mis compañeros de grupo por compartir cada momento del día conmigo.

Uno muy especial a la Caboclada por esos momentos buenos y malos vividos…

A Zayli (La Mizu de mi corazón) por brindarme tanto amor y apoyo en los últimos tiempos.

A todos los que me quieren.

Dedicatoria

A mis padres por todo el cariño y apoyo brindado, por ser ellos mi inspiración para lograr ser lo que soy y que vivan orgullosos de mi actuar ante la vida y ante la Revolución.

Resumen

Los procesos de la gestión de la información y los conocimientos adquieren en la actualidad un papel protagónico cuando de lograr eficiencia y calidad se trata, ya sea en el campo investigativo, de formación o producción. Gracias al constante desarrollo de las Tecnologías de la Información y las Comunicaciones (TICS) estos procesos son automatizados.

En la empresa Cuba-Petróleo (Cupet) se manipula gran cantidad de información, la cual se necesita tener centralizada y que sea accesible tanto para la Unión como para el resto de entidades con las que cuenta. El presente trabajo de diploma pretende diseñar una arquitectura de software la cual brinde soporte a una aplicación que permita optimizar todo el proceso de gestión de reportes tanto en los pozos en perforación, en Ceinpet como en las oficinas de la DIPP y así elevar la eficiencia y control en el flujo de información en las diferentes estructuras, brindando la posibilidad de contar con la información lo más actualizada posible.

Palabras Claves

DIPP, CUPET, CEINPET, ARQUITECTURA DE SOFTWARE, MVC, REPORTES

Contenido

Índice de Ilustraciones

Índice de Software

Índice de Tablas

Introducción

La empresa Cuba-Petróleo (Cupet) es una de las empresas más importantes de nuestro país y esta es la encargada junto con otras compañías extranjeras interesadas en este sector del desarrollo petrolífero en Cuba .Esta empresa con sus distintas entidades distribuidas en todo el país intervienen en los diferentes procesos por los que debe pasar el crudo. Un punto clave de todo este extenso proceso ocurre en los pozos de perforación durante la etapa de perforación y extracción del crudo, para que dicha operación sea realizada correctamente y sin contratiempos los expertos que trabajan en estas áreas revisan un cúmulo de información las cuales son vitales para lograr el éxito en su labor. Dentro de los trabajadores que están implicados en esta etapa y más importante por su rol en la misma se encuentra el Supervisor de pozo el cual actúa a especie de cerebro del pozo coordinando los distintos procesos que en el mismo se gestionan, además del trabajo con los reportes diarios de perforación que se derivan de dichos procesos.

Actualmente no existe en CUPET ningún sistema que de manera integral se encargue de gestionar la información que envían los diferentes pozos de petróleos en perforación, debido a lo cual el flujo de información se realiza manualmente o vía correo electrónico. Esto provoca pérdidas de información, errores humanos y contar con información no actualizada de los pozos. Los reportes diarios constituyen prácticamente la información clave del proceso de perforación de pozos, por lo que un error en cualquiera de estos partes significa millones de dólares en pérdidas a la economía del país.

Se plantea como solución una aplicación web que se encargue de automatizar la gestión de información, así como la elaboración de los reportes diarios de información vinculadas a los pozos, tanto en los pozos de petróleo ,en Ceinpet así como en la oficina de despacho de la DIPP .Debido a la importancia de dicha aplicación para el mejor y seguro desarrollo de las labores de perforación y

extracción de crudo en los pozos se considera vital la propuesta de una Arquitectura de Software adecuada al proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS con el fin de velar por el correcto desarrollo y puesta en práctica del software. De lo expuesto anteriormente se determinan una serie de elementos que aportan a la **situación problemática** del proyecto que son:

- La gestión y generación de reportes tanto en los pozos petroleros como en otros centros de Cupet constituye un elemento clave para ese sector económico tan importante para el país. Actualmente la totalidad del trabajo con estos reportes se realiza de forma manual.

- Existen aplicaciones que han automatizado estos procesos, pero han sido desarrolladas en plataformas privativas.

- Demora en el envío de reportes entre los distintos centros de Cupet.

Problema a resolver:

¿Qué arquitectura de software será más factible para desarrollar el proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS?

Objetivo general:

Se plantea definir una arquitectura de software para el proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS.

Objeto de estudio:

Está enmarcado en las diferentes arquitecturas de software más utilizadas en el desarrollo de aplicaciones empresariales.

Campo de acción:

La arquitectura en el proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS.

Hipótesis:

Una selección adecuada, robusta y flexible de arquitectura para el proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS, facilitará un mejor desarrollo de la Aplicación y permitirá alcanzar el éxito del proyecto optimizando el uso de la tecnología para desarrollar la solución correcta que proporcionará valor real a sus usuarios y al negocio al que le dará soporte.

Para darle cumplimiento al objetivo trazado se plantearon las siguientes **tareas**:

1. Evaluar las distintas arquitecturas de software existentes.
2. Determinar las funcionalidades con que debe contar el sistema
3. Evaluar los Patrones arquitectónicos.
4. Definir la estrategia de desarrollo del proyecto.
5. Implementar una primera versión de los modelos.
6. Analizar con el cliente los casos de uso trabajados según la propuesta de arquitectura.
7. Caracterizar la alternativa propuesta.
8. Caracterizar la arquitectura propuesta.

Para la realización del presente trabajo de diploma se utilizaron los diferentes **métodos de investigación.**

Métodos Teóricos.

Análisis y síntesis:

Para conocer, reflexionar y aumentar los conocimientos acerca de la línea de investigación a partir de la consulta de la literatura científica correspondiente, y

luego, haciendo uso de la síntesis confeccionar un cuerpo coherente en el cual se resuman los resultados obtenidos del análisis.

Histórico – Lógico:

Para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos

La inducción y la deducción:

Se empleó para pasar al conocimiento de casos particulares y generales que sirvieron de guía en la formulación teórica de algunas conclusiones.

Modelación:

En la modelación del diseño propuesto

Métodos Empíricos

Observación:

Se utilizó para comprobar el estado real de los pozos petroleros, la DIPP en Varadero y en Ceinpet en consecuencia con las necesidades de obtención de información precisa y concreta a través de una aplicación Web.

Entrevistas:

A todo el personal (ingenieros, técnicos) que de una forma labora en los pozos petroleros, la DIPP y en Ceinpet y que requieren de informaciones precisas y confiables sobre aspectos específicos inherentes a la solución del problema de investigación.

El presente trabajo se encuentra estructurado de la siguiente forma:

Capitulo 1. **ESTADO DEL ARTE Y FUNDAMENTACIÓN TEÓRICA**.

En ésta sección se realiza la descripción del estado del arte para este tipo de aplicación, así como de las tecnologías a usar en el diseño, desarrollo e implementación del proyecto **.**Además se realiza un estudio de las arquitecturas más comunes en la actualidad que más se utilizan en el desarrollo de aplicaciones semejantes a la nuestra.

Capitulo 2. Tecnologías a usar

En esta sección se reflejará la propuesta de la arquitectura de nuestra aplicación y la descripción correspondiente, además de la descripción de cada una de las herramientas y tecnologías a emplear en la misma.

Capitulo 3. **LÍNEA BASE DE LA ARQUITECTURA**

En esta sección se describe la solución arquitectónica del sistema, incluyendo además otros artefactos como son las vistas arquitectónicas definidas por la metodología RUP.

CAPÍTULO 1 ESTADO DEL ARTE Y FUNDAMENTACIÓN TEÓRICA

Introducción.

En ésta sección se realiza la descripción del estado del arte para este tipo de aplicación, así como de las tecnologías a usar en el diseño, desarrollo e implementación del proyecto **.**Además se realiza un estudio de las arquitecturas más comunes en la actualidad que más se utilizan en el desarrollo de aplicaciones semejantes.

1. Estado del Arte

En la actualidad existen una gama de producciones de software para el sector del Petróleo, encaminadas a resolver disímiles problemáticas dentro de esta esfera.
El punto de partida en la búsqueda del petróleo es la exploración que realiza los estudios preliminares para la localización de un yacimiento. Para identificar el petróleo en los poros de las rocas y decidir la mejor forma de extraerlo de las grandes profundidades en la tierra y en el mar, el hombre utiliza los conocimientos de dos ciencias: la Geología y la Geofísica.
La perforación es la segunda etapa en la búsqueda de petróleo. La profundidad de un pozo es variable, dependiendo de la región y de la profundidad a la cual se encuentra la estructura geológica o formación seleccionada con posibilidades de contener petróleo. Una vez comprobada la existencia del petróleo, otros pozos se perforarán para evaluarse la extensión del yacimiento.
Durante la perforación también se toman registros eléctricos que ayudan a conocer los tipos de formación y las características físicas de las rocas, tales

como densidad, porosidad, contenidos de agua, de petróleo y de gas natural. Igualmente se extraen pequeños bloques de roca a los que se denominan “corazones” y a los que se hacen análisis en laboratorio para obtener un mayor conocimiento de las capas que se están atravesando. Con toda la información adquirida durante la perforación del pozo es posible determinar con bastante certeza aspectos que contribuirán al éxito de una operación de terminación. Al finalizar la perforación el pozo queda literalmente entubado (revestido) desde la superficie hasta el fondo, lo que garantiza su consistencia y facilitará posteriormente la extracción del petróleo en la etapa de producción.

Dado este cúmulo de información y reportes que se generan en las distintas etapas se han elaborado distintas aplicaciones para ayudar al mejor cumplimiento de las labores de los especialistas en sus distintas aéreas, por ejemplo:

Well Logger (Creador de Registros de Pozos) (1)

Well Logger permite crear informes de perforación de suelos y diagramas de construcción de pozos. Los ingenieros de proyecto y los geólogos pueden al usar Well Logger en un ordenador portátil en su lugar de trabajo completar rápidamente la documentación necesaria, este proceso ocurre normalmente durante las paradas de los trabajos de perforación. Well Logger ofrece una sencilla, aunque robusta, interfaz de usuario que ofrece presentaciones personalizables, patrones definidos por el usuario, escala ajustable y vista previa de la impresión. Well Logger tiene una interface de hoja de cálculo fácil de usar, con cajas de entrada que simplifican la entrada de datos para cada perforación. La información de entrada incluye litología de la perforación, muestras tomadas, construcción del pozo o detalles anexos de la perforación, y la información general acerca del proyecto y la perforación. Well Logger puede crear sus informes de perforación y pozo en la mitad del tiempo que los programas tradicionales (estilo CAD).

Software WELLSIGHT. (2)

El software WellSight son un conjunto de software utilizados para captación, informes, monitoreo, seguimiento, reconciliación y exportación de datos .Originalmente este software era para el desarrollo petrolífero en la Cuenca occidental de Canadá pero luego de ver sus beneficios se ha extendido su uso a diferentes usuarios del mundo.

Ventajas

- Minimización de tiempo de transferencia de datos con aplicaciones del cliente
- Mejor calidad de datos
- Cuatro opciones diferentes para exportación de datos
- Disponible a todos los usuarios de software WELLSIGHT
- (Sitio Web/Vínculos para descargas)

WellSight System Inc. es una empresa de software Canadiense.

STRATER (3)

Es una potente e innovadora herramienta para el registro de pozos y el trazado de perforaciones. Con esta herramienta los geólogos ya no tendrán que invertir más tiempo y dinero para crear registros profesionales de pozos. Strater es una herramienta potente, sencilla de usar y con un precio muy asequible.

Con Strater se puede visualizar gráficamente:

- Profundidad o elevación
- Notas, comentarios y otros datos de texto
- Petrología, % petrología y descripciones litológicas.

- Potencial espontáneo, rayos gamma, calibrador, neutrónica, (DST), densidad aparente, resistividad, ritmo de perforación, gas total, calidad de los gases y datos sónicos
- Contabilidad de impactos, número y tipo de muestras, permeabilidad, (RQD), lecturas OVM, %recuperación
- Concentración de contaminantes, contenido de humedad y detalles de construcción de pozos.
- Datos de ensayos, petrología de mineralización o alteración, valores BTU (Unidades Térmicas Británicas) y datos de contenido de cenizas.
- Virtualmente cualquier tipo de dato de profundidad o intervalo

Pelotón (4)

Pelotón es una empresa líder en el desarrollo de sistemas de información de Construcción y Operaciones de pozos. La oficina principal de Pelotón está ubicada en Calgary, Alberta. Los productos de Pelotón incluyen WellView, SiteView, RigView y el Generador de Gráficos de Estados Mecánicos

- **WellView**

WellView es un sistema de administración de información de pozos completo, para planificación, perforación, completamientos, mantenimiento y rehabilitación de pozos. Esta base de datos que comprende datos de perforación y del pozo reduce la duplicación de entrada de datos, mejora la calidad del dato y crea un ambiente que facilita compartir datos y la colaboración entre grupos.

- **MasterView**

La tecnología más reciente de Pelotón, simplifica las operaciones suministrando un flujo ininterrumpido de datos entre las aplicaciones de Pelotón y productos externos. Se utiliza el MasterView, para obtener el mayor beneficio de los productos Pelotón, ampliar la integración de datos y su precisión, obtener

reportes exhaustivos en cualquier etapa del ciclo de vida de la operación y prevenir retrasos costosos durante la construcción, perforación y la programación de recursos.

- **RigView**

Es un sistema para cronogramas de proyectos de pozo y equipos capaz de programar las actividades de perforación, completamientos, reacondicionamiento, servicios y pruebas. Comparte archivos y programas, administra actividades, revisa contratos y genera reportes.

- **SiteView**

Es un sistema de información de construcción, rehabilitación y recuperación ambiental de localizaciones. Planifica, programa actividades y mide el desempeño en una base de datos consolidada. Maneja las operaciones más eficientemente y reduce costos.

La totalidad de los softwares antes citado son propietarios lo cual en las condiciones económicas que presenta el país es un obstáculo ya que los costos de licencias y adquisición de los mismos son relativamente elevados. También la economía cubana debe recibir un empuje de tipo informático en la mayoría de sus sectores ya que cuenta con el personal y el conocimiento necesario para así desarrollarlo, además todos estos sectores y en especial el petrolero que es el que nos ocupa deben apoyarse más, siempre que se puede y sea favorable, en el desarrollo tecnológico cubano el cual es mucho más confiable y seguro .Este aspecto ha sido tenido en cuenta en la situación problemática de la investigación lo cual justifica la decisión de desarrollar una aplicación que agrupe mucha de las funcionalidades de los softwares antes citados y contribuir así al mejor funcionamiento de las distintas entidades en Cupet que tiene algún nexo con los reportes que se elaboran en las mismas.

1.1 Tendencia y Tecnologías actuales

1.1.1 Arquitectura de Software

Saber que una Arquitectura de Software, también denominada Arquitectura lógica, consiste en tener en cuenta un conjunto de patrones y abstracciones coherentes que proporcionan el marco de referencia necesario para guiar la construcción del software para un sistema de información. La Arquitectura de Software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. (5)

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. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. (5)

Citando otros autores se tiene que…

La Arquitectura de Software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos. (6)

La arquitectura no es el software operacional. Mas bien, es la representación que capacita al ingeniero del software para: (Primero) analizar la efectividad del diseño para la consecución de los requisitos fijados, (Segundo) considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil, y (Tercero) reducir los riesgos asociados a la construcci6n del software. (7)

Estableciendo una comparación de conceptos entre autores se entiende mejor el expuesto por Pressman y Bass donde ambos se enfocan mejor en la definición real de la arquitectura destacando el trabajo con los componentes del software así como la relación entre los mismos.

.

1.2 ESTILOS ARQUITECTÓNICOS Y PATRONES

Se define que un **estilo arquitectónico** es como una familia de sistemas de software en términos de un patrón de organización estructural, que define un vocabulario de componentes y tipos de conectores y un conjunto de restricciones de cómo pueden ser combinadas. (8)

Buschmann define **estilo arquitectónico** como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. (9)

De las definiciones antes expuestas ambas se entienden como correctas ya que abordan el tema de los distintos componentes organizados estructuralmente.

Algunos estilos arquitectónicos:

Estilos de Flujo de Datos

Tubería y filtros

Estilos Centrados en Datos

Arquitecturas de Pizarra o repositorio

Estilos de llamada y retorno

Model-View-Controller (MVC)

Arquitectura en capas

Arquitecturas orientadas a objetos

Arquitecturas basadas en componentes

Estilos de código móvil

Arquitectura de máquinas virtuales

Estilos heterogéneos

Sistemas de control de procesos

Arquitecturas basadas en atributos

Estilos Peer-to-Peer

Arquitecturas basadas en eventos

Arquitecturas orientadas a servicios

Arquitecturas basadas en recursos

Importancia de los estilos:

- ❖ Sirven para sintetizar y tener un lenguaje que describa la estructura de las soluciones.
- ❖ Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas.
- ❖ Definen los patrones posibles de las aplicaciones.
- ❖ Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

A continuación se abordan algunas de las arquitecturas más utilizadas cuando de realizar una aplicación web se refiere.

1.1.2 Modelo Vista Controlador (MVC)

Es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones Web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página, el modelo es el Sistema de Gestión de Base de Datos y el controlador representa la Lógica de negocio.

Descripción del patrón:

- Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.

- Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

- Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde al modelo.

1.1.3 Arquitectura Cliente/Servidor

Esta arquitectura es definida por IBM como una tecnología que proporciona al usuario el acceso a las aplicaciones, datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo.

Es una arquitectura de procesamientos cooperativa donde uno de los componentes solicita servicios a otro. Consiste en que un programa (cliente) realiza peticiones a otro (servidor), que le proporciona la respuesta. En ella se encuentra distribuida la capacidad de proceso entre el cliente y el servidor, aunque las ventajas de tipo organizativo son más importantes, debido a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita el diseño del sistema.

Qué es un Cliente

El cliente es el que realiza una petición solicitando un servicio, esta solicitud puede convertirse en varias solicitudes de trabajo mediante redes LAN o WAN. La ubicación de los datos o las aplicaciones solicitadas es totalmente transparente para el cliente.

Qué es un Servidor

Un servidor es uno o varios recursos dedicados a responder las solicitudes de los clientes. Pueden estar conectados a los clientes mediante redes LAN o WAN con el fin de proveer múltiples servicios a los clientes como acceso a una base de datos, procesamiento de imágenes, fax, etc. La separación lógica entre el cliente y el servidor permite que el servidor no se ejecute necesariamente sobre una misma máquina ni sea necesariamente un sólo programa.

Ventajas de la Arquitectura Cliente / Servidor.

- Centralización del control: los recursos, la integridad y el acceso a los datos son controlados por el servidor, para prevenir que un programa cliente no autorizado o defectuoso pueda afectar al sistema.

- Escalabilidad: se puede aumentar la capacidad de clientes y servidores por separado.

- Se reduce el tráfico de red considerablemente, el cliente se conecta al servidor utilizando un protocolo de alto nivel.

1.1.4 Arquitectura en Capas

Esta arquitectura consiste en una programación por capa que tiene como objetivo separar la lógica de negocio de la lógica de diseño, su ventaja principal es que el desarrollo se puede realizar en varios niveles, permitiendo que los cambios se hagan en un nivel requerido no afecte a los otros niveles. Esta arquitectura permite distribuir el trabajo en niveles, así, cada grupo de trabajo estará aislado de los demás niveles.

Capas

Capa de presentación:
Es la intermediaria entre el usuario y el sistema, es una interfaz o ventana que captura la información del usuario realizando un filtrado previo para comprobar que no existen errores de formato y le comunica la información solicitada. Esta capa se comunica únicamente con la capa de negocio.

Capa lógica del negocio:
En esta capa es donde se encuentran los programas que se ejecutan y se establecen las reglas que el sistema debe cumplir. Esta capa se comunica con la capa de presentación para recibir las solicitudes y mostrar los resultados, y con la capa datos, para solicitar al gestor de base de datos acciones cómo almacenar o recuperar datos de él.

Capa de acceso a datos:

Está compuesta por uno o varios gestores de bases de datos que realiza el almacenamiento de toda la información, recibe solicitudes de almacenamiento o recuperación de datos desde la capa de negocio. Físicamente estas capas pueden encontrarse en un mismo ordenador, aunque lo más usual es que existan varios ordenadores clientes, que es donde reside la capa de presentación. La capa de negocio y de datos puede separarse en múltiples ordenadores, permitiendo que la base de datos se pueda establecer en varios ordenadores en caso de que su tamaño y complejidad aumente, y recibirán las peticiones del ordenador en que se encuentre la capa de negocio. De igual forma ocurre en caso de que la complejidad sea en la capa de negocio. Cuando los sistemas son muy complejos, por lo general se establece una serie de ordenadores sobre los cuales corre la capa de datos, y otra serie de ordenadores sobre los cuales corre la base de datos.

1.1.5 Patrones

Definición de un patrón

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma".

Christopher Alexander

Decir además que un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones. Surgieron en el año 1977, y fueron inventados por Christopher Alexander.

Citando otros autores tenemos que…

Un patrón es un modelo que podemos seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos al tratar de lograr ciertos objetivos.

Los patrones capturan la experiencia existente y probada para promover buenas prácticas. (10)

Se considera que ambos conceptos tocan el tema clave de los patrones que es el de reutilizar la experiencia humana antes distintos problemas ya probados y solucionados.

Categorías de patrones

Según la escala o nivel de abstracción:

- **Patrones arquitecturales**: Aquellos que expresan un esquema organizativo estructural fundamental para sistemas software.
- **Patrones de diseño**: Aquellos 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.

Patrón Arquitectónico

Se define patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. Buschmann (9)
En líneas generales, un patrón sigue el siguiente esquema:

Contexto. Es una situación de diseño en la que aparece un problema de diseño

Problema. Es un conjunto de fuerzas que aparecen repetidamente en el contexto

Solución. Es una configuración que equilibra estas fuerzas. Ésta abarca:

1. Estructura con componentes y relaciones

2. Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

Con la intención de hacer una comparación clara entre estilo arquitectónico y patrón arquitectónico, la siguiente tabla presenta las diferencias entre estos conceptos: Buschmann (9).

Ver en Anexos Tabla #1

Patrón de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Un patrón de diseño identifica: **Clases**, **Instancias**, **Roles, Colaboraciones** y la **distribución de responsabilidades**. (10)

Un patrón de diseño es:

- Una solución estándar para un problema común de programación.
- Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- Un proyecto o estructura de implementación que logra una finalidad determinada.
- Un lenguaje de programación de alto nivel.
- Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- Conexiones entre componentes de programas.
- La forma de un diagrama de objeto o de un modelo de objeto.

Ventajas

- Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo.
- Por tanto, están basados en la recopilación del conocimiento de los expertos en desarrollo de software.

Idiomas

Estos se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como patrones de idioma, describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y predictibilidad. Los estándares de codificación utilizados están vinculados con el framework de desarrollo utilizado.

Además las convenciones de código son importantes para los programadores por un gran número de razones:

- El 80% del coste del código de un programa va a su mantenimiento.
- Casi ningún software lo mantiene toda su vida el autor original.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.
- Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención.

A continuación las convenciones que se tuvieron en cuenta durante la construcción de la aplicación.

- **Instrucciones de Nomenclatura de las acciones (Métodos):** Las acciones son métodos con el nombre executeNombreAccion de una clase llamada nombreModuloActions que hereda de la clase sfActions y se encuentran agrupadas por módulos.
- **Instrucciones de Nomenclatura de Vistas (Interfaces):** Las vistas son páginas con el nombre NombrePaginaSuccess que hereda de la clase sfView y se encuentran agrupadas por módulos.
- **Instrucciones de Nomenclatura de Variables:** Debido al uso del lenguaje PHP todas las variables declaradas le antecede el símbolo $.
- **Instrucciones de Nomenclatura de comentarios:** Para comentariar cada uno de los distintos métodos que se programaron para la aplicación y para su mejor entendimiento y posterior documentación del mismo se enmarcaron las notas referentes a los mismos entre los símbolos /*…*/

Entre las normas seguidas por el código de Symfony, se encuentra el estándar **"UpperCamelCase**" para el nombre de las clases y variables. Solamente existen dos excepciones: las clases del núcleo de Symfony empiezan por sf (por tanto en minúsculas) y las variables utilizadas en las plantillas que utilizan la sintaxis de separar las palabras con guiones bajos

1.3 Framework Y Modelos

Un Framework simplifica el desarrollo de una aplicación mediante la automatización de algunos de los patrones utilizados para resolver las tareas comunes. Además, un Framework proporciona estructura al código fuente, forzando al desarrollador a crear código más legible y más fácil de mantener. Por último, un Framework facilita la programación de aplicaciones, ya que encapsula operaciones complejas en instrucciones sencillas. (11)

Es un marco de trabajo definido en la cual otro software puede ser construido, puede incluir soporte de programas, bibliotecas, entre otros software para ayudar a desarrollar o unir componentes de un proyecto.

1.3.1 Framework Symfony

Symfony es un completo Framework diseñado para optimizar, gracias a sus características, el desarrollo de las aplicaciones web. Para empezar, separa la lógica de negocio, la lógica de servidor y la presentación de la aplicación web. Proporciona varias herramientas y clases encaminadas a reducir el tiempo de desarrollo de una aplicación web compleja. Además, automatiza las tareas más comunes, permitiendo al desarrollador dedicarse por completo a los aspectos específicos de cada aplicación. El resultado de todas estas ventajas es que no se debe reinventar la rueda cada vez que se crea una nueva aplicación web. (11)

Symfony está desarrollado completamente con PHP 5. Ha sido probado en numerosos proyectos reales y se utiliza en sitios web de comercio electrónico de primer nivel. Symfony es compatible con la mayoría de gestores de bases de datos, como MySQL, PostgreSQL, Oracle y SQL Server de Microsoft. Se puede ejecutar tanto en plataformas *nix (Unix, Linux, etc.) como en plataformas Windows. (11)

1.3.2 Framework Cake

CakePHP es una plataforma de software para desarrollo rápido de aplicaciones en PHP. Comprende librerías, clases e infraestructuras de ejecución para programadores que crean aplicaciones web y que originalmente estuvo inspirado en la plataforma de desarrollo Ruby On Rails. Su objetivo primordial es facilitar una forma de trabajar estructurada y rápida, pero sin perder flexibilidad.

CakePHP posee varias características que lo hacen una gran opción como plataforma de desarrollo rápido de aplicaciones. La principal de ellas es el lenguaje de programación en que se basa, PHP, el más usado en entornos web.

Usa una arquitectura MVC, que obliga a seguir una estructura determinada, de la que nos beneficiaremos a la hora de mantener o ampliar las funcionalidades del código. También incluye una serie de librerías con funciones de AJAX, Javascript, Formularios HTML, etc.

1.3.3 Modelo "4+1"

La arquitectura sigue el Modelo "4+1" vista, presentado en (12), este Modelo define cinco vista para la arquitectura en conjunto con los escenarios, y es presentado en la siguiente figura:

Ver en Anexos Tabla 2

1.4 Metodologías de Desarrollo

Todo desarrollo de software es riesgoso y difícil de controlar, pero si este proceso no es guiado por una metodología, obtenemos como resultado clientes insatisfechos con el producto. La metodología de desarrollo es "un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a realizar nuevo software".
Sintetizando lo anterior, una metodología "representa el camino para desarrollar software de una manera sistemática". Las metodologías persiguen tres necesidades principales:

- Mejores aplicaciones, conducentes a una mejor calidad.
- Un proceso de desarrollo controlado.
- Un proceso normalizado en una organización, no dependiente del personal.

Algunas de las metodologías que se pueden utilizar para guiar un proceso de desarrollo de software son:

- La Metodología Rational Unified Process (**RUP**) es más adaptable para proyectos de largo plazo.
- La Metodología Extreme Programing (**XP**) en cambio, se recomienda para proyectos de corto plazo.
- La Microsoft Solution Framework (**MSF**) se adapta a proyectos de cualquier dimensión y de cualquier tecnología.

1.4.1 Metodología (RUP)

La metodología de desarrollo el **Proceso Unificado de Rational** (*Rational Unified Process* en inglés, habitualmente resumido como **RUP**), es un proceso de desarrollo de software y junto con el Lenguaje Unificado de Modelado UML, constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. El RUP no es un sistema con pasos firmemente establecidos, sino un conjunto de metodologías adaptables al contexto y necesidades de cada organización. (13)

El ciclo de vida RUP es una implementación del Desarrollo en espiral. Fue creado ensamblando los elementos en secuencias semi-ordenadas. A continuación una gráfica que muestra cómo varía el esfuerzo asociado a las disciplinas según la fase en la que se encuentre el proyecto RUP.

Ver en Anexos Ilustración 2

Las primeras iteraciones (en las fases de Inicio y Elaboración) se enfocan hacia la comprensión del problema y la tecnología, la delimitación del ámbito del

proyecto, la eliminación de los riesgos críticos, y al establecimiento de una línea base de la arquitectura.

Durante la fase de inicio las iteraciones hacen mayor énfasis en actividades de modelado del negocio y de requerimientos.

En la fase de elaboración, las iteraciones se orientan al desarrollo de la línea base de la arquitectura, abarcan más los flujos de trabajo de requerimientos, modelo de negocios (refinamiento), análisis, diseño y una parte de implementación orientado a la línea base de la arquitectura.

En la fase de construcción, se lleva a cabo la construcción del producto por medio de una serie de iteraciones.

Para cada iteración se selecciona algunos Casos de Uso, se refina su análisis y diseño y se procede a su implementación y pruebas. Se realizan tantas iteraciones hasta que se termine la implementación de la nueva versión del producto.

En la fase de transición se pretende garantizar que se tiene un producto preparado para su entrega a la comunidad de usuarios. Como se puede observar en cada fase participan todas las disciplinas, pero que dependiendo de la fase el esfuerzo dedicado a una disciplina varía.

Los elementos del RUP son:

Actividades: Son los procesos que se llegan a determinar en cada iteración.
Trabajadores: Vienen hacer las personas o gentes involucradas en cada proceso.
Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.4.2 Metodología XP

La metodología **XP** es una de la más exitosa para el desarrollo de proyectos a corto plazo y consisten en una implementación rápida. Características de XP, la metodología se basa en:

Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir.

Refabricación: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: Consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores.
- La simplicidad, al desarrollar y codificar los módulos del sistema.
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.

1.4.3 Metodología MSF

La metodología **MSF** es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.Esta metodología tiene las siguientes características:

- **Adaptable:** Es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.

- **Escalable:** Puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.

- **Flexible:** Es utilizada en el ambiente de desarrollo de cualquier cliente.

- **Tecnología Agnóstica:** Puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.5 Lenguaje de Modelado

Para representar adecuadamente la arquitectura de un sistema es necesario contar con varios diagramas o vistas (14). Dada la cantidad de características y de elementos que tiene un sistema de software no es posible incluirlos todos en un solo diagrama y que sirva, además, para todas las personas que participan en el desarrollo. Cada una de estas vistas es una estructura de la arquitectura del sistema, que muestran una parte del sistema como un conjunto de componentes, conectores y restricciones sobre sus tipos y relaciones. Además, cada estructura puede relacionarse con las demás para complementar la visión integral del sistema.

Para que la arquitectura se convierta en una herramienta útil dentro del desarrollo y mantenimiento de los sistemas de software es necesario que se cuente con una manera precisa de representarla.

1.5.1 ADL

Las herramientas que se han elaborado para representar una arquitectura de software son los Lenguajes de Definición de Arquitecturas o ADL (Architecture Description Language).

La definición de la arquitectura usando un ADL incluye la posibilidad de especificar la estructura y el comportamiento. Bass (6) proponen que el establecimiento de una notación estándar para la representación de arquitecturas, a través de un lenguaje de descripción arquitectónica, permite mejorar la comunicación entre el autor y el lector, logrando tener un medio de entendimiento común, ahorrando tiempo en indagar el significado de los gráficos que representan la arquitectura y sus componentes. Un ADL también deberá ser capaz de facilitar el modelamiento de sistemas de software que sigan distintos patrones de arquitectura.

Para realizar la descripción de la arquitectura, los ADLs utilizan esencialmente componentes, que son las piezas físicas de la implementación de un sistema, y los conectores para modelar la interacción. Estos conectores pueden ser implícitos en algunos ADLs o bien modelarse como elementos de primera clase (15), dependiendo del ADL y del patrón de arquitectura que se esté modelando.

Es de anotar que casi todos los ADLs se originan en ámbitos universitarios. (16)

Ver en Anexos Tabla 3

Criterios de definición de un ADL

Un ADL debe modelar o soportar los siguientes conceptos:

- ❖ Componentes
- ❖ Conexiones
- ❖ Composición jerárquica, en la que un componente puede contener una sub-arquitectura completa
- ❖ Paradigmas de computación, es decir, semánticas, restricciones y propiedades no funcionales
- ❖ Paradigmas de comunicación
- ❖ Soporte de herramientas para modelado, análisis, evaluación y verificación

Sin embargo, los lenguajes desarrollados hasta el momento presentan diferentes problemas para su utilización en una empresa, como: (30)

- ❖ Requieren una extensa capacitación.
- ❖ No son amigables para presentar la arquitectura a personas ajenas a la construcción del software.
- ❖ No tienen herramientas ni metodologías de apoyo.
- ❖ Algunos se encuentran especializados solo en un tipo particular de sistemas.
- ❖ Sólo tienen en cuenta una sola estructura del sistema

1.5.2 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje gráfico de modelamiento que usa conceptos de orientación por objetos. Este lenguaje tiene una sintaxis y una semántica bien definida, sirviendo además para todas las etapas de desarrollo .En UML se utilizan para el modelamiento de un sistema diferentes elementos y relaciones, que tienen una

semántica y sintaxis definidas. Estos elementos se agrupan en diagramas preestablecidos que corresponden a diferentes proyecciones del sistema. (17) (18).

La gran ventaja de UML es el hecho de que poco a poco se ha venido adoptando en diferentes medios empresariales y académicos como el lenguaje "estándar" para el análisis y diseño de los sistemas de software. Gracias a la posibilidad de extender el UML y a la construcción de herramientas y metodologías que apoyan este lenguaje se ha convertido en el estándar de facto en la actualidad para el modelamiento de los sistemas.

1.6 HERRAMIENTAS CASE

1.6.1 Rational Rose

Rational Rose es una herramienta para modelado visual que automatiza y simplifica la creación y la modificación de los diseños de UML. Es la herramienta CASE[1] desarrollada por los creadores de UML, que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases y entregables.

El navegador UML de Rational Rose permite establecer una trazabilidad real entre el modelo (análisis y diseño) y el código ejecutable. Facilita el desarrollo de un proceso cooperativo en el que todos los agentes tienen sus propias vistas de información (vista de Casos de Uso, vista Lógica, vista de Componentes y vista de Despliegue), pero utilizan un lenguaje común para comprender y comunicar la estructura y la funcionalidad del sistema en construcción.

Sus productos están centrados en la metodología del Proceso Racional Unificado o RUP (Rational Unified Process).

[1] CASE: (Computer-Aided Software Engineering). Ingeniería de Software Asistida por Ordenador.

1.6.2 Visual Paradigm

Herramienta CASE que da soporte al modelado visual con UML 2.0.Dentro de las características que ofrece se encuentran las siguientes:

- Entorno de creación de diagramas para UML 2.0.
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Disponibilidad de integrarse en los principales IDEs.
- Disponibilidad en múltiples plataformas.
- Es posible importar un área de trabajo (WorkSpace). (19) (20)

1.6.3 MagicDraw UML

MagicDraw 10.5 es una herramienta de modelaje con completas características UML. Ha sido implementada al 100% en JAVA y se ejecuta en la mayor parte de las plataformas. Diseñado para los analistas del negocio, los analistas del software, los programadores, los ingenieros del QA, y los escritores de la documentación, esta herramienta de desarrollo dinámica y versátil facilita análisis y el diseño de los sistemas (OO) y de las bases de datos orientados objeto.

Características

- Permite la navegación rápida a través de sus modelos

- Deriva modelos de código de fuente existente en segundos justos
- Elimina la preparación de documentos aburrida con la generación de informe automática
- Amplía capacidades de UML más allá de UML 2,0 en un broche de presión MagicDraw hace esto en minutos sin la codificación adicional (MAGICDRAW).

El principal inconveniente que presenta MagicDraw es la utilización de una máquina virtual de java, esto es precisamente por haber sido desarrollado en java. Esto requeriría demasiados recursos de memoria por lo que pondría muy lenta el accionar con la máquina. (21)

1.7 Sistemas Gestores de Base de datos (SGBD)

En un ambiente multiusuario el SGBD ofrece un control centralizado de la información. Los objetivos que plantean estos sistemas están relacionados con la intención de evitar los problemas que existían en los sistemas de información orientados a los procesos.

Los principales objetivos son:

- Evitar la redundancia de los datos, eliminando así la inconsistencia de los mismos.
- Mejorar los mecanismos de seguridad de los datos y la privacidad. Podemos distinguir cuatro tipos de contextos para usar mecanismos de seguridad: seguridad contra accesos indebidos a los datos, seguridad contra accesos no autorizados a la base de datos, seguridad contra destrucción causada por el entorno (fuego, inundación, robo, etc.), seguridad contra fallos del propio sistema (fallos del hardware, del software, etc.).

- Asegurar la independencia de los programas y los datos, es decir, la posibilidad de modificar la estructura de la base de datos (esquema) sin necesidad de modificar los programas de las aplicaciones que manejan esos datos.
- Mantener la integridad de los datos realizando las validaciones necesarias cuando se realicen modificaciones en la base de datos.
- Mejorar la eficacia de acceso a los datos, en especial en el caso de consultas imprevistas.

Todos estos aspectos, resultarán inútiles si el SGBD no es capaz de responder adecuadamente al conjunto de aplicaciones y a las exigencias de los usuarios.

1.7.1 PostgreSQL

PostgreSQL es un servidor de base de datos relacional de software libre, publicado bajo la licencia BSD. PostgreSQL proporciona un gran número de características que normalmente sólo se encontraban en las bases de datos comerciales tales como DB2 u Oracle.

Entre muchas de sus características citar:

- **DBMS Objeto-Relacional**

  PostgreSQL aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad son consultas SQL declarativas, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, herencia, y arreglos.
- **Altamente Extensible**

  PostgreSQL soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario.

- **Soporte_SQL_Comprensivo**

  PostgreSQL soporta la especificación SQL99 e incluye características avanzadas tales como las uniones (joins) SQL92.

- **Integridad Referencial**

  PostgreSQL soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos de la base de datos.

- **API Flexible**

  La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.

- **Lenguajes Procedurales**

  PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.

- **MVCC**

  MVCC, o Control de Concurrencia Multi-Versión (Multi-Version Concurrency Control), es la tecnología que PostgreSQL usa para evitar bloqueos innecesarios. Si alguna vez ha usado algún DBMS con capacidades SQL, tal como MySQL o Access, probablemente habrá notado que hay ocasiones en las una lectura tiene que esperar para acceder a información de la base de datos. La espera está provocada por usuarios que están escribiendo en la base de datos. Resumiendo, el lector está bloqueado por los escritores que están actualizando registros.

  Mediante el uso de MVCC, PostgreSQL evita este problema por completo. MVCC está considerado mejor que el bloqueo a nivel de fila porque un lector nunca es bloqueado por un escritor. En su lugar, PostgreSQL

mantiene una ruta a todas las transacciones realizadas por los usuarios de la base de datos. PostgreSQL es capaz entonces de manejar los registros sin necesidad de que los usuarios tengan que esperar a que los registros estén disponibles.

- **Cliente/Servidor**

  PostgreSQL usa una arquitectura proceso-por-usuario cliente/servidor. Esta es similar al método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

- **Write Ahead Logging (WAL)**

  La característica de PostgreSQL conocida como *Write Ahead Logging* incrementa la dependencia de la base de datos al registro de cambios antes de que estos sean escritos en la base de datos. Esto garantiza que en el hipotético caso de que la base de datos se caiga, existirá un registro de las transacciones a partir del cual podremos restaurar la base de datos. Esto puede ser enormemente beneficioso en el caso de caída, ya que cualesquiera cambios que no fueron escritos en la base de datos pueden ser recuperados usando el dato que fue previamente registrado. Una vez el sistema ha quedado restaurado, un usuario puede continuar trabajando desde el punto en que lo dejó cuando cayó la base de datos. (22)

1.7.2 MySQL

MySQL Database Server es la base de datos de código fuente abierto más usada del mundo. Su ingeniosa arquitectura lo hace extremadamente rápido y fácil de personalizar. La extensiva reutilización del código dentro del software y una aproximación minimalística para producir características funcionalmente ricas, ha dado lugar a un sistema de administración de la base de datos incomparable en velocidad, compactación, estabilidad y facilidad de despliegue. La exclusiva separación del núcleo (core server) del

manejador de tablas, permite funcionar a MySQL bajo control estricto de transacciones o con acceso a disco no transaccional ultrarrápido.

Qué es MySQL? (23)

- MySQL es un sistema de administración de bases de datos. Una base de datos es una colección estructurada de datos. Esta puede ser desde una simple lista de compras a una galería de pinturas o el vasto monto de información en una red corporativa. Para agregar, acceder y procesar datos guardados en un computador, usted necesita un administrador como MySQL Server. Dado que las computadoras son muy buenas manejando grandes cantidades de información, los administradores de bases de datos juegan un papel central en computación, como aplicaciones independientes o como parte de otras aplicaciones.
- MySQL es un sistema de administración relacional de bases de datos. Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo. Esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

1.8 Lenguajes de Programación y Servidores

Un lenguaje de programación es una técnica estándar de comunicación que permite expresar las instrucciones que han de ser ejecutadas en una computadora. (24)

Actualmente con el desarrollo acelerado de las redes informáticas de comunicación y en especial con el surgimiento de Internet y de la Intranet, las aplicaciones y sitios web se han hecho muy populares, al extremo de utilizarse en complejos sistemas. Esta tecnología necesita pocos recursos por parte del cliente para ser utilizada pues solo requiere de un navegador de Internet.

1.8.1 Perl

Es un lenguaje de propósito general, libre y está distribuido bajo licencia GPL. Inicialmente desarrollado para la manipulación de texto, pero en la actualidad es utilizado para el desarrollo de varias tareas, entre las que se encuentran administración de sistemas, desarrollo Web, programación en red y desarrollo de GUI (del inglés Graphic User Interface, Interfaz Gráfica de Usuario).
Es fácil de usar, eficiente y completo. Soporta tanto la programación estructurada como la programación orientada a objetos y la programación funcional, tiene incorporado un poderoso sistema de procesamiento de texto y una enorme colección de módulos disponibles. (25)

1.8.2 Python

Python es un lenguaje de scripting multiplataforma y orientado a objetos, preparado para realizar cualquier tipo de programa, desde aplicaciones Windows a servidores de red o incluso, páginas web. Es un lenguaje interpretado, lo que significa que no se necesita compilar el código fuente para poder ejecutarlo, lo que ofrece ventajas como la rapidez de desarrollo e inconvenientes como una menor velocidad.
Ventajas que brinda Python

- La cantidad de librerías que contiene, tipos de datos y funciones incorporadas en el propio lenguaje, que ayudan a realizar muchas tareas habituales sin necesidad de tener que programarlas desde cero.
- La sencillez y velocidad con la que se crean los programas. Un programa en Python puede tener de 3 a 5 líneas de código menos que su equivalente en Java o C.
- La cantidad de plataformas en las que podemos desarrollar, como Unix, Windows, OS/2, Mac, Amiga y otros.

- Python es gratuito, incluso para propósitos empresariales (26).

1.8.3 PHP

El lenguaje de programación PHP (*Preprocessed Hypertext Pages*) fue creado por Rasmus Lerdorf a finales de 1994, su principal objetivo está encaminado a la construcción de páginas *Web* del lado del servidor (27).

PHP (acrónimo de "PHP: Hypertext Preprocessor") es un lenguaje interpretado de alto nivel embebido en páginas HTML y ejecutado en el servidor, sin ninguna posibilidad de determinar que código ha producido el resultado recibido. Lo mejor de usar PHP es que es extremadamente simple para el principiante, pero a su vez, ofrece muchas características avanzadas para los programadores profesionales.

Es un lenguaje "open source" y puede ser utilizado en cualquiera de los principales sistemas operativos del mercado, incluyendo Linux, muchas variantes Unix (incluido HP-UX, Solaris y OpenBSD), Microsoft Windows, Mac OS X, RISC OS y probablemente alguno más. Posee la facilidad de funcionar tanto para Unix (con Apache) como para Windows (con Microsoft Internet Information Server) de forma que el código creado para una de ellas no tiene porqué modificarse al pasar a la otra.

Soporta la mayoría de servidores web de hoy en día, incluyendo Apache, Microsoft Internet Information Server, Personal Web Server, iPlanet y Netscape, Oreilly Website Pro server, Caudium, Xitami, OmniHTTPd y muchos otros.

También brinda la posibilidad de usar programación de procedimientos ó programación orientada a objetos. Aunque no todas las características estándares de la programación orientada a objetos están implementadas en la versión actual de PHP, muchas librerías y aplicaciones grandes están escritas íntegramente usando programación orientada a objetos (28)

1.8.6 Servidores Web

Acerca de Apache

El **servidor HTTP Apache** es un software (libre) servidor HTTP de código abierto para plataformas Unix (BSD, GNU/Linux, etc.), Windows, Macintosh y otras, que implementa el protocolo HTTP/1.1 y la noción de sitio virtual. Cuando comenzó su desarrollo en 1995 se basó inicialmente en código del popular NCSA httpd 1.3, pero más tarde fue reescrito por completo. Su nombre se debe a que Behelendorf eligió ese nombre porque quería que tuviese la connotación de algo que es firme y enérgico pero no agresivo, y la tribu Apache fue la última en rendirse al que pronto se convertiría en gobierno de EEUU, y en esos momentos la preocupación de su grupo era que llegasen las empresas y "civilizasen" el paisaje que habían creado los primeros ingenieros de internet. Además Apache consistía solamente en un conjunto de parches a aplicar al servidor de NCSA. Era, en inglés, *a patchy server* (un servidor "parcheado").

Entre sus características tenemos:

- Fiabilidad: Alrededor del 90% de los servidores con más alta disponibilidad funcionan con Apache.
- Gratuidad: Apache es totalmente gratuito, y se distribuye bajo la licencia Apache Software License, que permite la modificación del código.
- Extensibilidad: se pueden añadir módulos para ampliar las ya de por si amplias capacidades de Apache. Hay una amplia variedad de módulos, que permiten desde generar contenido dinámico (con PHP, Java, Perl, Phyton,...), monitorizar el rendimiento del servidor, atender peticiones encriptados por SSL, hasta crear servidores virtuales por IP o por nombre (varias direcciones web son manejadas en un mismo servidor) y limitar el ancho de banda para cada uno de ellos. Dichos módulos incluso pueden ser creados por cualquier persona con conocimientos de programación. (29)

1.9 ROL DE ARQUITECO DEL SOFTWARE

A diferencia de un programador, el Arquitecto de Software debe dominar la mayor cantidad de tecnologías de software y prácticas de diseño, para así poder tomar decisiones adecuadas para garantizar el mejor desempeño, reuso, robustez, portabilidad, flexibilidad, escalabilidad y mantenibilidad de las aplicaciones. Estas decisiones sobre la estructura y dinámica de la aplicación son plasmadas en una notación formal estandarizada como lo es UML.

Resumiendo la idea anterior se puede plantear que…

Hay dos formas de convertirse en un buen arquitecto: aprendiendo a definir las soluciones con base en la propia experiencia (el camino largo), o reutilizando el conocimiento de los expertos a nivel mundial plasmado en patrones de arquitectura y diseño (el camino corto).

1.10 CONCLUSIONES PARCIALES

Se concluye que:
En este capítulo se han expuesto las principales características y definiciones sobre la Arquitectura de Software así como temas referentes a las metodologías de desarrollo que se emplean actualmente en el mundo del desarrollo del software, las clasificaciones de los estilos y patrones tanto arquitectónicos como de diseño para el desarrollo de una Arquitectura de Software específica y las herramientas de modelado y desarrollo que se utilizan para el desarrollo de una aplicación Web. También se ha realizado un estudio profundo del estado del arte para conocer del mundo del petróleo en cuanto a aplicaciones vinculadas con este sector se refiere.

CAPÍTULO 2 TECNOLOGÍAS A UTILIZAR

INTRODUCCIÓN

En esta sección se reflejará la propuesta de la arquitectura de la aplicación y la descripción correspondiente, además de la descripción de cada una de las herramientas y tecnologías a emplear y el porqué de la selección de las mismas.

2.1 Estilo de Arquitectura Definido

2.1.1 ¿Por qué MVC?

MVC

La base arquitectónica del sistema (SIPP) se ha modelado haciendo uso del Patrón (MVC) con el objetivo de utilizar las ventajas que el mismo brinda entre ellas la separación de las responsabilidades de cada una de las capas que lo conforman.

Ver en Anexos Ilustración 3

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, aunque este aún tiene acceso a formularios que se encuentran en el

Modelo. En el sistema la comunicación de las vista con el modelo se realiza solamente por medio del Controlador. La arquitectura de MVC2 es una implementación del MVC modificada, el mayor cambio es que el Modelo ya no dispara los eventos a sus Vistas.

Ver en Anexos Ilustración 4

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

El modelo es el responsable de:

- Acceder a la capa de almacenamiento de datos. Lo ideal es que el modelo sea independiente del sistema de almacenamiento (definido así en el sistema). Definir además 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.

Como nota agregar que el Framework escogido sigue la arquitectura MVC2, a continuación un diagrama del flujo de eventos de MVC2+Symfony.

Ver en Anexos Ilustración 5

Ventajas de MVC

- Múltiples vistas del mismo modelo
- Vistas sincronizadas
- Flexibilidad para cambiar las vistas y los controladores
- La aplicación puede soportar distintos tipos de interfaz de usuario.

- Aumenta en gran medida el nivel de reusabilidad de código. Facilita una evolución continuada de los sistemas, sin puntos de ruptura, ya que un cambio en un sistema afectará a uno o más componentes pero nunca afectará significativamente al núcleo "core" de la aplicación.
- Facilidad de desarrollo y acortamiento del “Tiempo de Comercialización”, debido a la paralelización de tareas.
- Evita poner código indebido en la capa inapropiada.
- Facilita el despliegue de la aplicación en caso de modificaciones en el modelo de datos.

Inconvenientes de MVC

- Complejidad creciente.
- Cambios innecesarios. Puede ser que no todas las vistas estén interesadas en los cambios.
- Conexión entre la vista y el controlador. Hay que usar los dos a la vez.
- Tanto la vista como el controlador son específicos de una plataforma.
- Algunas herramientas de diseño de interfaces de usuario incorporan parte del procesamiento de eventos entrada. El controlador deja de ser necesario.

2.2 Framework Definido

2.2.1 ¿Por qué Symfony?

Ver en Anexos Ilustración 6

Symfony brinda muchas ventajas con su uso entre ellas:

- Fácil de instalar y configurar en la mayoría de plataformas (y con la garantía de que funciona correctamente en los sistemas Windows y *nix estándares)
- Independiente del sistema gestor de bases de datos

- Sencillo de usar en la mayoría de casos, pero lo suficientemente flexible como para adaptarse a los casos más complejos
- Basado en la premisa de "convenir en vez de configurar", en la que el desarrollador solo debe configurar aquello que no es convencional
- Sigue la mayoría de mejores prácticas y patrones de diseño para la web
- Preparado para aplicaciones empresariales y es adaptable a las políticas y arquitecturas propias de cada empresa, además de ser lo suficientemente estable como para desarrollar aplicaciones a largo plazo.
- Código fácil de leer que incluye comentarios de phpDocumentor y que permite un mantenimiento muy sencillo.
- Fácil de extender, lo que permite su integración con librerías desarrolladas por terceros.(11)

Además Symfony automatiza la mayoría de elementos comunes de los proyectos web, como por ejemplo:

- La capa de internacionalización que incluye Symfony permite la traducción de los datos y de la interfaz, así como la adaptación local de los contenidos.
- La capa de presentación utiliza plantillas y *layouts* que pueden ser creados por diseñadores HTML sin ningún tipo de conocimiento del Framework. Los *helpers* incluidos permiten minimizar el código utilizado en la presentación, ya que encapsulan grandes bloques de código en llamadas simples a funciones.
- Los formularios incluyen validación automatizada y relleno automático de datos (*"repopulation"*), lo que asegura la obtención de datos correctos y mejora la experiencia de usuario.
- Los datos incluyen mecanismos de escape que permiten una mejor protección contra los ataques producidos por datos corruptos.

- La gestión de la caché reduce el ancho de banda utilizado y la carga del servidor.
- La autenticación y la gestión de credenciales simplifican la creación de secciones restringidas y la gestión de la seguridad de usuario.
- El sistema de enrutamiento y las URL *limpias* permiten considerar a las direcciones de las páginas como parte de la interfaz, además de estar optimizadas para los buscadores.
- El soporte de e-mail incluido y la gestión de APIs permiten a las aplicaciones web interactuar más allá de los navegadores.
- Los listados son más fáciles de utilizar debido a la paginación automatizada, el filtrado y la ordenación de datos.
- Las interacciones con Ajax son muy fáciles de implementar mediante los *helpers* que permiten encapsular los efectos Javascript compatibles con todos los navegadores en una única línea de código.(11)

También Symfony puede ser completamente personalizado para cumplir con los requisitos de las empresas que disponen de sus propias políticas y reglas para la gestión de proyectos y la programación de aplicaciones. Por defecto incorpora varios entornos de desarrollo diferentes e incluye varias herramientas que permiten automatizar las tareas más comunes de la ingeniería del software:

- Las herramientas que generan automáticamente código han sido diseñadas para hacer prototipos de aplicaciones y para crear fácilmente la parte de gestión de las aplicaciones.
- El Framework de desarrollo de pruebas unitarias y funcionales proporciona las herramientas ideales para el desarrollo basado en pruebas (*"test-driven development"*).
- La barra de depuración web simplifica la depuración de las aplicaciones, ya que muestra toda la información que los programadores necesitan sobre la página en la que están trabajando.

- La interfaz de línea de comandos automatiza la instalación de las aplicaciones entre servidores.
- Es posible realizar cambios *"en caliente"* de la configuración (sin necesidad de reiniciar el servidor).
- El completo sistema de log permite a los administradores acceder hasta el último detalle de las actividades que realiza la aplicación.(11)

A continuación se mostrará la estructura de carpetas que presenta el Framework y explicaremos las más importantes :(11)

Ver en Anexos Ilustración 7

Donde:

App: Contiene un directorio por cada aplicación del proyecto (normalmente, frontend y backend para la parte pública y la parte de gestión respectivamente)

Config: Almacena la configuración general del proyecto
Data: En este directorio se almacenan los archivos relacionados con los datos, como por ejemplo el esquema de una base de datos, el archivo que contiene las instrucciones SQL para crear las tablas e incluso un archivo de bases de datos de SQLite.
Lib: Almacena las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto. El subdirectorio Model guarda el modelo de objetos del proyecto.

2.3 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, se debe romper

el proceso de diseño web en pequeños pasos independientes basados en los detalles importantes dentro de estos requerimientos, con el objetivo de tener una visión de cómo podemos aplicar los Patrones en el sistema. Los patrones que se exponen a continuación se tuvieron en cuenta en el Framework Symfony o fueron tenidos en cuenta en la etapa de implementación de la aplicación.

2.3.1 Singleton

Propósito

El patrón de diseño **Singleton** (instancia única) está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella.

Estructura

Ver en Anexos Ilustración 8

Cuando usarlo

- Cuando debe haber únicamente una instancia de una clase y debe ser claro su acceso para los clientes.
- Cuando la “Instancia Única” 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 “Instancia Única” 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).

2.3.2 Command

Propósito

El patrón de diseño **Command** (Orden) está diseñado para encapsular una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma. Además este patrón permite solicitar una operación a un objeto sin conocer realmente el contenido de esta operación, ni el receptor real de la misma.

Estructura

Ver en Anexos Ilustración 9

Donde:

- AbstractCommand.

Clase que ofrece un interfaz para la ejecución de órdenes. Define los métodos do y undo que se implementarán en cada clase concreta.

- ConcreteCommand.

Clase que implementa una orden concreta y sus métodos do y undo. Su constructor debe inicializar los parámetros de la orden.

- Invoker.

Clase que instancia las órdenes, puede a su vez ejecutarlas inmediatamente (llamando a do) o dejar que el CommandManager lo haga.

- CommandManager.

Responsable de gestionar una colección de objetos orden creadas por el Invoker. Llamará a los métodos doIt y unDoIt. Gestionará su secuenciación y reordenación

Conocer que:

- ❖Que tan inteligente debe ser un comando*:* se debe considerar que un Command puede simplemente invocar a un receiver o puede realizar operaciones complejas que ningún otro objeto está en capacidad de realizar por sí solo.
- ❖Soporte a la opción deshacer y rehacer: para implementar estas opciones el ConcreteCommand debe incluir información extra, pertinente al estado para así poder deshacer o rehacer una solicitud.
- ❖ Evite la acumulación de errores en el proceso de deshacer*:* en la medida en que se hacen y deshacen operaciones es posible que el estado al que se llega diverge del estado original de los objetos. Es necesario que el ConcreteCommand contenga suficiente información para que sea capaz de hacer que los objetos vuelvan al estado original.

2.3.3 Front Controller

Propósito

Para aplicaciones Web se recomienda utilizar este patrón que obliga a que 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 .Success.
- ❖ Normalmente se utiliza junto con un Dispatcher que es el responsable de redirigir el flujo de ejecución hacia la .Success adecuada. Este Dispatcher puede ser realizado por el propio controlador o estar en una clase a parte.

Estructura

Ver en Anexos Ilustración 10

Aspectos de interés

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

2.3.4 Experto

Propósito

El patrón de diseño **Experto** está diseñado en que 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. Hay que tener en cuenta que esto es aplicable mientras estemos considerando los mismos aspectos del sistema:

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

2.3.4 Controlador

Propósito

El patrón de diseño **Controlador** está diseñado para 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.

2.3.5 Active Record (Registro Activo)

Propósito

Active Record es un patrón en el cual, el objeto contiene los datos que representan a un renglón (o registro) de la tabla o vista, además de encapsular la

lógica necesaria para acceder a la base de datos. De esta forma el acceso a datos se presenta de manera uniforma a través de la aplicación. Lógica de Negocio + Acceso a Datos en una misma clase. Una clase Active Record consiste en el conjunto de propiedades que representa las columnas de la tabla más los típicos métodos de acceso como las operaciones CRUD, búsqueda (Find), validaciones, y métodos de negocio.

En que situaciones CONVIENE usar este patrón:

- Lógica de negocio simple y poco relacionado con otras entidades.
- Es ideal cuando la estructura de la tabla coincide con la estructura de la clase.

En que situaciones NO CONVIENE de Active Record:

- Es simple. Esto es bueno y malo al mismo tiempo. Con lógica de negocio compleja, este patrón pierde coherencia.
- Otra desventaja que al estar tan acoplado a la estructura de la clase, un cambio en el diseño de la tabla, implica cambiar la clase.
- En situaciones de operaciones de alto volumen de datos, el overhead que se paga en el pasaje y carga de datos, es innecesario. Esta desventaja aplica tanto a Active Record, como a cualquier otro diseño orientado a objetos.
- Muchos “puristas” de OOP critican que Active Record tiene una misma clase tanto la responsabilidad de acceder a la base como de manejar la lógica de negocio y “ensucia” el código.

2.3.6 Registro

Propósito

Un objeto conocido que otros objetos pueden utilizar para encontrar objetos y servicios comunes. Registro es en esencia un objeto global o al menos parece uno, (Singleton) el cual es accedido directamente por otros objetos.

2.3.7 Caché

El patrón de diseño **Caché** 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.

2.3.8 Decorator + Composite

Patrón Decorator (Envoltorio)

Este patrón describe cómo agregar funciones a los objetos dinámicamente. Decorator es un patrón estructural que compone los objetos recursivamente para permitir una participación abierta a una serie de responsabilidades adicionales. Estas responsabilidades se añaden a los objetos en vez de añadir a las clases.

Estructura

Ver en Anexos Ilustración 11

Donde:

- **Componente:** define la interfaz para los objetos que pueden tener las responsabilidades que se les añade dinámicamente.
- **Decorador:** mantiene una referencia a un componente de objeto y define una interfaz que se ajusta a los componentes de la interfaz.

- **ConcreteComponent:** define un objeto para nuevas responsabilidades que se pueden adjuntar.
- **ConcreteDecorator:** agrega responsabilidades al componente

La principal ventaja de usar este patrón radica en que hay más flexibilidad en la adición de responsabilidades a los objetos que se pueden añadir estáticos con la herencia.

Patrón Composite (Objeto compuesto)

Este patrón es otro ejemplo de un objeto patrón estructural. En él se describe cómo crear una clase formada por la jerarquía de clases de dos tipos de objetos (primitivas y compuesto) sin preocuparse de cómo tratar a los objetos en una forma diferente o similar. El compuesto de objetos primitivos nos componen y otros compuestos objetos arbitrarios en estructuras complejas. El patrón de compuestos y, más concretamente, la composición por agregación, nos ayuda a crear estructuras de *árbol* para encapsular no sólo relacionados con objetos, sino una serie de objetos, que pueden contener otros objetos del mismo o de otros tipos. Esto resulta en una especie de objeto árbol, o sea, permite tratar objetos compuestos como si de uno se tratase.

Estructura

Ver en Anexos Ilustración 12

Donde:

Componente (Component)

- Declara la interfaz para los objetos en la composición
- Implementa el comportamiento predeterminado para la interfaz común a las clases

- Declara una interfaz para el acceso y la gestión de sus componentes hijos

Hoja (Leaf)

- Representa la hoja de objetos en la composición
- Define el comportamiento de los objetos primitivos en la composición

Compuesto (Composite)

- Define el comportamiento de los componentes de tener hijos
- Almacena componentes hijos
- Ejecuta operaciones relacionadas con los hijos en la interfaz de componentes.

Cliente

- Manipula objetos en la composición a través de la interfaz de componente

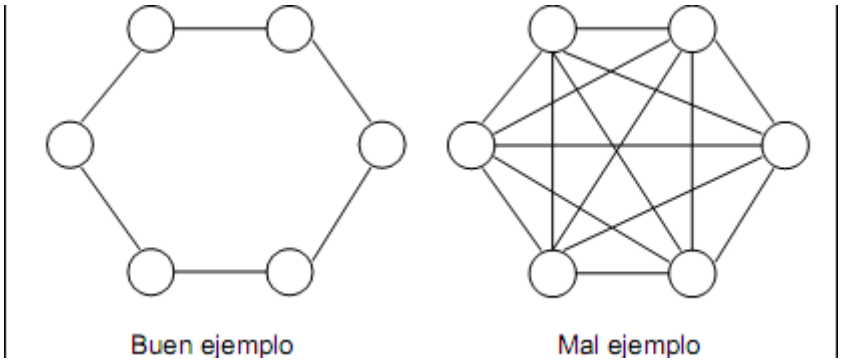
La principal ventaja de usar este patrón radica en que puede tratar objetos compuestos como si de uno se tratase.

2.3.9 Bajo Acoplamiento

¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?
El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo (o débil) acoplamiento no depende de muchas otras. Una clase con alto (o fuerte) acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente, presentan los siguientes problemas:

- ❖ Los cambios de las clases afines ocasionan cambios locales.
- ❖ Son más difíciles de entender cuando están aisladas.
- ❖ Son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.

Ejemplo



Ventajas

- ❖ El Bajo Acoplamiento es un principio que debemos recordar durante las decisiones de diseño: es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño.
- ❖ El Bajo Acoplamiento estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento.
- ❖ El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades.

- No se afectan por cambios de otros componentes.
- Fáciles de entender por separado.
- Fáciles de reutilizar.

2.3.10 Alta Cohesión

¿Cómo mantener la complejidad dentro de límites manejables?

En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.

Una clase con baja cohesión hace muchas cosas no afines o un trabajo excesivo. No conviene este tipo de clases pues presentan los siguientes problemas:

- Son difíciles de comprender.
- Son difíciles de reutilizar.
- Son difíciles de conservar.
- Son delicadas: las afectan constantemente los cambios.

Las clases con baja cohesión a menudo representan un alto grado de abstracción o han asumido responsabilidades que deberían haber delegado a otros objetos.

Ventajas

- Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica a1 valorar sus decisiones de diseño.

- Una regla práctica es la siguiente: una clase de alta cohesión posee un número relativamente pequeño, con una importante funcionalidad relacionada y poco trabajo por hacer. Colabora con otros objetos para compartir el esfuerzo si la tarea es grande. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizara. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y 1os mejoramientos. La ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización.

- El patrón Alta Cohesión - como tantas otras cosas en la tecnología de objeto - presenta semejanzas con el mundo real. Todos sabemos que, si alguien asume demasiadas responsabilidades - sobre todo las que debería delegar -, no será eficiente.
- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de
- reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

2.4 Lenguaje de Modelado

2.4.1 ¿Por qué UML?

Las desventajas que presentan los ADLs pueden ser superadas si se utiliza un lenguaje de Modelamiento que sea conocido en la industria y que además esté apoyado por herramientas y metodologías de desarrollo, este lenguaje de Modelamiento es UML, que se está convirtiendo en una notación estándar de hecho en las empresas. UML permite que se represente de manera semi-formal la estructura general del sistema, con la ventaja de que este mismo lenguaje

puede ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios.

2.5 Herramienta Case Seleccionada

2.5.1 ¿Por qué 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. 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.
En SIPP la versión a utilizar es la Enterprise Edition, debido a las características que brinda: (19) (20)

- 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.

2.6 Lenguaje de Programación Seleccionado

2.6.1 ¿Por qué PHP?

Luego de haber analizado los diferentes lenguajes de programación se concluye que es conveniente la utilización de PHP por los siguientes aspectos: (27) (28)

- Está soportado en la mayoría de las plataformas de Sistemas Operativos, mientras que con ASP –la cuál es propiedad de Microsoft- no es multiplataforma.
- Es libre, lo cual presenta una alternativa de fácil acceso para todos
- PHP se puede ejecutar en múltiples servidores.
- PHP tiene una estructura limpia y ordenada, muy parecida a C, lo que lo hace fácil de comprender, y una vez aprendido, es más fácil aprender otros como Java Script, ActionScript, o C y sus derivados.
- LA plataforma .NET es lenta pero sobre todo el costo de licencia del IDE es muy elevado.
- Para desarrollar en PHP no requieres nada más que un editor de texto plano.
- PHP se actualiza constantemente.
- PHP es abierto a diferentes arquitecturas y paradigmas de programación.
- PHP se comunica de forma directa con distintas bases de datos, de forma nativa.

Otro aspecto considerado a la hora de la elección de este lenguaje de programación es la experiencia y el conocimiento del equipo de desarrollo de dicho lenguaje y de sus ventajas.

2.7 Gestor de Base de datos Definido

2.7.1 ¿Por qué PostgreSQL?

Una de las principales características que hace de PostgreSQL tan popular es precisamente que se distribuye bajo licencia GPL.

Además provee las siguientes características:

Instalación ilimitada

- PostgreSQL, no tiene costo asociado a la licencia del software.
- Esto tiene varias ventajas adicionales:
- Modelos de negocios más rentables con instalaciones a gran escala.
- No existe la posibilidad de ser auditado para verificar cumplimiento de licencia en ningún momento.
- Flexibilidad para hacer investigación y desarrollo sin necesidad de incurrir en costos adicionales de licenciamiento.

Extensible

- El código fuente está disponible para todos sin costo.

Multiplataforma

- PostgreSQL está disponible en casi cualquier Unix (34 plataformas en la última versión estable), y una versión nativa de Windows está actualmente en estado beta de pruebas.

Interfaz con diversos lenguajes

- C, C++, Java, Delphi, Python, Perl, PHP, Bash entre otros.

Una lista breve de características técnicas que PostgreSQL ofrece:

- Cumple completamente con ACID
- Cumple con ANSI SQL
- Integridad referencial
- Replicación (soluciones comerciales y no comerciales) que permiten la duplicación de bases de datos maestras en múltiples sitios de replica
- Interfaces nativas para ODBC, JDBC, C, C++, PHP, Perl, TCL, ECPG, Python y Ruby.
- Reglas.
- Vistas.
- Triggers.
- Unicode.
- Secuencias.

- Herencia.
- Outer Joins.
- Sub-selects.
- Una API abierta.
- Procedimientos almacenados.
- Soporte nativo SSL.
- Lenguajes procedurales.
- Respaldo en caliente.
- Bloqueo a nivel mejor-que-fila.
- Índices parciales y funcionales.
- Autentificación Kerberos nativa.
- Soporte para consultas con UNION, UNION ALL y EXCEPT.
- Extensiones para SHA1, MD5, XML y otras funcionalidades.
- Herramientas para generar SQL portable para compartir con otros sistemas compatibles con SQL.
- Sistema de tipos de datos extensible para proveer tipos de datos definidos por el usuario, y rápido desarrollo de nuevos tipos.
- Funciones de compatibilidad para ayudar en la transición desde otros sistemas menos compatibles con SQL.

La versión 8.2 es la que se utiliza para darle cumplimiento a los objetivos trazados en un inicio, ésta es una versión estable. Otro aspecto tenido en cuenta con la elección de este gestor es la experiencia y el conocimiento asociado con la herramienta que presenta el equipo de desarrollo del proyecto. (22)

2.8 Metodología de Desarrollo Definida

2.8.1 ¿Por qué RUP?

El ciclo de vida de RUP organiza las tareas en fases e iteraciones. Esta divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades. El equipo de trabajo se encuentra familiarizado con esta metodología lo cual es un elemento a tener en cuenta ya que se ahorra tiempo en capacitación al mismo además de la amplia documentación que brinda RUP, factor clave para el proyecto ya que le aporta requisitos de calidad, escalabilidad y organización al futuro software a desarrollar.

Existieron además dos elementos los cuales influyeron en la elección de esta metodología, el trabajo con el cliente y la plataforma de desarrollo que brinda la metodología para su desarrollo.

2.9 Otros Aspectos

IDE

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic, por ejemplo, puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.

Los IDE proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Python, Java, C#, Delphi, Visual Basic, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

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 plugins 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.

Réplica de Datos

La réplica de datos es una técnica que permite copiar y distribuir idénticamente las tablas de una base de datos en múltiples bases de datos ubicadas en diferentes nodos de la red. La replicación asegura que los datos correctos estén siempre disponibles en el momento y en el lugar necesario.

¿Por qué usar replicación?

Las instituciones utilizan la replicación en sus aplicaciones por varias razones, las cuáles pueden ser categorizadas de la siguiente forma:

- Distribución de datos a otras ubicaciones
- Consolidación de datos desde otras ubicaciones
- Intercambio bidireccional de datos con otras ubicaciones
- Alguna variantes o combinaciones de los anteriores

Existen en el mundo múltiples soluciones que permiten llevar a cabo la réplica de datos enfocadas al gestor de base de datos utilizado en cuestión. Pueden citarse entre ellos Postgres-R, PgReplicator, Usogres, ErServer, aunque ninguno de ellos implementa una solución de réplica para bases de datos fragmentadas.

Para la réplica de datos de SIPP se empleo un software desarrollado por la propia universidad, DEW:
El DEW surge como respuesta a la necesidad de mantener actualizados los datos del Sistema de Gestión Penitenciaria Venezolano (SIGEP). Se fundamenta en la copia de datos de una localización a otra. Pretende cubrir las principales necesidades relacionadas con la distribución de datos entre los gestores más populares como la protección, recuperación, sincronización de datos, transferencia de datos entre diversas localizaciones o la centralización de la información en una única localización.
Permite además la representación de complejos escenarios de replicación con herramientas para la definición de localizaciones o nodos y la selección de los datos de replicación. Cuenta además con una herramienta Web para la administración y monitoreo de los datos de replicación.

2.10 Conclusiones Parciales

Se concluye que:
El binomio RUP/UML será la metodología y lenguaje usados para el proceso de desarrollo del SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS. Por demás se cuenta con una potente herramienta para la modelación, la Herramienta CASE*:* Visual Paradimg. Como Framework de trabajo contaremos con Symfony y Lenguaje de Programación PHP5.Como SGBD contaremos con PostgreSQL y como IDE de Desarrollo Eclipse.

CAPÍTULO 3 LÍNEA BASE DE LA ARQUITECTURA

Introducción

En esta sección se describe la solución arquitectónica del sistema, incluyendo además otros artefactos como son las vistas arquitectónicas definidas por la metodología RUP.

3.1 Arquitectura del Negocio

Como se ha planteado anteriormente el negocio se enmarca en 3 entidades de Cupet que son la DIPP, el Ceinpet y los pozos de perforación. La DIPP está conformada por una serie de Tráiler los cuales todos están interconectados con cable UTP par trenzado con una velocidad de 100 Mbps .La información que llega a este centro es a través de una línea conmutada (frame relay 256).Se considera que existe buena conectividad en esta entidad.

En Ceinpet igual existe buena conectividad desde la Habana, no ocurre así hacia los pozos o viceversa.
En los pozos de perforación la distribución de los puestos de trabajo es en una serie de trialers los cuales todos están interconectados con cable UTP par trenzado con una velocidad de 100 Mbps

Como nota agregar que la principal dificultad vinculada al flujo de información desde los pozos hacia los distintos centros donde se procesa esa información es el tema relacionado con la conexión de los pozos con el exterior donde presentan frecuentes perdidas del servicio.

3.2 Estructura del Equipo de Desarrollo

3.2.1 Ambiente de Desarrollo

A continuación se mencionará las herramientas que se definieron en el proyecto SISTEMA DE INFORMACION DE PERFORACION DE POZOS PETROLEROS, de esta forma nos ayudará a conformar los puestos de trabajo por roles.

Herramientas de Modelado.

- Lenguaje de Modelado: UML 2.0
- Herramientas CASE: Visual Paradimg, versión: Enterprise Edition

Herramientas de Desarrollo.

- Lenguaje de Programación PHP5 y Framework Symfony.
- Gestor de Base de Datos PostgreSQL.
- Macromedia Dreamweaver 8
- IDE Eclipse
- Software de Replicación Slony I

3.2.2 Estructura del Equipo de Trabajo

El equipo de trabajo está conformado así:

Ver en Anexos Tabla 4

3.2.3 Configuración de los puestos de trabajos por roles

- **Arquitecto**
  1. PC, con mouse y teclado.
  2. Dreamweaver 8, Apache, Symfony

3. Visual Paradimg.
4. PostgreSQL
5. Instalación del paquete Office/ Open Office.

❖ **Analista**

1. PC, con mouse y teclado.
2. Instalación de Visual Paradimg.
3. Instalación del paquete Office/ Open Office.

❖ **Implementador**

1. PC, con mouse y teclado.
2. Dreamweaver 8, Apache, Symfony.

❖ **Diseñador BD**

1. PC, con mouse y teclado.
2. Instalación de Visual Paradimg.
3. Instalación de PostgreSQL

3.3 Organigrama de la Arquitectura

3.3.1 Visión General de la Arquitectura

El siguiente diagrama representa la estructura del sistema SIPP:

Ver en Anexos Ilustración 13

Descripción de responsabilidades de los distintos componentes del diagrama:

Componente 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 controladora.

Componente 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.
- Recibe peticiones realizadas por la capa Interfaz y las entrega a la modelo.

Componente Modelo:

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

- Control de los roles de usuario para en contraste con las reglas del negocio.
- Ejecuta las peticiones realizadas por la controladora.
- Realiza peticiones a través de la ORM generada.

Acceso a Datos (ORM): *(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 Tecnologías que se emplearan en cada componente:

Ver en Anexos Tabla 5

3.3.2 Módulos del Sistema

- Módulo del Pozo
- Módulo de la DIPP
- Módulo de Seguridad
- Módulo de Visualización de la Información

- Módulo de Manejo de Archivo

Responsabilidades de cada módulo:

Módulo del Pozo:

- Gestiona toda la información referente con los reportes que se elaboran en el pozo.

Módulo de la DIPP:

- Gestiona toda la información referente a la recepción de los reportes enviados desde el pozo así como la elaboración de los partes que elabora dicha oficina.

Módulo de Seguridad:

- Control de la información del sistema.
- Control de creación y asignación Roles de usuarios.
- Validaciones

Módulo de Visualización de la Información:

- Encargado de la visualización de las distintas graficas necesarias en los distintos reportes.

Módulo de Manejo de Archivo:

Encargado del trabajo con archivos por parte del sistema así como de las salidas a la información existente con el formato adecuado.

3.3.3 Restricciones de acuerdo a la estrategia de diseño

- El diseño de la aplicación, se hará aplicando el paradigma de la programación orientada a objetos.

- Se utilizarán las tecnologías que brindan el Framework definido.

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, se utilizaran las vistas arquitectónicas definidas por la metodología RUP.

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

3.4.1 Vista de Caso de Uso

En esta vista se muestra la percepción que tiene el usuario de las funcionalidades del sistema mediante la representación de los actores y casos de usos más importante.

Los casos de uso describen un conjunto de secuencias de acciones, incluyendo variaciones que un sistema lleva a cabo y que conduce a un resultado observable de interés para un determinado actor. El sistema cuenta con 39 Casos de Uso, de los cuales 9 son arquitectónicamente significativos; los cuales comenzarán a desarrollarse en la primera iteración del proyecto. Están representados por módulos de la siguiente forma:

Ver en Anexo Ilustración 1

Ver en Anexo Tabla 6

3.4.2 Vista Lógica

Ver en Anexos Ilustración 14

A continuación se brinda una descripción de la estructura y de los distintos elementos del Framework:

Ver en Anexos Tabla 7

3.4.3 Vista de despliegue

La arquitectura física o Vista de despliegue toma en cuenta primeramente los requisitos no funcionales del sistema tales como la disponibilidad, confiabilidad (tolerancia a fallas), rendimiento (performance), y escalabilidad. El software ejecuta sobre una red de computadores o nodos de procesamiento (o tan solo nodos). Los variados elementos identificados –redes, procesos, tareas y objetos– requieren ser mapeados sobre los variados nodos. Esperamos que diferentes configuraciones puedan usarse: algunas para desarrollo y pruebas, otras para emplazar el sistema en varios sitios para distintos usuarios. Por lo tanto, el mapeo del software en los nodos requiere ser altamente flexible y tener un impacto mínimo sobre el código fuente en sí.

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.

Ver en Anexos Ilustración 15

3.4.3.1 Descripción de los Nodos Físicos

Nota: Se modeló la base de datos de ambas entidades separadas de los nodos del servidor de aplicación para mejor comprensión del diagrama. La base de

datos conjunto con el servidor de aplicación estarán en ambas entidades en una sola máquina (dígase PC dedicada a trabajar como servidor o un servidor en sí).

Nodo Servidor Web

Ver en Anexos Ilustración 16

Requerimientos de Hardware

- ❖ Pentium IV o superior
- ❖ 1 Giga de memoria RAM
- ❖ 2 Gigas de espacio en disco a ocupar

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.

Nodo Servidor de Bases de Datos (Replica)

Ver en Anexos Ilustración 17

Requerimientos de Hardware

- ❖ Procesador Pentium 3 (o superior)
- ❖ 256 o superior de Memoria RAM
- ❖ Almacenamiento en discos SCSI en espejo y capacidad igual o superior a los 80 Gigas cada disco. Tecnología de respaldo de datos históricos.

Subsistemas de Implementación.

Siguiendo una estrategia de réplicas de la base de datos en este nodo estará ejecutándose uno de los servidores PostgreSQL. La lógica del tratamiento de los datos no se implementará aquí sino en el servidor Web en la misma aplicación.

Nodo Servidor de Bases de Datos (Central)

Ver en Anexos Ilustración 18

Requerimientos de Hardware

- ❖ Procesador Pentium 3 (o superior)
- ❖ 256 o superior de Memoria RAM
- ❖ Almacenamiento en discos SCSI en espejo y capacidad igual o superior a los 160 Gigas cada disco. Tecnología de respaldo de datos históricos.

Subsistemas de Implementación.

Siguiendo una estrategia de de réplicas de la base de datos en este nodo estará ejecutándose el servidor central PostgreSQL. La lógica del tratamiento de los datos no se implementará aquí sino en el servidor Web en la misma aplicación.

Nodo Cliente Pozo

Ver en Anexos Ilustración 19

Requerimientos de Hardware

- ❖ Procesador Pentium 3 (o superior)
- ❖ 256 o superior de Memoria RAM
- ❖ Sistema Operativo con un navegador Web compatible con HTML 2.0 y CSS

Subsistemas de Implementación.

Las estaciones de trabajo donde operara la aplicación deben contar con navegadores con soporte HTML y java script.

Nodo Cliente DIPP

Ver en Anexos Ilustración 20

Requerimientos de Hardware

- ❖ Procesador Pentium 3 (o superior)
- ❖ 256 o superior de Memoria RAM

- Sistema Operativo con un navegador Web compatible con HTML 2.0 y CSS

Subsistemas de Implementación.

Las estaciones de trabajo donde operara la aplicación deben contar con navegadores con soporte HTML y java script.

3.4.3.2 Descripción elementos e interfaces de comunicación:

Http *(***HyperText Transfer Protocol**): Es un protocolo de transferencia de hipertexto es el protocolo usado en cada transacción de la Web (WWW). El hipertexto es el contenido de las páginas Web, y el protocolo de transferencia es el sistema mediante el cual se envían las peticiones de acceso a una página y la respuesta con el contenido. También sirve el protocolo para enviar 'información adicional en ambos sentidos, como formularios con campos de texto. HTTP es un protocolo sin estado, es decir, que no guarda ninguna información sobre conexiones anteriores.

TCP/IP: Es un protocolo DARPA que proporciona transmisión fiable de paquetes de datos sobre redes. El nombre TCP / IP Proviene de dos protocolos importantes de la familia, el Transmission Control Protocol (TCP) y el Internet Protocol (IP). Todos juntos llegan a ser más de 100 protocolos diferentes definidos en este conjunto.

El TCP / IP es la base del Internet que sirve para enlazar computadoras que utilizan diferentes sistemas operativos, incluyendo PC, minicomputadoras y computadoras centrales sobre redes de área local y área extensa. TCP / IP fue desarrollado y demostrado por primera vez en 1972 por el departamento de defensa de los Estados Unidos, ejecutándolo en el ARPANET una red de área extensa del departamento de defensa.

Cliente /Servidor

Como servidor Web se utiliza Apache 2, por sus características de Software Libre y compatibilidad con el lenguaje de desarrollo utilizado, PHP.El cliente podrá ser Netscape 3 (o superior), Mozilla 2.0, Internet Explorer 4.2 (o superior) y compatibles; ejecutándose tanto en Sistema Operativo Windows como Linux.

3.4.4 Vista de Implementación

La vista de implementación describe cómo se implementan los componentes físicos mostrados en vista de distribución agrupándolos en subsistemas organizados en capas, ilustra, además las dependencias entre éstos. Básicamente, se describe el mapeo desde los paquetes y clases del modelo de diseño a subsistemas y componentes físicos.

En el siguiente diagrama se muestra la distribución de los componentes comunes que integran el sistema:

Ver en Anexos Ilustración 21

3.4.5 Vista de Datos

El sistema cuenta con 39 tablas que permiten la persistencia de datos para los módulos de forma común. Todas las tablas son arquitectónicamente significativas para el sistema.

Ver en Anexos Ilustración 22

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 Symfony nos ofrece facilidades en el manejo de los datos, además toma lo mejor de la arquitectura MVC y del lenguaje PHP 5.

3.5 Requerimientos No Funcionales

El tratamiento de los distintos requisitos de calidad es explicado en el documento de arquitectura, aunque mencionaremos continuación cada uno de ellos y una breve descripción de los mimos enfocados a SIPP.

3.5.1 Usabilidad

- Preparar a los Administradores en la gestión de Roles y Permisos.

Esta acción específica del sistema lleva un nivel medio de complejidad, se debe preparar un curso para instruir a los que trabajaran con estas funcionalidades.

- Preparar a los usuarios del sistema en la gestión de los reportes.

Esta acción específica del sistema lleva una complejidad media, se debe preparar una instrucción referente al manual de usuario que acompañará a la aplicación para instruir a los que trabajarán con estas funcionalidades.

- Visibilidad del estado del sistema.

La aplicación debe mantener siempre informado al usuario del estado del sistema así como de los caminos que este pueda tomar con una retroalimentación visual apropiada en un tiempo razonable. El sistema ofrecerá al usuario una respuesta que le indique lo que está sucediendo en cada una de las operaciones que realiza.

- Consistencia y estándares.

Una buena interfaz contribuye al aumento de la productividad si es consistente en todos los diálogos que desarrolla, basándose en el conocimiento que el usuario ha adquirido con otras aplicaciones y en la aplicación propia. Se debe mantener la consistencia en todas las aplicaciones relacionadas. Deberán implementar las mismas reglas de diseño para mantener la consistencia en toda la interacción. El usuario debe ser capaz de saber en cada momento en qué

contexto está trabajando, de donde viene y a donde va. Esto se puede realizar con indicadores gráficos como iconos o colores diferentes para cada situación.

- Prevención de errores.

El mejor tratamiento de los errores es prevenirlos con un buen diseño de los diálogos desde el primer momento en que ocurren, minimizando los riesgos de que puedan ocurrir. Se debe realizar un buen diseño de mensajes de error que den la posibilidad al usuario de retraerse antes de que se realice la acción y se comprometan los datos.

- Correspondencia entre el sistema y el mundo real.

El sistema debe hablar el lenguaje de los usuarios, con palabras, frases y conceptos familiares para el usuario, siempre en el contexto de la aplicación. Se debe hacer que la información aparezca en un orden lógico y natural. El usuario no tiene por qué conocer los términos técnicos utilizados en el mundo informático. La aplicación debe interactuar con el usuario de forma que este perciba las palabras y frases cotidianas de la metáfora de la aplicación.

- Flexibilidad y eficiencia de uso.

El sistema se debe diseñar para que lo puedan manejar diferentes tipos de usuarios, en función de su experiencia con la aplicación. De esta manera se aumentará la productividad del usuario y se ganará en usabilidad. Una innovación muy importante en el desarrollo de interfaz de usuario es proporcionar al sistema técnicas de adaptabilidad. De esta forma la interfaz se adecua de forma automática a las características del usuario. Un sistema adaptativo puede estar basado en la experiencia del usuario con la aplicación o en la observación de las tareas que el usuario realiza repetidamente. Establecer aceleradores para aumentar la velocidad de la interacción para los usuarios expertos. Esto se puede realizar con pulsaciones de teclas rápidas, iconos,…etc. El usuario deberá poder

adaptar la interfaz a su conveniencia, por ejemplo, tendrá la posibilidad de quitar o añadir iconos, menús o barras de iconos.

- Ayuda y documentación.

El mejor sistema es el que no necesita ningún tipo de documentación, pero de todas formas hay que proporcionar al usuario ayuda y documentación. Esta debe ser fácil de encontrar y enfocada a la tarea que el usuario realiza. Se deben listar sólo los pasos necesarios para la realización de la tarea.

3.5.2 Rendimiento

- Tiempo de respuesta de transacción (menos de 3 segundos).

Teniendo en cuenta que el sistema no es de tiempo real y se plantea este tiempo como un tiempo máximo promedio estimado, el cual debe ser mucho menor; aunque si se alcanza valor no afectará el funcionamiento del sistema. Además las consultas a la base de datos no serán de un nivel de complejidad alto que influya en el rendimiento de la aplicación así como los niveles de concurrencia de usuarios que accedan al servidor de aplicaciones serán bajos.

- El sistema necesita las PC clientes con 256 o superior de RAM y un Navegador que soporte Java script.

Hay funcionalidades que están desarrolladas en java script por lo que es imprescindible que las PC clientes lo soporten.

- El sistema debe permitir trabajar conectados concurrentemente 250 usuarios como mínimo.

La cantidad de usuarios que se encuentran en los pozos no sobrepasan los 6, pero en Ceinpet y la DIPP pueden interactuar con la aplicación un número de usuarios que no sobrepasa la cifra antes expuestas.

- El sistema necesita un servidor de base de datos en una PC o servidor con (1 o superior) gigas de RAM y 2 discos duras de 160Gigas.

Para ganar en velocidad a la hora del trabajo con datos, además para el trabajo con los backup de respaldos de la base de datos.

3.5.3 Soporte

- El soporte y/o mantenimiento del sitio no debe detener el servicio

No debe existir ningún problema técnico en las PC que utilizarán los usuarios ya que traería problemas en el funcionamiento de la aplicación por lo que es necesario su chequeo y revisión diaria.

3.5.4 Restricciones de diseño

- El framework tiene en cuenta las posibles restricciones de diseño para la aplicación

Estos son: ajuste a estándares (una determinada manera de codificar un dato), limitaciones hardware (por los equipos disponibles), seguridad (por los distintos niveles de acceso a la información que deben tener los usuarios), mantenimiento (se debe tener en cuenta la ampliación del sistema), adaptación al entorno y políticas de borrado

- El diseño para la aplicación será de acuerdo a los distintos paradigmas y adecuado con la arquitectura seleccionada.

El diseño de la aplicación se hará aplicando Programación Orientada a Objetos. Diseño e implementación de una arquitectura flexible, que permita la fácil integración o desintegración de componentes. La arquitectura debe soportar migrar la interfaz de usuario sin impactos considerables en re-implementación.

3.5.5 Adquisición de Componentes

- Esta gestión la realizará Cupet

Cupet se encargara de la adquisición de los componentes necesarios, dígase PC y servidores como soporte para la aplicación.

3.5.6 Interfaz

- Interfaz de interacción con el usuario.

Interfaces amigables, fáciles de interactuar con ellas.

3.5.7 Requerimientos de Hardware

Nota: En cuanto a los componentes solicitados a continuación de capacidad de disco duro, memoria RAM, etc., en vista a que se prevé un aumento en cuanto al número de pozos en perforación y este aspecto incide en un aumento de uso y rendimiento de la aplicación por ejemplo en la DIPP se han solicitados componentes para un rendimiento óptimo de la aplicación.

3.5.7.1 Servidor Web

- Hardware de la estación de trabajo del servidor Web, donde se ejecutará el sistema.
- Procesador Pentium 3 (o superior)
- 1 Giga Memoria RAM
- 2 Giga de espacio en disco a ocupar

3.5.7.2 Servidor de Bases de Datos

- Hardware de la estación de trabajo servidor de Base de Datos.
- Procesador Pentium 3 (o superior)
- 1 Giga Memoria RAM

- Almacenamiento en discos SCSI en espejo y capacidad igual o superior a los 160 Giga cada disco. Tecnología de respaldo de datos históricos.

Nota: en el caso de los pozos con 2 discos de 80 Gigas es óptimo.

3.4.7.3 PC Cliente

- Hardware de la estación de trabajo del cliente.
- Procesador Pentium 3 (o superior).
- 256 o superior Memoria RAM

3.4.7.4 Redes

- Redes de la estación de trabajo del cliente.

La red existente en las instalaciones debe ser capaz de soportar transacciones de paquetes de información de al menos 25 máquinas a la vez.

El tipo de cableado debe ser de par trenzado UTP con soporte de 100Mpbs.

Debe estar protegido contra fallos de corriente y conectividad, además de cronometrar los tiempos de realización de copias de seguridad. La transmisión se implementarán utilizando el protocolo TCP/IP que permite la recuperación de datos.

3.5.8 Requerimientos de Software

3.5.8.1 PC Cliente

- Software instalado en la estación de trabajo del cliente.
- 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), Mozilla 2.0, Internet Explorer 4.2 (o superior) y compatibles.

3.5.8.2 PC Servidor Web

- Software instalado en el Servidor Web.
- Servidor Web Apache 2.2.X o superior.

3.5.8.3 Servidor de Bases de Datos

- Software instalado en el Servidor de Base de datos.

Servidor de Bases de Datos PostgreSQL8.2.

3.5.9 Seguridad

- Seguridad del sistema (C.I.D).

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.
Disponibilidad: Los usuarios autorizados (autenticados por dominio y según su roll) se les garantizarán 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.

- Seguridad del sistema(otros aspectos)

La seguridad se tratará desde la fase de diseño del sistema. Se garantizará un fuerte tratamiento de excepciones. Parte de la seguridad corre por parte de los framework propuestos (Symfony)

Los usuarios de la BD solamente tendrán acceso a las tablas y Bases de Datos a las cuales se les designe según el rol que desempeña, no se utilizarán usuarios con privilegios administrativos para realizar las conexiones entre servidores.

3.6 Conclusiones Parciales

Se concluye que:

Se expone la propuesta de la Arquitectura de Software para el proyecto Sistema de Información en Pozos en Perforación para Empresas Petroleras, definiendo las 4+1 vistas y los requerimientos no funcionales que son muy importantes para la Arquitectura. También se muestra la estructura del equipo de desarrollo para una mejor organización del personal y comprensión del mismo. Además se definieron los módulos que conforman al Sistema de Información en Pozos en Perforación para Empresas Petroleras, de estos: sus responsabilidades y funciones.

CONCLUSIONES

La investigación realizada muestra que en el campo de la arquitectura de software no hay consenso generalizado en cuanto a los conceptos más importantes, ni a la forma de documentar la arquitectura. La descripción de la arquitectura no incluye información que sea sólo necesaria para validar o verificar la arquitectura. Por tanto no tiene casos o procedimientos de pruebas, y no incluye una vista del modelo de prueba. Ésta debe mantenerse actualizada a lo largo de la vida del sistema para reflejar los cambios y las adiciones que son relevantes para la arquitectura.

La propuesta de solución presentada en este trabajo ha permitido realizar una documentación adecuada de la arquitectura de software del sistema que se desea construir, la cual es comprendida por los miembros del equipo de desarrollo. La comprensión de la arquitectura por parte del equipo de trabajo muestra el cumplimiento del objetivo del presente trabajo, ya que permitió a los mismo el elaborar un sistema portable, con una interfaz sencilla y bien estructurada, haciéndola más usable y comprensible para el usuario,

garantizando la accesibilidad y seguridad de las informaciones y servicios contenidos en el mismo.

Recomendaciones

Se recomienda:

- Continuar la profundización del estudio de la arquitectura propuesta para SIPP.
- El refinamiento constante de la arquitectura propuesta, durante el ciclo de desarrollo.
- La revisión de cada uno de los escenarios de la aplicación para evaluar a fondo las restricciones que se le imponen a la arquitectura.
- El análisis de las dependencias entre los subsistemas de implementación para evitar que existan cuellos de botella en alguno de ellos y que se vea afectado en cuanto a tiempo la entrega de cada una de las actividades de implementación.
- Realizar una evaluación de los costes de la tecnología utilizada, aunque la misma en su totalidad es libre, es necesario evaluar los costes del hardware que la soportan.
- La aplicación de cualquiera de los métodos de evaluación de la arquitectura, con el objetivo de identificar las principales debilidades, tanto en la descripción arquitectónica como en el propio diseño arquitectónico

Referencia Bibliográfica

1. freedownloadmanager. *freedownloadmanager.* [Online] 02 24, 2009. [Cited: 11 06, 2008.] http://www.freedownloadmanager.org/es/downloads/Bien_Maderero_7024_p/. 1.

2. wellsight. *wellsight.* [Online] 02 24, 2009. [Cited: 11 06, 2008.] http://www.wellsight.com/. 2.

3. addlink. *addlink.* [Online] 02 24, 2009. [Cited: 11 06, 2008.] http://www.addlink.es/productos.asp?pid=430. 3.

4. Peloton. *Peloton.* [Online] 02 24, 2009. [Cited: 02 20, 2009.] http://www.peloton.com/es/. 4.

5. Wikipedia. *Wikipedia.* [Online] 02 24, 2009. [Cited: 12 05, 2008.] http://es.wikipedia.org/wiki/Arquitectura_de_software. 5.

6. BASS, L., CLEMENTS, P., & KAZMAN, R. Software Architecture in practice. Addison-Wesley., 1998.6

7. Pressman .*Ingenieria de Software.Un Enfoque Practico.* Madrid : Facultad de Informatica y Escuela Universitaria de Informatica, 1997. 7.

8. Shaw y Garlan [Cited: 12 05, 2008.] http://www.fing.edu.uy/inco/cursos/gestsoft/Presentaciones/Evaluacion%20de%20Arquitecturas%20-%20G10/Evaluacion%20de%20Arquitecturas.doc.8.

9. BUSCHMANN, F., MEUNIER, R., ROHNERT, H., SOMMERLAD, P., & STAL, M. Pattern – Oriented Software Architecture. A System of Patterns. John Wiley & Sons, Inglaterra, 1996.9.

10. UML y Patrones. Mexico : Craig Larman, 1999. 10.

11. Symfony La guia definitiva. Francia : Fabien Potencier, Octubre de 2005. 11.

12. KRUCHTEN., P. “The 4+1 View Model of Architecture.” IEEE Software, Noviembre de 1995.12.

13. Wikipedia. *Wikipedia.* [Online] 02 24, 2009. [Cited: 12 05, 2008.] http://es.wikipedia.org/wiki/RUP. 13.

14. Len Bass, Paul Clements y Rick Kazman. Software Architecture in Practice. Sei Series in Software Architectures. Addison Websley, 1998.14.

15. G.D. ABOWD, R. A. A. D. G. "Formalizing style to understand descriptions of software architecture". ACM Transactions on Software Engineering and Methodology, 1995.15.

16. Siona. *Siona.* [Online] 03 10, 2009. [Cited: 02 10, 2009.] http://siona.udea.edu.co/~aoviedo/Arquitectura%20de%20Software/adl.htm.16.

17. Jason E. Robbins, Nenad Medvidovic, David F. Redmiles y David S. Rosenblum. Integrating Architec-ture Description Languages with a Standard Design Method. 20th International Conference on Software.17.

18. Grady Booch, James Rumbaugh e Ivar Jacobson. The Unified Modeling Language User Guide. Rational Software Corporation. Addison-Wesley, 1999.18.

19. Visual Paradimg. [Cited: 12 06, 2008.] Disponible en: <http://www.visual-paradigm.com/>.19.

20. Sierra, María. Trabajando con Visual Paradimg for UML. Universidad de Cantambria Facultad Ciencias.20

21. (MAGICDRAW) [Cited: 12 06, 2008.]Disponible en: <http://www.magicdraw.com>.21.

22. Ventajas de PostgreSQL, 2003-05-31, Néstor A. Díaz L. [Cited: 02 10, 2009.]Disponible en: <http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html>.22.

23. Mato García, Rosa María. Sistemas de Base de Datos. Editorial Pueblo y Educación ,2005. pág. 3-4.23.

24. Lévenez, É. 2007. Computer Languajes History. [En línea] 2007. [Citado el: 2 de Febrero de 2009.] http://www.levenez.com.lang/.24.

25.Parlante, N. 2002**.** Essential Perl. [En línea] 2002. [Citado el: 15 de enero del2009.] http://cslibrary.stanford.edu/108/EssentialPerl.thml#aboutperl.25.

26.Alvarez, Miguel Angel. 2003**.** Qué es Python. [En línea] 19 de 11 de 2003. [Citado el: 15 de Enero de 2009.] http://www.desarrolloweb.com.articulos/1325.php.26.

27. Guedez, Yamila. 2006**.** Glosario de términos. [En línea] 2006. [Citado el: 2 de Febrero de 2009.] http://www.icad.com.ve/soporte/documentos/glosario.27.

28.S, Christian Van Der Henst. 2001. Qué es PHP. [En línea] 2001. [Citado el: 2 de Febrero de 2009.] http://www.maestrosdelweb.com/editorial/phpintro.28.

29.. Wikipedia. *Wikipedia.* [Online] 02 24, 2009. [Cited: 12 05, 2008.] http://es.wikipedia.org/wiki/Apache. 29.

30. Representación de la arquitectura de Software usando UML. Colombia : Universidad Icesi, 28-jul-2006 . 30.

Bibliografía

ANA M. MORENO, M. I. S. Patrones de Usabilidad: Mejora de la Usabilidad del Software desde el momento Arquitectónico.

BASTARRICA, M. C. Atributos de Calidad y Arquitectura del Software.

CÁRDENAS, S. Á. Diseño de la arquitectura y los servicios web.

ERIKA CAMACHO, F. C., GABRIEL NÚÑEZ Arquitecturas de Software, Abril 2004. Disponible en http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf

JOHN WORSLEY, J. D. and M. H. EDITADO POR ANDREW BROOKINS PostgreSQL Práctico, 2001. Disponible en http://www.sobl.org/traducciones/practical-postgres/node19.html

JOSEPH F. MARANZANO, S. A. R. A. G. H. Z. Architecture Reviews: Practice and Experience.

LASSO, A. Arquitectura de Software. Disponibilidad http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art110.asp

MARÍA L. RODRÍGUEZ, M. N., MIGUEL J. HORNOS, PATRICIA PADEREWSKI, JOÉ LUIS GARRIDO Hacia la satisfación de requisitos de la calidad de los sistemas colaborativos mediante un diseño arquitectónico.

PARRA, J. D. Hacia una Arquitectura Empresarial basada en Servicios. Disponible en http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art143.asp

REYNOSO, C. B. Introducción a la Arquitectura de Software, 2004. Disponible en http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp

WADOOA SOA (Service-oriented Architecture) Arquitectura Orientada a Servicios, 2007. Disponible en http://wadooa.com/doku.php/soa

S. E. I. (2000) Software Architecture.

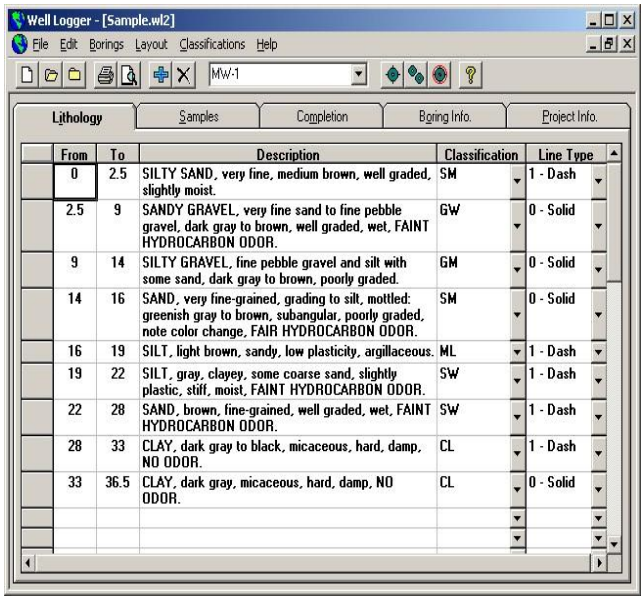
Bass, L., Clements, P., y Kazman, R. (1998) Software Architecture in Practice. Addison Wesley.

Bastarrica, M. C. (2005) Atributos de Calidad y Arquitectura del Software. Volumen 4

Booch, G. (1999). The Unified Modeling Language User Guide, Addison Wesley.

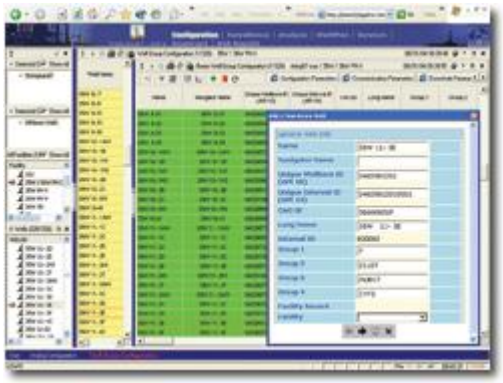
Bosch. J. (2000). Design & Use of Software Architecture". Addison Wesley.

Anexos



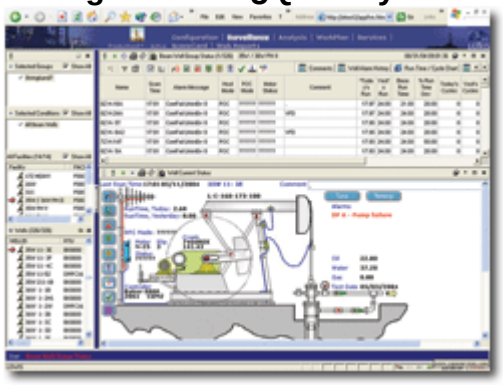
From	To	Description	Classification	Line Type
0	2.5	SILTY SAND, very fine, medium brown, well graded, slightly moist.	SM	1 - Dash
2.5	9	SANDY GRAVEL, very fine sand to fine pebble gravel, dark gray to brown, well graded, wet, FAINT HYDROCARBON ODOR.	GW	0 - Solid
9	14	SILTY GRAVEL, fine pebble gravel and silt with some sand, dark gray to brown, poorly graded.	GM	0 - Solid
14	16	SAND, very fine-grained, grading to silt, mottled: greenish gray to brown, subangular, poorly graded, note color change, FAIR HYDROCARBON ODOR.	SM	0 - Solid
16	19	SILT, light brown, sandy, low plasticity, argillaceous.	ML	1 - Dash
19	22	SILT, gray, clayey, some coarse sand, slightly plastic, stiff, moist, FAINT HYDROCARBON ODOR.	SW	1 - Dash
22	28	SAND, brown, fine-grained, well graded, wet, FAINT HYDROCARBON ODOR.	SW	1 - Dash
28	33	CLAY, dark gray to black, micaceous, hard, damp, NO ODOR.	CL	1 - Dash
33	36.5	CLAY, dark gray, micaceous, hard, damp, NO ODOR.	CL	0 - Solid

Software 1 Well Logger



Software 2 Well Sight

Configuración - ¿Qué hay en el campo?



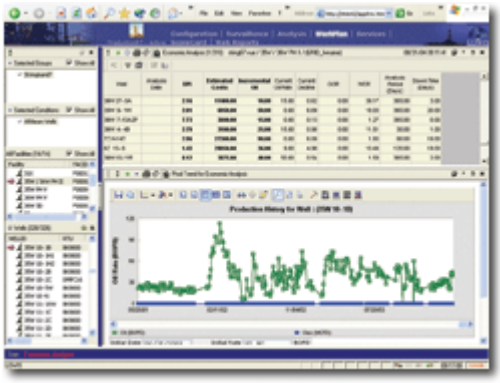
Software 3 Well Sight

Vigilancia - ¿Qué está pasando?

SIPP

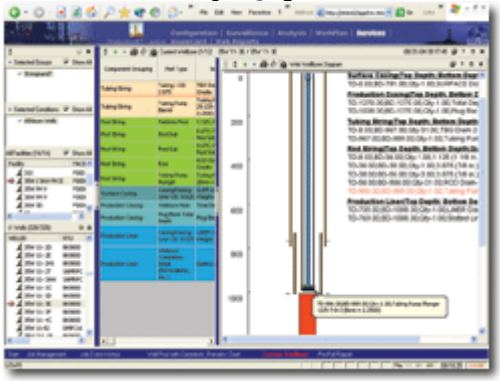
Software 4 Well Sight

Análisis-¿Está funcionando como se espera?



Software 5 Well Sight

Plan de Trabajo ¿Qué debería hacerse a continuación?



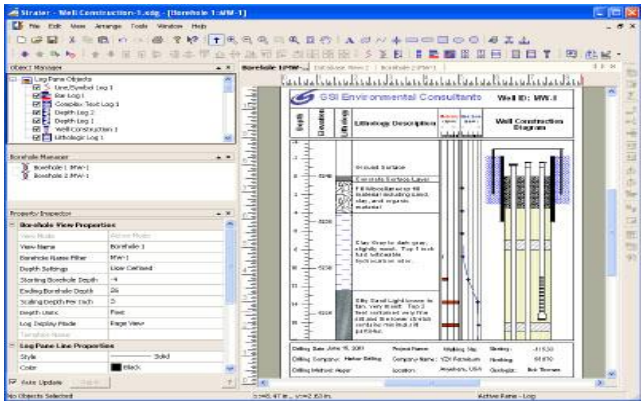
Software 6 Well Sight

Servicios - ¿Cómo deberían ser ejecutados?

SIPP

Software 7 Well Sight

Carta de Puntuación - ¿Qué tan bien se hizo?



Software 8 Strater

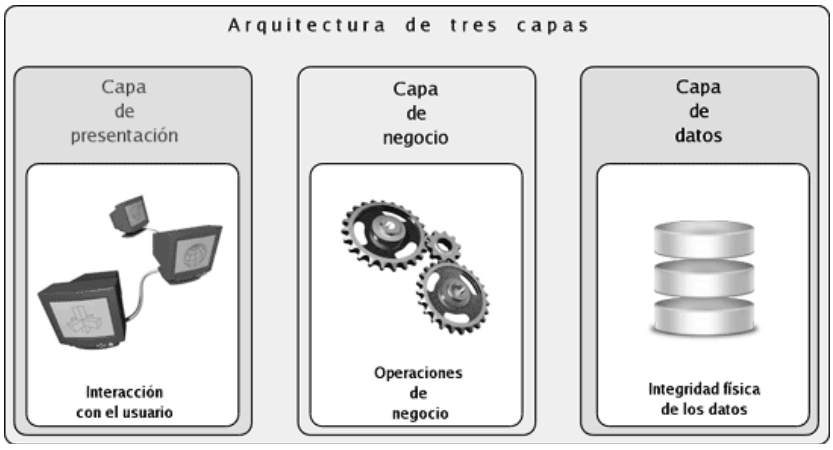


Ilustración 2 Arquitectura en Tres capas

Tabla 1 Diferencias entre Estilos y Patrones Arquitectónicos

Estilo Arquitectónico	Patrón Arquitectónico
Sólo describe el esqueleto estructural general para aplicaciones	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación
Son independientes del contexto al que puedan ser aplicados	Partiendo de la definición de *patrón*, requieren de la especificación de un contexto del problema
Cada estilo es independiente de los otros	Depende de patrones más pequeños que contiene, patrones con los que interactúa, o de patrones que lo contengan
Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual de diseño	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta
Son una categorización de sistemas	Son soluciones generales a problemas comunes

Tabla 2 Vistas 4+1 de RUP

Modelo 4+1	Arquitectura
Use – Case View	Vista de Caso de Uso, Restricciones, Calidad
Logical View	Vista Lógica
Process View	Vista de Procesos
Deployment View	Vista de Despliegue
Implementation View	Vista de Implementación

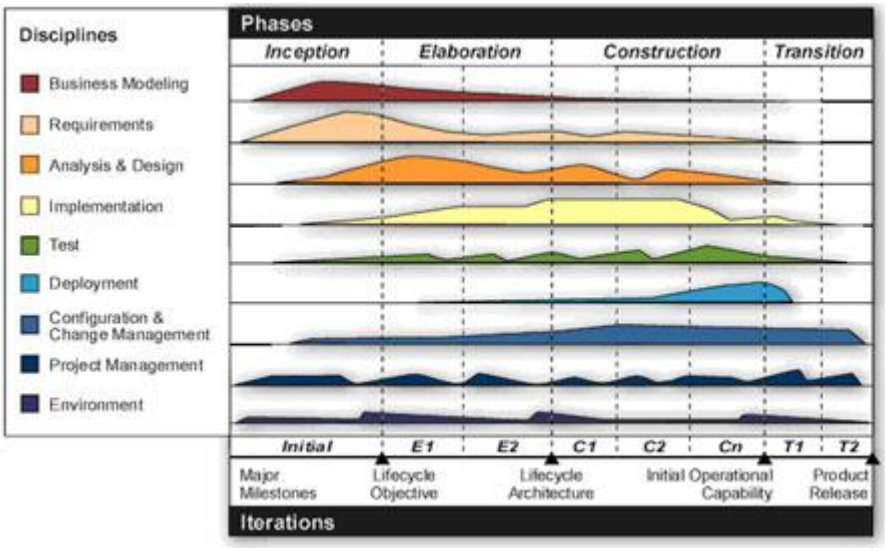


Ilustración 3 RUP

Tabla 3 Cronología de los distintos ADL

ADL	Fecha	Investigador - Organismo	Observaciones
Acme	1995	Monroe & Garlan (CMU), Wile (USC)	Lenguaje d'intercambio d'ADLs
Aesop	1994	Garlan (CMU)	ADL de propósito general, énfasis en estilos
ArTek	1994	Terry, Hayes-Roth, Erman (Teknowledge, DSSA)	Lenguaje específico de dominio - No es ADL
Armani	1998	Monroe (CMU)	ADL asociado a Acme
C2 SADL	1996	Taylor/Medvidovic (UCI)	ADL específico d'estilo
CHAM	1990	Berry / Boudol	Lenguaje d'especificación
Darwin	1991	Magee, Dulay, Eisenbach, Kramer	ADL con énfasis en dinámica
Jacal	1997	Kicillof, Yankelevich (Universidad de Buenos Aires)	ADL - Notación de alto nivel para descripción y prototipado
LILEANNA	1993	Tracz (Loral Federal)	Lenguaje de conexión de módulos
MetaH	1993	Binns, Englehart	ADL específico de dominio

		(Honeywell)	
Rapide	1990	Luckham (Stanford)	ADL & simulación
SADL	1995	Moriconi, Riemenschneider (SRI)	ADL con énfasis en mapeo de refinamiento
UML	1995	Rumbaugh, Jacobson, Booch (Rational)	Lenguaje genérico de modelado – No es ADL
UniCon	1995	Shaw (CMU)	ADL de propósito general, énfasis en conectores y estilos
Wright	1994	Garlan (CMU)	ADL de propósito general, énfasis en comunicación
xADL	2000	Medvidovic, Taylor (UCI, UCLA)	ADL basado en XML

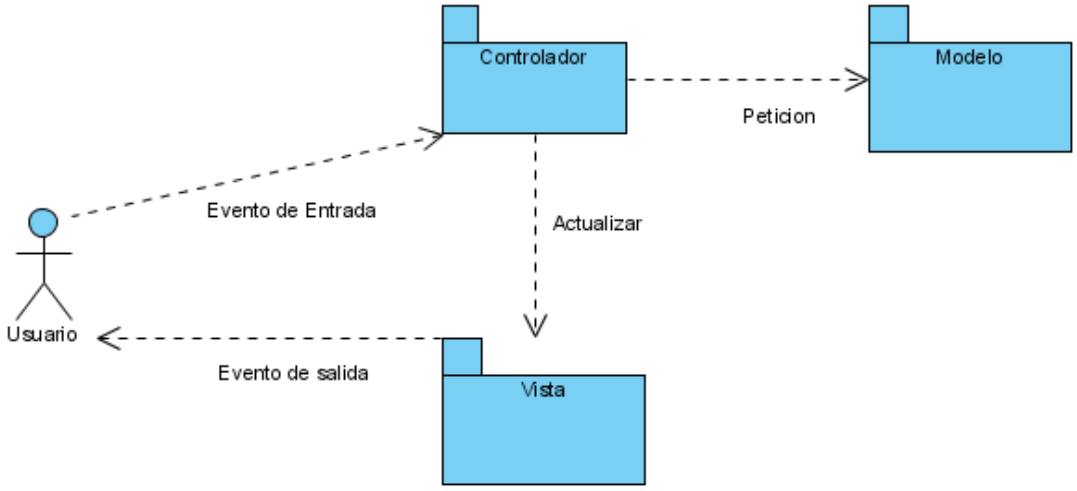


Ilustración 4 Flujo de mensajes del patrón MVC

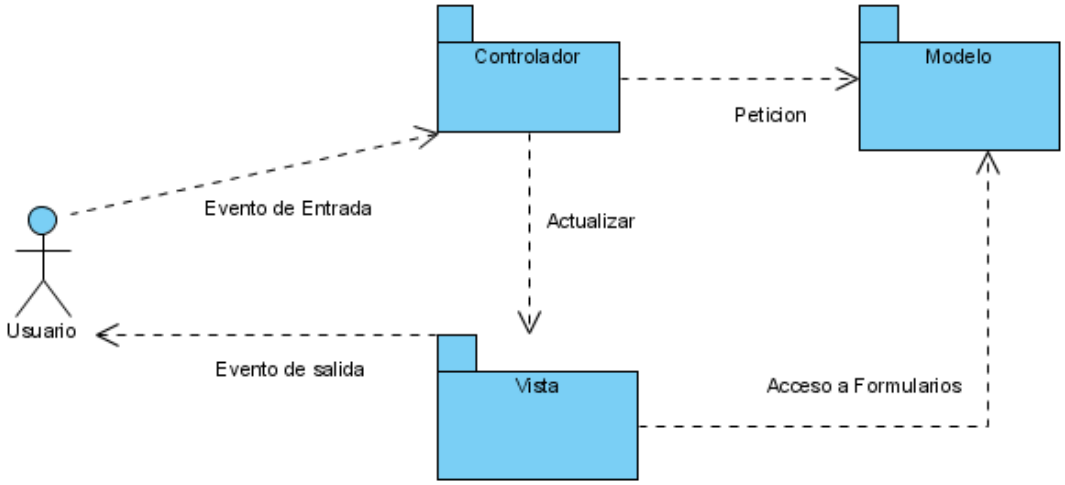


Ilustración 5 Flujo de mensajes del patrón MVC2.

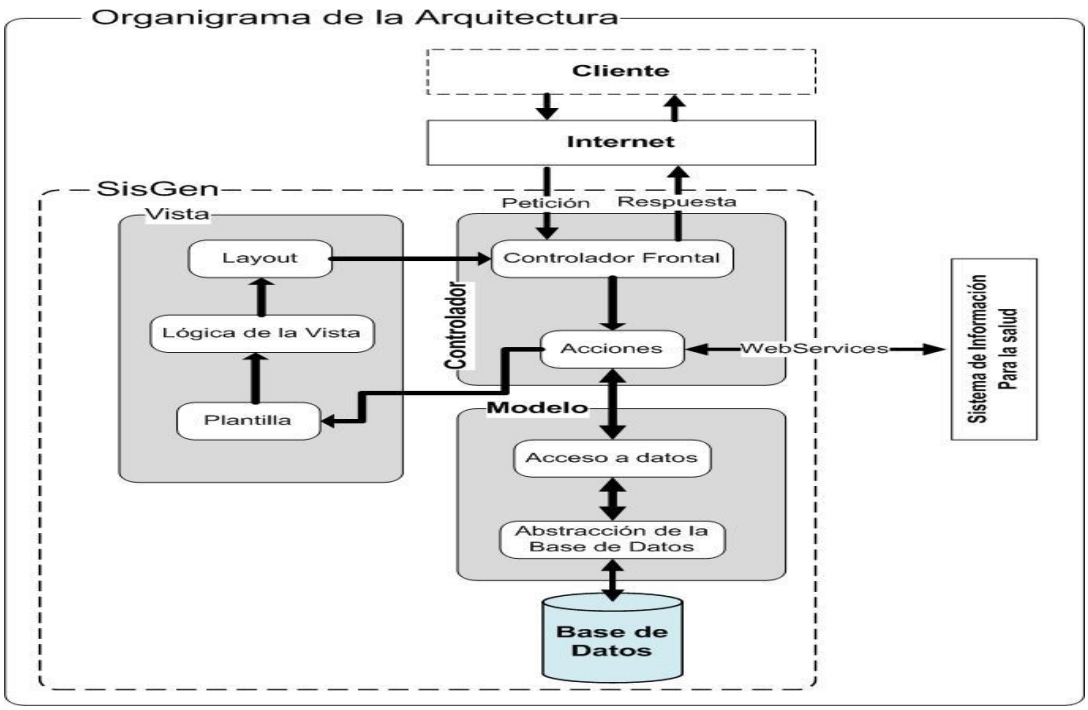


Ilustración 6 Organigrama de la Arquitectura

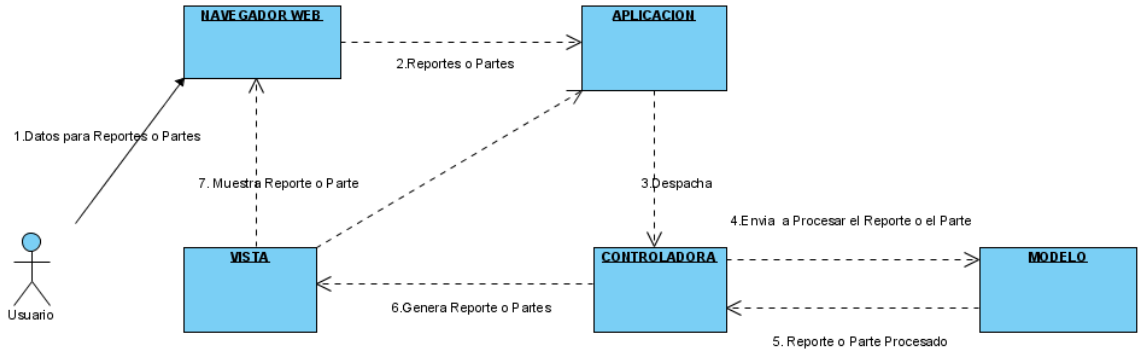


Ilustración 7 Estructura del paso de mensaje entre las capas vista, Modelo y Controlador

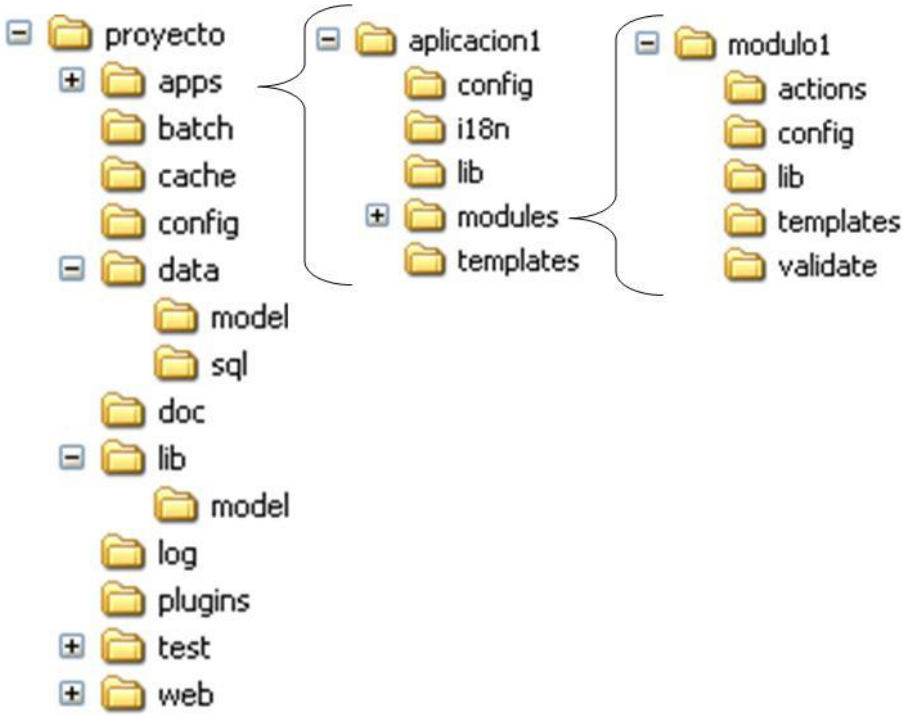


Ilustración 8 Estructura del Framework Symfony

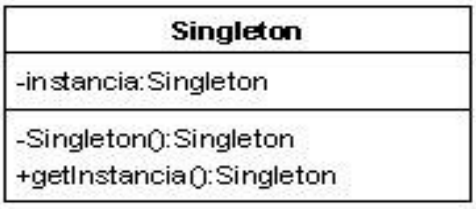


Ilustración 9 Estructura del patrón de diseño Singleton

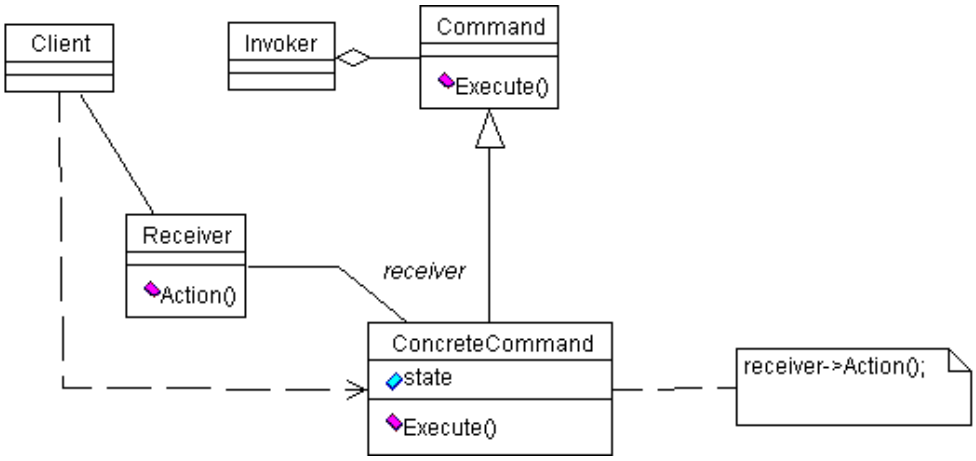


Ilustración 10 Estructura del patrón de diseño Command

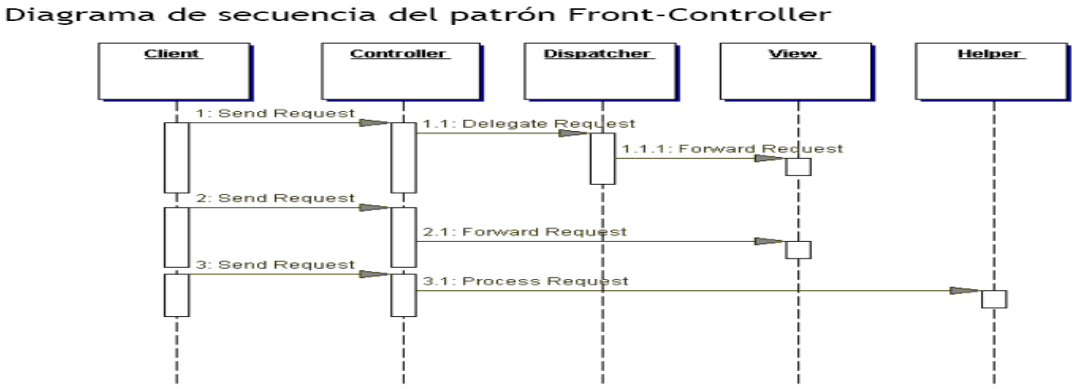


Ilustración 11 Diagrama de Secuencia del patrón Front Controller

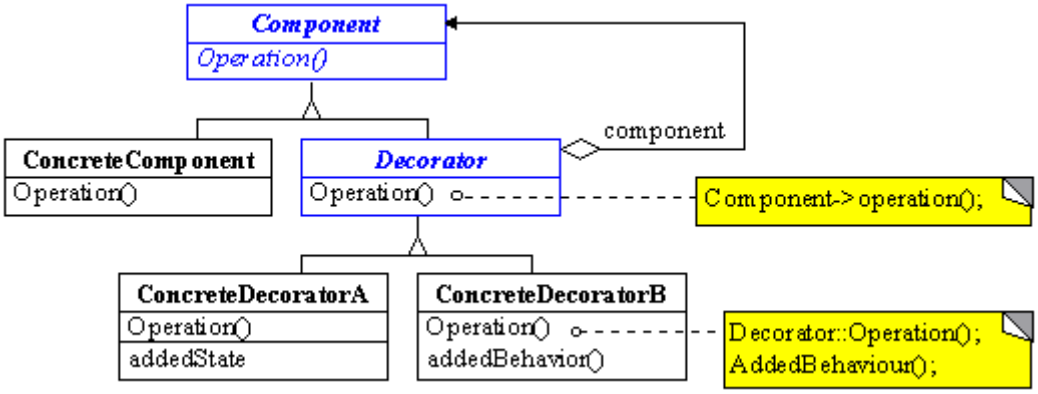


Ilustración 12 Estructura del patrón de diseño Decorator

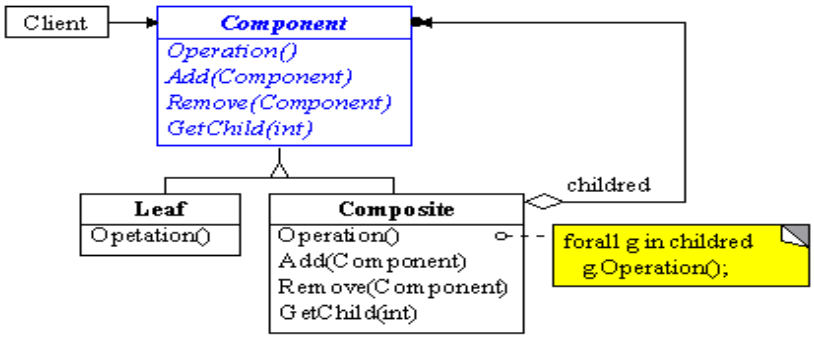


Ilustración 13 Estructura del patrón de diseño Composite

Tabla 4 Integrantes del Proyecto

Nombre	Rol
Dennis Meriño Menadier	Líder de Proyecto
Yaquelin Cintra Almaguer	Segunda Jefa de Proyecto
David Tavares Cuevas	Analista Principal
Michel Buzón Tur	Arquitecto.
Maikel Hugo Álvarez González	Diseñador(Base de Datos)
David Rodríguez Lagrana	Jefe de Desarrollo (Implementador)
Yordan Gallardo Avilés	Diseñador(Base de Datos)
Zenia María Rodríguez López	Programador(Base de Datos)
Yelina Hernández Plasencia	Analista
Reynel Gómez Martínez	Analista
Roseli Lemes Acosta	Analista
Noel Pérez Bello	Programador
Camilo Nelson	Programador(Base de Datos)
Daniel Torres Machado	Programador
Eduardo Ramírez Cruz	Programador
Milton León Rodríguez	Analista
Aniuvys Reyes Sifontes	Programador.
Alberto Esteban Rosales Ramírez	Programador.
Amirka Palacios	Programador.

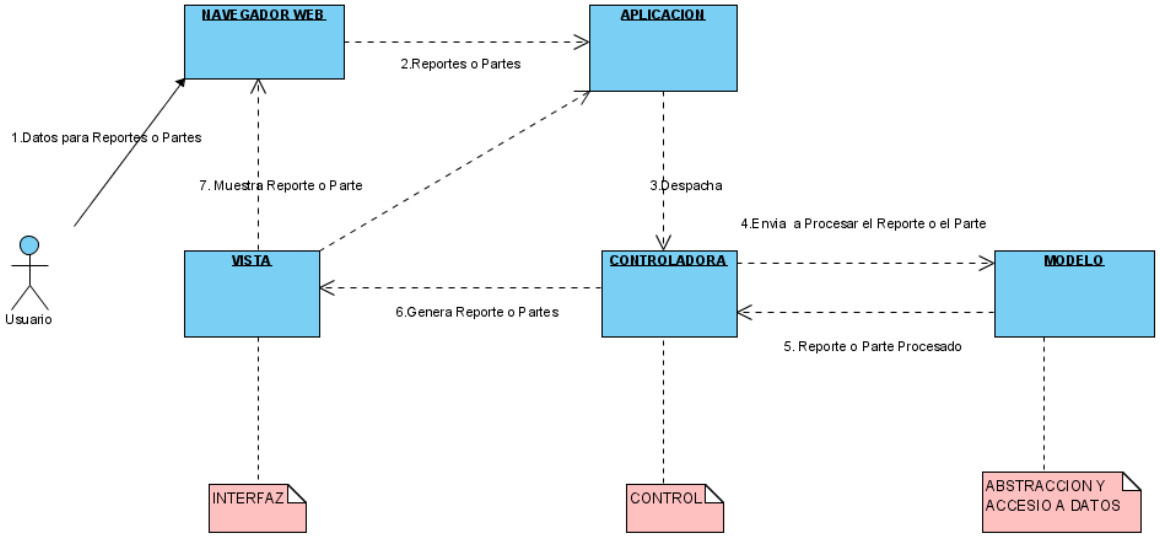


Ilustración 14 Estructura de capas del patrón MVC aplicado al Sistema

Tabla 5 Tecnologías que se emplearan en cada componente:

Área	PHP	Symfony	Java script	PostgreSQL
Interfaz	X	X	X	
Control	X	X		
Dominio Acceso a Datos	X	X		X

Tabla 6 Cantidad de Casos de Uso del sistema por módulos.

Módulos	Cantidad de Casos de Uso
Pozo	22
DIPP	7
Visualización	4
Trabajo con Archivos	1
Seguridad	4

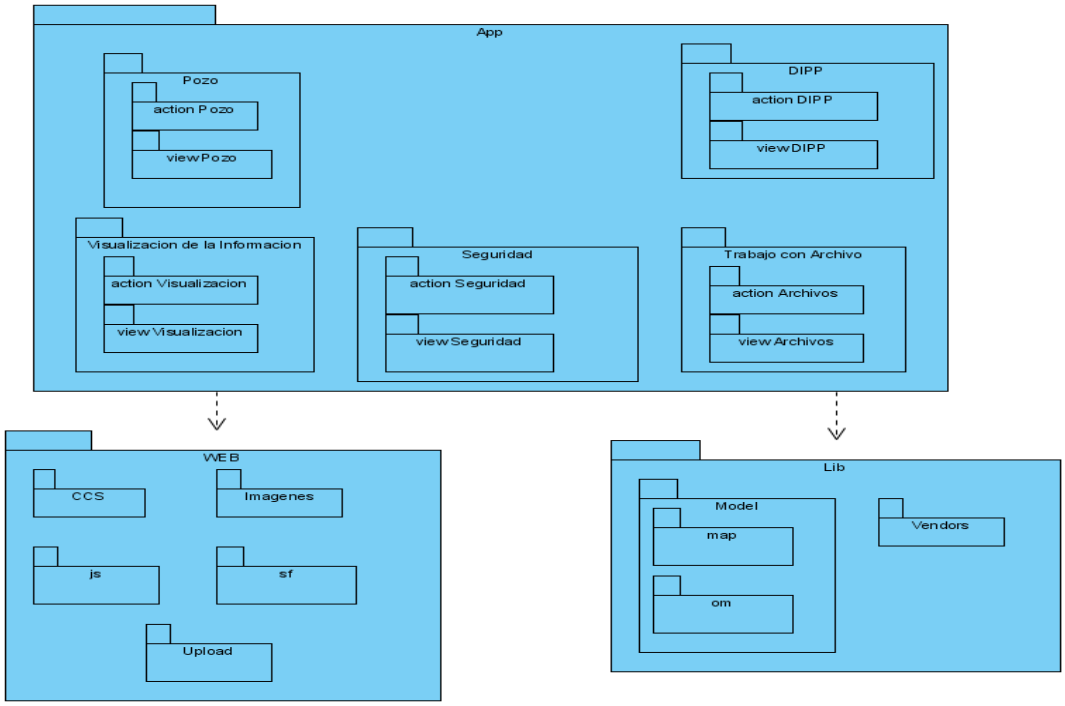


Ilustración 15 Vista lógica del sistema

Tabla 7 Descripción de los paquetes del sistema

Paquete	Descripción
App	El paquete contiene los distintos módulos que componen el sistema
Pozo	El paquete contiene las clases Interfaz (vistas) y controladoras del módulo pozo.
DIPP	El paquete contiene las clases Interfaz (vistas) y controladoras del módulo DIPP.
Visualización de la informaci ón	El paquete contiene las clases Interfaz (vistas) y controladoras del módulo Visualización de la información.
Seguridad	El paquete contiene las clases Interfaz (vistas) y controladoras del módulo Seguridad.
Trabajo	El paquete contiene las clases Interfaz (vistas) y

con archivo	controladoras del módulo Trabajo con archivo.

Paquete	Descripción
WEB	El paquete contiene los archivos de apoyo necesarios para el diseño gráfico de la interfaz de la aplicación web.
CSS	El paquete contiene los archivos de hojas de estilo creados con CSS (archivos con extensión .css)
Imágenes	El paquete contiene las imágenes del sitio con formato .jpg, .png o .gif)
js	El paquete contiene los archivos de Javascript con extensión .js
upload	En el paquete se pueden almacenar los archivos subidos por los usuarios. Aunque normalmente este directorio contiene imágenes, no se debe confundir con el directorio que almacena las imágenes del sitio (images/). Esta distinción permite sincronizar los servidores de desarrollo y de producción sin afectar a las imágenes subidas por los usuarios

Paquete	Descripción
LIB	El paquete contiene las clases y librerías externas. Se suele guardar todo el código común a todas las aplicaciones del proyecto
Model	El paquete contiene las clases Modelo del sistema
map	El paquete contiene las clases resultantes del mapeo de datos a la base de datos
om	El paquete contiene las clases que contienen los get y set para acceder a las clases obtenidas del mapeo de datos
Vendors	El paquete contiene las clases definidas por terceros y que son usadas de apoyo por el sistema

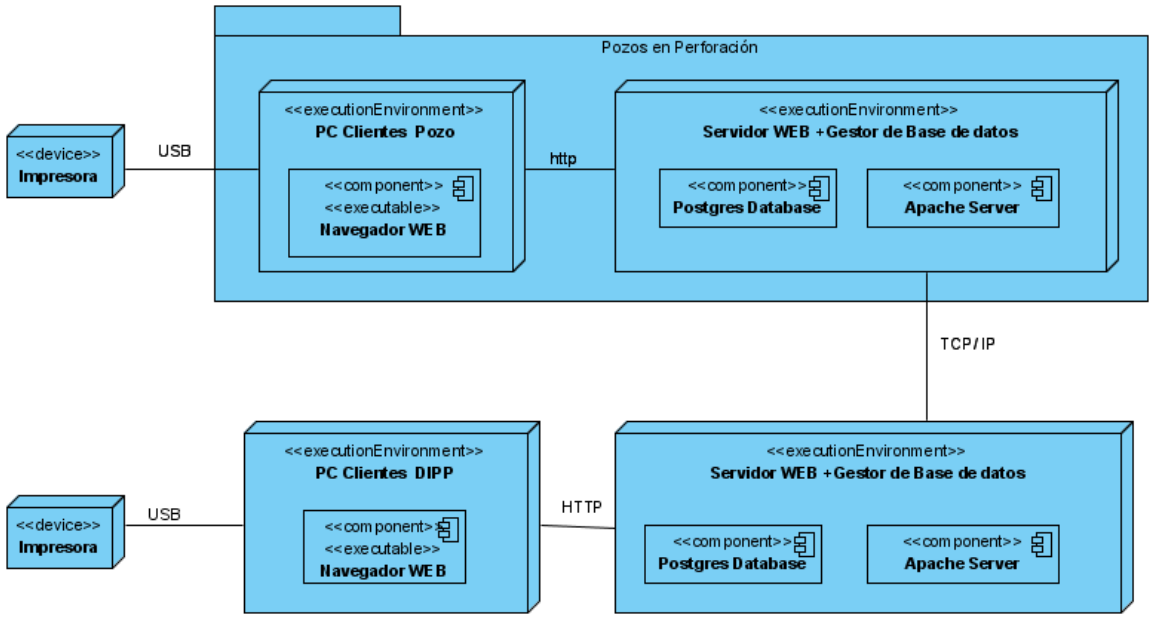


Ilustración 16 Diagrama de despliegue

Ilustración 17 Servidor Web.

Ilustración 18 Servidor de Bases de Datos (Pozo).

Ilustración 19 Servidor de Bases de Datos (Unión).

Ilustración 20 PC Clientes Pozo.

Ilustración 21 PC Clientes DIPP.

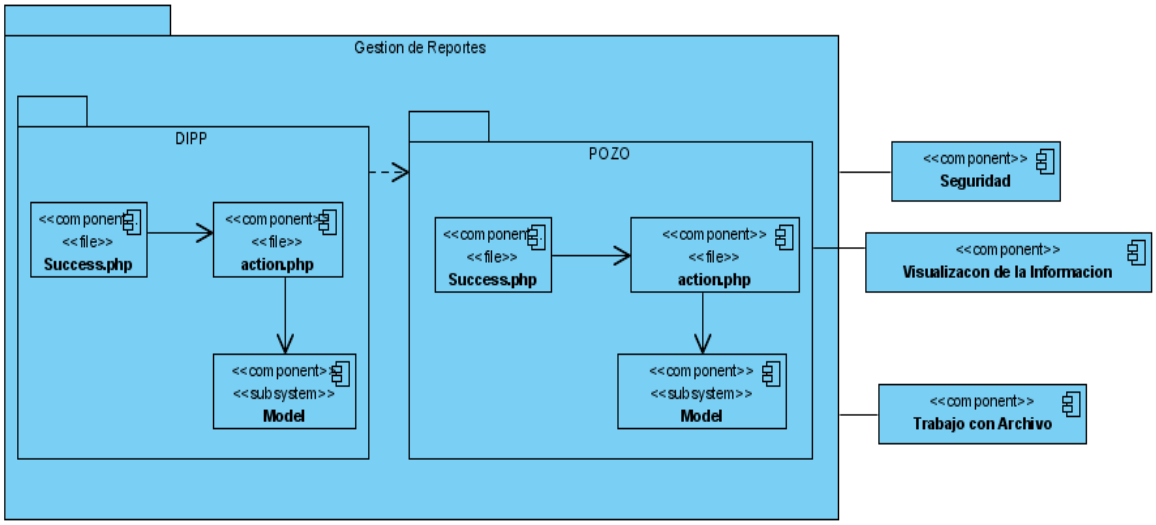


Ilustración 22 Vista de Implementación del sistema.

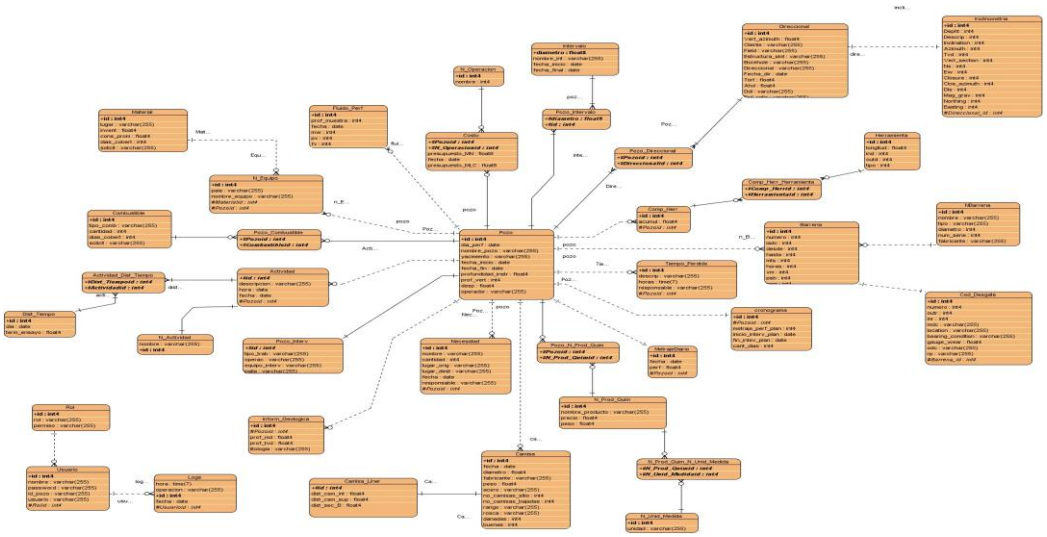


Ilustración 23 Modelación de la BD del sistema

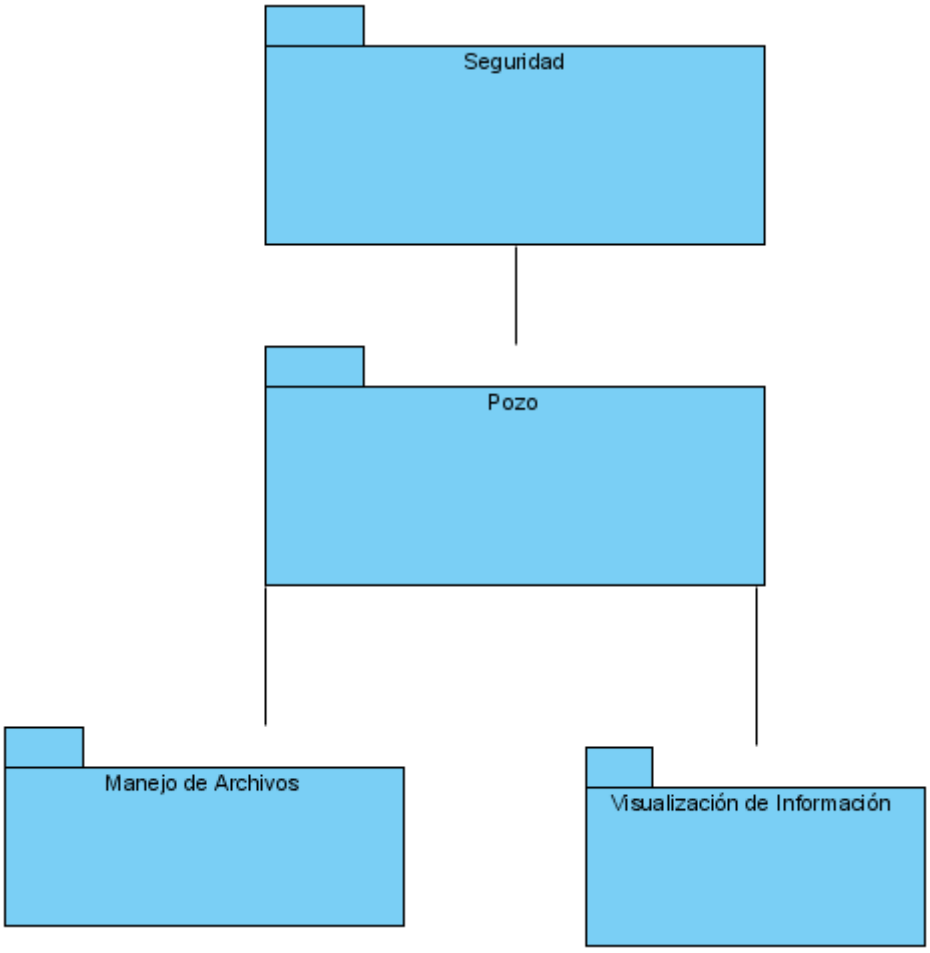


Ilustración 24 Vista de Caso de Uso

Glosario de Términos

- Estilo CAD: Estilo para el tratamiento de datos gráficos.
- WellSight: Wellsite Information transfer standard markup Language (Lenguaje para marcar normas para transferir información del sitio del pozo)
- DST: Genera informes relacionados con la Dirección de Servicios Tecnológicos
- RQD: índices RQD (*Rock Quality Designation*) se define como el porcentaje de recuperación de testigos de más de 10 cm de longitud (en su eje) sin tener en cuenta las roturas frescas del proceso de perforación respecto de la longitud total del sondeo. Además saber que para determinar el RQD en el campo o zona de estudio de una operación minera, existen tres procedimientos de cálculo.
- ADLs: Lenguajes de Definición de Arquitecturas
- RUP: Proceso Racional Unificado
- MVC: Modelo-Vista-Controladora
- Framework: Entorno de Trabajo
- DIPP: Dirección de Intervención y Perforación de Pozos.
- CUPET: Empresa Cuba Petróleo.
- CEINPET: Centro de Investigaciones del Petróleo
- MINBAS: Ministerio de la Industria Básica
- ORM: Mapeo Objeto Relacional (Capa de Abstracción de la Base de datos)