UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 09

TÍTULO: *Arquitectura de la Plataforma de Servicios a Pozos para el CEINPET*

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE

INGENIERO EN CIENCIAS INFORMÁTICAS

AUTOR: *Reinier Perez Mirabal*

TUTOR: *Lic. Yesnier Bravo García*

Ciudad de La Habana, julio 2010.

"Año 52 de la Revolución"

DECLARACIÓN DE AUTORÍA

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

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

Reinier Perez Mirabal

Lic. Yesnier García Bravo

_____

_____

RESUMEN

La investigación surge en el marco de trabajo de las aplicaciones desarrolladas por el Polo Productivo Petrosoft de la Universidad de las Ciencias Informáticas (UCI) para el Centro de Investigaciones del Petróleo (CEINPET). Este centro plantea la necesidad de integrar estas aplicaciones en un sistema para tenerlas centralizadas, por lo que se decidió diseñar una arquitectura para implementar una plataforma que cumpla este objetivo. La arquitectura diseñada proporcionará los elementos necesarios para el desarrollo de una plataforma que brinde a los especialistas del centro un entorno de trabajo provisto de las funcionalidades necesarias para la evolución de sus investigaciones.

Actualmente este proyecto pertenece al Centro de Desarrollo de Informática Industrial (CEDIN) de la propia UCI; el cual se encuentra en los procesos de re implementación de estas aplicaciones con nuevas tecnologías y herramientas para dar solución a las necesidades del cliente. Al término de esta tarea se podrá lograr la integración antes planteada siguiendo la arquitectura propuesta para este fin con este trabajo. En este documento se describe y detalla la propuesta de herramientas y tecnologías a utilizarse; así como los estilos arquitectónicos y patrones de diseño a seguir para darle solución al problema. También se generan las vistas de la arquitectura y es evaluada la solución mediante un método para la evaluación de arquitecturas.

Se espera con este documento crear la línea base de la arquitectura para facilitar al equipo de desarrollo una guía en el desarrollo de la plataforma de servicios a pozos.

PALABRAS CLAVES

Arquitectura de software, arquitectura orientada a componentes, herramienta, tecnología, estilo, patrón, plataforma, componente, aplicación, petróleo, evaluación de la arquitectura de software.

INTRODUCCIÓN

La industria petrolera se ha convertido en uno de los pilares principales de la economía mundial dada la creciente necesidad de recursos energéticos, en particular de hidrocarburos por parte de la humanidad. El petróleo actualmente es considerado el recurso natural no renovable que mayor porcentaje energético aporta al consumo mundial. La industria del petróleo a nivel internacional hace algunos años viene destinando esfuerzos y recursos para mejorar los índices de producción en los yacimientos del crudo.

El Centro de Investigaciones del Petróleo (CEINPET) no está exento de estas tareas, es una institución que por más de 10 años se ha encargado de las investigaciones que se realizan en la empresa cubana del petróleo, siendo el único de su tipo en el país. Diversas son las actividades investigativas que en este se realizan, la mayoría de ellas con suma importancia para el desarrollo de la industria petrolera en Cuba.

Este centro está enfrascado en el empeño de sustituir los procedimientos tradicionales de manipulación y control de la información, que se realizan manualmente, por métodos automatizados de almacenamiento. Estos proveen un ambiente de trabajo para la toma de decisiones y el manejo de resultados. Estas ventajas y el auge de las aplicaciones informáticas que gestionan información, cada vez más utilizadas por las instituciones del país son la razón de su empeño.

La Universidad de las Ciencias Informáticas (UCI) desarrolló para ellos un conjunto de aplicaciones que sustituyeron el engorroso trabajo de revisar por horas el contenido de inmensas tablas Excel por un conjunto de especialistas. Para ahora contar con aplicaciones que automatizan algoritmos guiados por fórmulas y leyes matemáticas, entre otras ciencias. Solo que ahora desean integrar estas aplicaciones en una plataforma de servicios para lograr organización, tener todas estas aplicaciones centralizadas, que se aprovechen procesos que se realizan comúnmente entre ellas y que su a vez soporte futuros desarrollos para la empresa.

Esta plataforma permitirá que se compartan procesos de las diferentes aplicaciones, logrando así que el almacenamiento y manejo de los datos sea centralizado y agilizar todo el proceso de obtención de resultados. Además de lograr que estén integrados en un solo sistema todos los subsistemas que se desarrollen para esta empresa logrando una mejor organización y control.

Por lo expresado anteriormente se plantea como **problema a resolver** en la investigación: ¿Cómo integrar las soluciones actuales, así como los futuros desarrollos que se emprendan para el CEINPET en un sistema centralizado?

Teniendo en cuenta la amplia gama de posibilidades que brindan las tecnologías, se propone como **objetivo general** del presente trabajo: Elaborar una propuesta de arquitectura de software que soporte el desarrollo de una plataforma de servicios de aplicaciones para el CEINPET.

Siendo identificado después del estudio de la bibliografía (1) como **objeto de estudio**: El proceso de elaboración de la arquitectura de software.

Dentro de dicho objeto de estudio se enmarca el siguiente **campo de acción**: El diseño de una arquitectura de software para la plataforma de servicios de aplicaciones.

Para dar cumplimiento al objetivo general de esta investigación se llevaron a cabo las siguientes **tareas de la investigación**:

- Identificar las plataformas de servicios de aplicaciones implantadas por empresas nacionales e internacionales, relacionadas con la industria del petróleo o no.
- Identificar las necesidades y restricciones que el sistema debe tener.
- Seleccionar herramientas, tecnologías, patrones de diseño y estilos arquitectónicos a usar en la arquitectura para la plataforma.
- Diseñar una propuesta arquitectónica que cumpla con los requerimientos de la plataforma.
- Validar el diseño arquitectónico propuesto.

Finalmente, se muestran los métodos científicos que dan sustento a la investigación realizada.

Métodos Teóricos

- Histórico-Lógico: Para la realización del presente trabajo se comenzó realizando un estudio detallado de las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos de plataformas de integración de aplicaciones.

- Inductivo-Deductivo: A partir de la investigación realizada sobre las herramientas y tecnologías existentes se propondrá cuáles son las más factibles para el diseño de una arquitectura robusta que le dé sustento a la plataforma del CEINPET.
- Análisis y Síntesis: Se realizó un estudio y valoración de las herramientas y tecnologías existentes, que sirvieron de fundamento teórico para la investigación.

Métodos Empíricos

- Observación: De las aplicaciones desarrolladas anteriormente y de sistemas utilizados por el CEINPET.
- Revisión de documentos: Se revisaron documentos de la ingeniería del software, relacionados con las herramientas y tecnologías existentes y otros que pudieran aportar información sobre estos temas.

Con el estudio realizado se espera como resultado definir una arquitectura robusta que integre las aplicaciones desarrolladas y que permita la construcción de la plataforma de servicios de aplicaciones para el CEINPET cumpliendo con los requisitos solicitados por los clientes y resolviendo así el problema planteado.

El presente documento está estructurado en tres capítulos:

CAPÍTULO 1: “Fundamentación Teórica”: Se plantean todos los elementos teóricos que sustentan el problema científico y los objetivos del trabajo.

CAPÍTULO 2: “Tendencias y Tecnologías Actuales a Utilizar”: Se caracterizan, seleccionan y fundamentan las herramientas, tecnologías, estilos y patrones a usar en el desarrollo de la plataforma.

CAPÍTULO 3: “Presentación y Evaluación de la Solución Propuesta”: Se diseña y valida la propuesta arquitectónica para que cumpla con los requerimientos de la plataforma.

CAPÍTULO 1: Fundamentación Teórica.

1.1 Introducción

En este capítulo se enunciarán conceptos asociados al dominio del problema para su mejor entendimiento. Se hará una descripción general del objeto de estudio y la situación problemática. Además, se analizarán otras soluciones existentes. Se plantearán todos los elementos teóricos que sustentan el problema científico y los objetivos del trabajo.

1.2 Conceptos Asociados al Dominio del Problema

En este acápite se relacionan todos los conceptos que desde el punto de vista teórico permitan un mejor entendimiento de lo planteado.

Arquitectura de Software

Cada vez que se narra la historia de la arquitectura de software (o de la ingeniería de software, según el caso), se reconoce que en un principio, hacia 1968, Edsger Dijkstra, de La Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera. Dijkstra, sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores (2); aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases del mismo.

La arquitectura de software es un concepto fácil de entender, pero resulta difícil definirlo con precisión. En concreto, es difícil dibujar una línea precisa entre el diseño y la arquitectura, la arquitectura es un aspecto de diseño que se concentra en algunas características específicas.

La arquitectura es un nivel de diseño que se centra en aspectos: Más allá de los algoritmos y estructuras de datos de la computación; el diseño y la especificación de la estructura general del sistema emergen como una clase nueva de problema. Los aspectos estructurales incluyen la estructura global de control y la organización general; protocolos de comunicación, sincronización y acceso de datos; asignación de

funciones para diseñar elementos; distribución física, composición de elementos de diseño; ajuste y rendimiento; y selección entre otras alternativas de diseño. (3)

Pero la arquitectura es algo más que una estructura; es el concepto de más alto nivel de un sistema en su entorno. También incluye el "ajuste" con la integridad del sistema, con las restricciones económicas, con las preocupaciones estéticas y con el estilo. No se limita a un enfoque interior, si no que tiene en cuenta el sistema en su totalidad dentro del entorno de usuario y el entorno de desarrollo, un enfoque exterior. (4)

Se puede concluir que la Arquitectura Software es la representación de alto nivel de la estructura de un sistema o aplicación, que describe las partes que la integran, las interacciones entre ellas, los patrones que supervisan su composición, y las restricciones a la hora de aplicar esos patrones.

Plataforma

Desde el enfoque de la realización de sistemas informáticos, se le denomina plataforma a los diferentes ambientes creados para el desarrollo de software. Actualmente en el mundo existen diferentes tipos de plataformas de desarrollo para aplicaciones electrónicas y de escritorio. Hoy día los ambientes de programación se han dirigido especialmente al bienestar del desarrollador y a la unificación en un mismo entorno de desarrollo, la creación de aplicaciones con sus prestaciones para diferentes campos de acción como lo son, la web y las aplicaciones de escritorio. (5)

En informática, una plataforma de desarrollo es el entorno de software común en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo; sin embargo, también es posible encontrarla ligada a una familia de lenguajes de programación o a una Interfaz de programación de aplicaciones. (6)

Intel define una plataforma como un conjunto integrado de componentes que hace posibles modelos de uso específicos. (7)

Tipos y Características de las Plataformas

Hay diferentes tipos de plataformas: éstas pueden ser sistemas muy sencillos hasta llegar a ser altamente complejos y articulados. Una vez comprobadas las ventajas del uso de una plataforma es necesario

evaluar las diferentes herramientas de que se dispone y analizar sus funcionalidades, sin dejar de tomar en cuenta las exigencias formativas.(8) En una primera fase de orientación, resulta de gran utilidad considerar las numerosas aportaciones teóricas de quienes se han ocupado del análisis y evaluación de las características principales de estos sistemas y que han resaltado los elementos fundamentales a la hora de seleccionar una plataforma. (9)(10)(11)

Se tienen plataformas comerciales o propietarias, y por otro lado plataformas Open Source. Mientras que las primeras son realizadas por empresas de desarrollo de software para lograr su venta, las segundas son desarrolladas por una comunidad de informáticos que las pone a disposición sin costo alguno y cuya licencia permite el acceso al código fuente para mejorar el programa (licencia GNU GPL). Esta segunda opción resulta de particular interés para las organizaciones que se ocupan de la formación relacionada con la experimentación, como las universidades, y que necesitan investigar y experimentar a través de estas herramientas. De hecho, el software libre constituye un recurso verdaderamente valioso ya que es fácilmente accesible y disponible en comparación con plataformas comerciales que estas son extremadamente caras y de difícil acceso para los usuarios.

Componente

De manera general, un componente es un bloque de construcción modular para el software de cómputo. De manera más formal, la especificación unificada del lenguaje de modelado de OMG define un componente como “una parte modular desplegable y reemplazable de un sistema que encapsula implementación y expone un conjunto de interfaces”. (12)

En el contexto de la ingeniería del software orientada a objetos, un componente contiene un conjunto de clases que colaboran entre sí. En el contexto de la ingeniería del software convencional, un componente es un elemento funcional de un programa que incorpora la lógica del procesamiento, las estructuras internas de los datos necesarios para implementar dicha lógica, y una interfaz que permita la invocación del componente y el paso de los datos.

Un componente de software, según Clemens Alden Szyperski, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas. (13) Que sea una

unidad de composición y no de construcción quiere decir que no es preciso confeccionarla: se puede comprar hecha, o se puede producir para que otras aplicaciones la utilicen en sus propias composiciones.

1.3 Objeto de Estudio

Descripción General

En la introducción de este trabajo se definió como objeto de estudio el proceso de elaboración de la arquitectura de software. Para la implementación de cualquier software es muy importante su arquitectura ya que representa la estructura o las estructuras del sistema, que consta de componentes de software, las propiedades visibles externamente y las relaciones entre ellas. En RUP, la arquitectura de un sistema de software (en un punto determinado) es la organización o la estructura de los componentes importantes del sistema que interactúan mediante interfaces, con componentes compuestos de interfaces y componentes cada vez más pequeños.

Descripción Actual del Dominio del Problema

Actualmente nuestro país cuenta con un Centro de Investigaciones de la Unión CUBAPETRÓLEO del Ministerio de la Industria Básica denominado CEINPET, que surgió en 1996 y se dedica a la investigación aplicada en la industria del petróleo cubano, trabajando en función de la economía del país. Este centro, es el único en el país que se dedica a la ejecución de programas y proyectos de investigación y desarrollo, servicios científicos y técnicos, consultorías científicas y asesorías, comercialización de productos con alto valor agregado derivados de las investigaciones, y la prestación de servicios especializados científicos y tecnológicos para empresas nacionales y extranjeras. Es una organización líder que constituye el soporte tecnológico de CUPET, el cual es reconocido por su alta competitividad a nivel nacional e internacional.

El desarrollo de una plataforma de integración de aplicaciones para el CEINPET con una arquitectura robusta y flexible le permitirá integrar muchos procesos que se realizan individualmente en sus aplicaciones. En algunos casos se repiten tareas derrochando tiempo de ejecución y de búsqueda en la base de datos; y en otros, algunos resultados alternos de un proceso en una aplicación le sirven como dato a otra para su ejecución, desaprovechándose tiempo y recursos. Además de dar cabida para integrar nuevas funcionalidades o futuros desarrollos para esta empresa. Esta integración lograría también que

exista homogeneidad entre las herramientas y tecnologías a utilizar para el desarrollo de estas aplicaciones, logrando implementaciones más serias y la satisfacción del cliente.

1.4 Importancia y Necesidad de Definir una Arquitectura

Si se desea construir un Software de gran complejidad como es el caso de la plataforma de servicios de aplicaciones se necesita tener una visión común de sistemas en desarrollo. Por lo tanto, se necesita una arquitectura para:

- Comprender el sistema.
- Organizar el desarrollo.
- Fomentar la reutilización.
- Hacer evolucionar el sistema.

En RUP (Rational Unified Process), la arquitectura es principalmente un resultado del análisis y el flujo de trabajo de diseño. A medida que el proyecto reactiva este flujo de trabajo, de iteración en iteración, la arquitectura evoluciona hasta que está perfeccionada y pulida. Dado que todas las iteraciones incluyen integración y pruebas, la arquitectura es bastante robusta cuando se entrega el producto. Esta arquitectura es uno de los focos principales de las iteraciones de la fase de Elaboración, al final de la cual la arquitectura suele tener su línea base.

Características del Arquitecto

El arquitecto ideal debe ser una persona de letras, matemático, al corriente de estudios históricos, un estudiante diligente de la filosofía, conocido de la música, no ignorante de medicina, entendido en las respuestas de consultoría jurídica, al corriente de astronomía y de cálculos astronómicos.

Resumiendo, el arquitecto del software debe tener visión, y una vasta experiencia que permite tomar decisiones rápidamente y hacer el juicio adecuado, crítico en ausencia de la información completa.

Las principales actividades que realiza el arquitecto son:

- Priorizar los Casos de Uso: Definir los Casos de Uso como: críticos, secundarios, auxiliares u opcionales, lo que permite definir los módulos, subsistemas y escenarios así como la interacción entre ellos, que permita tomar decisiones hacia donde centrar los esfuerzos de implementación.
- Análisis de la arquitectura: Definir la arquitectura candidata del sistema teniendo en cuenta arquitecturas similares u otros sistemas, definir además los estilos arquitectónicos, patrones de arquitectura, los principales mecanismos de diseño arquitectónico.
- Identificar mecanismos de diseño: Refinar el análisis de la arquitectura teniendo en cuenta las restricciones impuestas por el entorno de implementación.
- Estructurar el modelo de implementación: Permite establecer la estructura en la que va a residir la implementación del sistema.
- Reutilización de elementos de diseño existentes: Permite analizar las interacciones en los diagramas de clases del Análisis para encontrar interfaces, diseño de clases y subsistemas. Refinar la arquitectura e incorporar elementos reutilizables si es posible, identificar problemas comunes a los que se pueda crear soluciones generales o comunes (patrones, o familias de productos).
- Identificar los elementos de diseño: Analizar las interacciones entre las clases de Análisis para identificar los elementos de modelo de diseño arquitectónico.
- Describir la arquitectura en tiempo de ejecución: Analizar requerimientos de concurrencia, identificar los procesos y la comunicación entre ellos, identificar el ciclo de vida de los mismos.

1.5 Análisis de las Soluciones Existentes en el Mundo

Existen en el mundo compañías que han dado soluciones muy viables para la integración de aplicaciones petroleras. Compañías que han desarrollado plataformas muy utilizadas en esta industria debido a la factibilidad de su uso y que además cuentan con gran aceptación por las empresas productoras de petróleo. Por ejemplo la mundialmente reconocida **Schlumberger** (14), una empresa alemana fundada hace casi un siglo que de unos años a la actualidad viene liderando el campo del petróleo como proveedor de servicios para compañías productoras de crudo y gas del mundo.

Esta empresa pone a disposición del usuario calificado una plataforma como **Petrel**, la cual cuenta con procesos como:

- Modelado estratigráfico
- Geofísica
- Modelado estructural
- Modelado de propiedad
- Ingeniería de pozo
- Simulación

Computer Modeling Group (15) es una empresa canadiense especializada en el modelado y simulación de soluciones prácticas para yacimientos de gas y crudo, procedimientos avanzados de recuperación de petróleo, ingeniería de reservorio, consulta, entrenamiento y soporte técnico para clientes alrededor del mundo. Aquí se muestran algunos de los procesos de su plataforma (**CMG**):

- Simulador
- Resultados del post-procesamiento de solicitudes
- Simulador de embalse de vapor, termal, y procesos avanzados

Kappa (16) es otra compañía de software creada en 1987 para la exploración y producción de petróleo. La mayoría de sus aplicaciones se integran ahora en un entorno de estación de trabajo única, **Ecrin**. Esta es una plataforma integrada para el análisis dinámico de flujo. Entre otras posibilidades que brinda esta empresa está el entrenamiento y consulta para el uso de sus servicios por los clientes vinculados a la industria del petróleo. Entre sus procesos se pueden encontrar:

- Saphir
- Diamant Master
- Topaze
- Rubis
- Amethyste
- Emeraude

La investigación de las plataformas puestas en el mercado por estas empresas vinculadas a la industria petrolera no arrojó muchos datos que pudieran interesar desde el punto de vista arquitectónico ya que entre las principales desventajas de su uso se encuentra; en primer lugar, que son software propietario y

por lo tanto no comparten su diseño ni su código. Además de que las licencias para utilizar estos productos son muy caras, un lujo q no nos podemos dar en nuestro país. Una razón en aumento para poner más empeño a la tarea de producir nuestra propia plataforma de servicios de aplicaciones para la industria petrolera de nuestro país, que integre nuestras propias aplicaciones.

1.6 Análisis de las Soluciones Desarrolladas en el Polo

El CEINPET no posee actualmente una plataforma que integre aplicaciones desarrolladas para la industria del petróleo. La mayoría de las aplicaciones que se utilizan son de producción extranjera y bajo licencia. Un ejemplo de ello es el Petrel mencionado anteriormente, utilizado por los especialistas del CEINPET en sus investigaciones. Con todos estos argumentos nuestra universidad se ha dado a la tarea de la producción de nuestras propias aplicaciones así como de la plataforma para integrarlas.

Por lo tanto, ya se están desarrollando las primeras versiones de aplicaciones como:

Sistema de Gestión de Lechadas para la Cementación de Pozos Petroleros, el cual automatiza el análisis de los datos del pozo para definir cuál sería la lechada más factible a utilizar debido a que la actividad de cementación de los pozos petroleros es fundamental en la durabilidad o vida del pozo. La misma cuenta con una serie de actividades de las que depende una correcta selección de dicha lechada por los especialistas. (17)

Para la construcción de este software se utilizó como lenguaje de modelado UML, como metodología RUP, la herramienta CASE utilizada fue Visual Paradigm, el gestor de base de datos PostgreSQL, como lenguaje de programación Java, el framework utilizado Hibernate, y con una arquitectura en capas.

Sistema para la Gestión de Información de Pérdida de Circulación durante la Perforación de Pozos Petroleros, el cual facilita el manejo y la búsqueda de la información referente a las averías y complejidades, las soluciones aplicadas y los resultados. Para que los especialistas conozcan las causas específicas que conducen a las complejidades en el sistema petrolero cubano y la búsqueda de soluciones efectivas o la predicción de futuros posibles problemas. (18)

En el desarrollo de este software se utilizó como lenguaje de programación PHP con framework Symfony, una arquitectura MVC, como gestor de base de datos PostgreSQL, como metodología RUP y lenguaje de modelado UML. Las herramientas fueron ZendStudio for Eclipse, EMS SQL Manager for PostgreSQL y Visual Paradigm como herramienta CASE.

Sistema para la Evaluación y Control del trabajo con los Fluidos de Perforación, el cual brinda a los especialistas reportes diarios sobre la evaluación y control del trabajo con los fluidos de perforación que es de vital importancia en el proceso de perforación. Por la razón de que es necesario percatarse de los errores que se pudieran cometer minimizando los riesgos y los gastos que pudieran significar pérdidas a la economía del país. (19)

Para la implementación de este software se utilizó RUP como metodología, UML como lenguaje de modelado y Visual Paradigm como herramienta CASE. Como gestor de base de datos se usó PostgreSQL, el lenguaje de programación Java, como ambiente de desarrollo integrado NetBeans los frameworks utilizados fueron Swing e Hibernate y se desarrolló con una arquitectura 3 capas.

Sistema para la Identificación de Aguas en Pozos Petroleros, el mismo automatiza el proceso completo de identificación de modelos de aguas encontradas en pozos petroleros, empleando técnicas de reconocimiento de patrones. (20)

Para su confección se utilizó como metodología de desarrollo AUP, como lenguaje de modelado UML y como herramienta CASE Visual Paradigm. El lenguaje de programación Csharp con framework MONO, el ambiente de desarrollo integrado fue Monodevelop 2.0 y como gestor de base de datos SQLite. El sistema se desarrolló con una arquitectura 3 capas.

Sistema para la gestión de la información de muestras de núcleos, el mismo ayuda a los especialistas a evaluar los daños que causan la formación de los diferentes filtrados de la perforación o aguas de inyección. Permite a los petrofísicos definir parámetros como la distribución de fluidos y propiedades eléctricas y estructurales. Hace posible además la caracterización de yacimientos y es capaz de proporcionar a los geofísicos una visión de la estructura de las capas del subsuelo. (21)

Como metodología de desarrollo se utilizó RUP, como lenguaje de modelado UML y como herramienta de modelado Visual Paradigm. El lenguaje de programación fue Java, el gestor de base de datos PostgreSQL, NetBeans como IDE de desarrollo y se utilizó una arquitectura 3 capas.

Sistema para el diagnóstico de daños en pozos petroleros, el cual ayuda a los especialistas a realizar un diagnóstico correcto del daño que pueda presentar el pozo, sigue una serie de pasos, en forma de algoritmo. Los mismos van formando una metodología, las cuales basan su desarrollo en un conjunto de métodos en su mayoría guiados por fórmulas y leyes matemáticas, de las cuales se derivan ciertas gráficas que permiten valorar el daño. (22)

Para su construcción se utilizó como lenguaje de modelado UML, el lenguaje de programación fue Java, el IDE de desarrollo NetBeans, como metodología RUP, como gestor de base de datos PostgreSQL, la herramienta CASE Visual Paradigm, como framework se utilizó Swing y fue desarrollado con arquitectura MVC.

Con el estudio realizado de las aplicaciones ya desarrolladas se puede llegar a la conclusión de que no existe homogeneidad entre las herramientas ni las tecnologías utilizadas para su desarrollo. Así como tampoco está definido un estándar para la arquitectura de las aplicaciones ya que fueron implementadas usando diferentes variantes. Por lo que se hace necesaria una homogeneización en cuanto a estos aspectos para el desarrollo de las futuras aplicaciones que se emprenda para el CEINPET; así como para lograr la integración de estas con la plataforma. Por estas razones se pretende volver a desarrollar estas aplicaciones y las demás que el cliente pueda necesitar en el futuro, con las tecnologías y herramientas que se propondrán en el próximo capítulo de este trabajo.

1.7 Propuesta de Módulos para la Plataforma de Servicios de Pozos

Tras el estudio de las soluciones existentes en el mundo, en especial de la plataforma Petrel de la compañía Schlumberger y de las soluciones desarrolladas por el Polo se pueden proponer entonces para nuestra plataforma los módulos o procesos de:

- Geofísica
- Modelado

- Simulación
- Utilidades

El proceso de **Geofísica** agruparía los subsistemas referentes a la sísmica y otros. El proceso de **Modelado** englobaría los subsistemas referentes a las fallas y daños en los pozos petroleros así como los modelos estructurales y de propiedades, entre los que se encuentran los análisis petrofísicos, de muestras de núcleos, rocas y otros. El proceso de **Simulación** contendría los subsistemas referentes a los análisis de fluidos de perforación y de aguas en pozos petroleros así como todos los modelos de simulación de los reservorios. El proceso de **Utilidades** tendría las prestaciones de los subsistemas relacionados con los reportes y las gráficas, las conversiones, el procesamiento de los datos, entre otras.

1.8 Conclusiones Parciales

En este capítulo se plantean todos los elementos teóricos que sustentan el problema científico y los objetivos del trabajo. Se enunciaron conceptos asociados al dominio del problema para su mejor entendimiento. Conceptos como el de arquitectura de software, plataforma y componente para poder comprender mejor los términos de este trabajo. Se describió el objeto de estudio en general y todo lo asociado al dominio del problema. Además, se analizaron otras soluciones existentes en el mundo y en el Polo productivo de plataformas de aplicaciones del petróleo y sus módulos así como su repercusión en la investigación. Por lo que se espera dar una visión al lector de los aspectos teóricos manejados en este trabajo.

CAPÍTULO 2: Tendencias y Tecnologías Actuales a Utilizar.

2.1 Introducción

En este capítulo se fundamentan las tecnologías y herramientas a utilizar para la construcción de la plataforma que dará solución al problema del CEINPET. Este grupo de aspectos lo conforman: la metodología para guiar el proceso de desarrollo del sistema; el lenguaje de modelado, capaz de proporcionar el entendimiento del proceso de desarrollo a través de diagramas; la herramienta CASE que soporte dicho modelado; el gestor de base de datos escogido; los lenguajes de programación; el entorno de desarrollo; las herramientas para el control de versiones; los frameworks que sustentarán la aplicación y los patrones de diseño y estilos arquitectónicos a aplicar.

2.2 Estilos Arquitectónicos

Aunque en los últimos 50 años se han desarrollado millones de sistemas de cómputo o software, la gran mayoría puede clasificarse en un número relativamente pequeño de estilos arquitectónicos.

Dewayne Perry y Alexander Wolf definen un estilo arquitectónico como una abstracción de tipos de elementos y aspectos formales a partir de diversas arquitecturas específicas. Un estilo arquitectónico encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus relaciones posibles.

Mary Shaw y Paul Clements identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer el sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

Robert Allen y David Garlan asimilan los estilos arquitectónicos a descripciones informales de arquitectura basadas en una colección de componentes computacionales, junto a una colección de conectores que describen las interacciones entre los componentes. Consideran que esta es una descripción deliberadamente abstracta, que ignora aspectos importantes de una estructura arquitectónica, tales como una descomposición jerárquica, la asignación de computación a los procesadores, la coordinación global y

el plan de tareas. En esta concepción, los estilos califican como una macro-arquitectura, en tanto que los patrones de diseño serían más bien micro-arquitecturas.

Mark Klein y Rick Kazman proponen una definición según la cual un estilo arquitectónico es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes aparejados por el uso del estilo. Los estilos arquitectónicos, afirman, son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas.

La definición de cada uno de estos autores es aceptada ya que está redactada desde diferentes puntos de vista pero siempre enfocando a la conexión entre los componentes de un sistema informático. Cada uno de estos autores además tienen su consideración acerca de las categorías de los estilos arquitectónicos y he aquí una muestra de los más representativos y vigentes:

- Estilos de Flujo de Datos
  - Tubería y filtros
- Estilos Centrados en Datos
  - Arquitecturas de Pizarra o Repositorio
- Estilos de Llamada y Retorno
  - Modelo-Vista-Controlador (MVC)
  - Arquitecturas en Capas
  - Arquitecturas Orientadas a Objetos
  - Arquitecturas Basadas en Componentes
- Estilos de Código Móvil
  - 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 (SOA)
- Arquitecturas Basadas en Recursos

Por lo que se puede concluir que para la construcción de un software se pueden definir uno o muchos estilos arquitectónicos. Cada uno va a describir una categoría de sistema que abarque ciertos aspectos, como pudieran ser:

- Un conjunto de componentes (base de datos, módulos computacionales) que realizan una funcionalidad requerida por el sistema.
- Un conjunto de conectores que permitan la comunicación, coordinación y cooperación entre los componentes.
- Un conjunto de restricciones que definen cómo se integrarían los componentes para formar el sistema.
- Un conjunto de modelos semánticos que permitan a un diseñador, mediante el análisis de las propiedades conocidas de las partes que lo integran, comprender las propiedades generales del sistema.

A continuación se relaciona el estilo arquitectónico propuesto y las arquitecturas relacionadas a su familia a utilizar en el desarrollo de la plataforma de servicios a pozos.

Estilo de Llamada y Retorno

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

Este estilo arquitectónico permite que un arquitecto de software obtenga una estructura de programa que resulta relativamente fácil modificar y cambiar de tamaño. A continuación se muestra dentro de este estilo arquitectónico cuáles fueron las arquitecturas propuestas para la implementación de la plataforma de servicios a pozos.

Arquitectura Orientada a Componentes

Una arquitectura basada en componentes describe una aproximación de ingeniería de software al diseño y desarrollo de un sistema. Esta arquitectura se enfoca en la descomposición del diseño en componentes funcionales o lógicos que expongan interfaces de comunicación bien definidas.

El estilo de arquitectura basado en componentes tiene las siguientes características:

- Es un estilo de diseño para aplicaciones compuestas de componentes individuales, los cuales pudieran ser las unidades de modelado, diseño e implementación.
- Pone énfasis en la descomposición del sistema en componentes lógicos o funcionales que tienen interfaces bien definidas, las cuales están separadas de las implementaciones.
- Define una aproximación de diseño que usa componentes discretos, los que se comunican a través de las interfaces.

Los siguientes son los principales beneficios del estilo de arquitectura basado en componentes para la plataforma de servicios de aplicaciones:

- **Facilidad de Instalación:** Cuando una nueva versión esté disponible o un nuevo componente se desee integrar, este se podrá reemplazar sin impacto en otros componentes o el sistema.
- **Costos reducidos:** El uso de componentes de terceros permite distribuir el costo del desarrollo y del mantenimiento o de definir líneas de desarrollo.
- **Facilidad de desarrollo:** Los componentes implementan una interfaz bien definida para proveer la funcionalidad definida permitiendo el desarrollo sin impactar otras partes del sistema.
- **Reusable:** El uso de componentes reutilizables significa que ellos pueden ser usados para distribuir el desarrollo y el mantenimiento entre múltiples aplicaciones y sistemas.

Todas las ventajas que provee este estilo arquitectónico son significativas para el desarrollo de la plataforma de servicios de aplicaciones por lo que se propone utilizarlo en el desarrollo de la misma. Por esta razón las aplicaciones y subsistemas serán desarrollados como componentes a integrar en la plataforma, pudiendo cada uno implementarse usando el estilo arquitectónico que satisfaga las

necesidades de su implementación, solo que debe proveer interfaces bien definidas para su integración en el sistema. Por lo que se propone como macro-arquitectura usar la de orientada a componentes.

La única desventaja que puede tener el uso del estilo arquitectónico basado en componentes es el problema de incompatibilidad de versiones que ya ha sido largamente superado en toda la industria a lo largo de los años. Por lo que debe de aparecer entre los requisitos no funcionales la especificación de las versiones de cada una de las herramientas y tecnologías a utilizar.

Arquitectura en Capas

El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico. No confundir con el término “nivel”.

El estilo en capas se comporta como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Se enfoca en la distribución de roles y responsabilidades de forma jerárquica proveyendo una forma muy efectiva de separación de responsabilidades. El rol indica el modo y tipo de interacción con otras capas, y la responsabilidad indica la funcionalidad que está siendo desarrollada.

La variante más usada de este estilo arquitectónico es la de tres capas, el cual propone una capa de presentación para interactuar con el usuario, otra de lógica de negocio para implementar las funcionalidades necesarias y otra capa para el acceso a datos la cual se encarga de interactuar con la base de datos y gestionar la información necesaria.

El estilo de arquitectura basado en capas se identifica por las siguientes características:

- Describe la descomposición de servicios de forma que la mayoría de la interacción ocurre solamente entre capas vecinas.
- Las capas de una aplicación pueden residir en la misma máquina física (mismo nivel) o puede estar distribuido sobre diferentes computadores (n-niveles).
- Los componentes de cada capa se comunican con otros componentes en otras capas a través de interfaces muy bien definidas.

Los principales beneficios que provee el estilo de arquitectura basado en capas para la plataforma de servicios a pozos son:

- **Abstracción:** Las capas permiten que se realicen cambios a un nivel abstracto. Los cambios en una capa no tienen por qué afectar otras.
- **Aislamiento:** El estilo de arquitectura en capas permite aislar los cambios en tecnologías a ciertas capas para reducir el impacto en el sistema total.

Esta arquitectura se adapta para el diseño de las aplicaciones a re implementar en la primera versión, como es el caso de MUDMAN y ANACEM, las cuales ya fueron implementadas utilizando este estilo arquitectónico. El cual permite con la creación de las capas que cuando la complejidad del sistema aumente, este se pueda ampliar con facilidad. En caso de algún problema o cambio en alguna capa se va directamente a la misma sin tener que revisar completamente el código. Con su utilización se garantiza una organización en el sistema, lo que le posibilita a los implementadores hacer los problemas complejos en secuencias de pasos incrementales, además facilita la corrección de errores.

Arquitectura Modelo-Vista-Controlador (MVC)

Para el desarrollo de aplicaciones también es utilizado el estilo arquitectónico Modelo-Vista-Controlador que como bien indica su nombre propone tres componentes fundamentales que se relacionan entre sí. Este trata de realizar un diseño que desacople los datos de una aplicación, la interfaz de usuario, y la lógica de negocio en tres componentes distintos, con la finalidad de mejorar la reusabilidad. De esta forma, las modificaciones en las vistas impactan en menor medida en la lógica de negocio o los datos.

El estilo conocido como Modelo-Vista-Controlador (MVC) separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes. (23)

Clases del estilo:

Modelo: Gestiona el comportamiento y los datos de la aplicación, responde a las peticiones que realizan las vistas sobre su estado y permite su actualización normalmente desde el controlador.

Vista: Interpreta las acciones del usuario, accediendo a las operaciones de negocio de la aplicación y modificando a partir de sus resultados el estado del modelo y la navegación entre vistas.

Controlador: Muestra el estado al usuario de la aplicación, redirigiendo las acciones que realiza sobre el interfaz al controlador.

Este estilo arquitectónico es propuesto por los desarrolladores del framework QT (el cual se detalla en el siguiente capítulo) para la implementación de las interfaces de usuario. Se propone utilizar este framework en la capa de presentación de las aplicaciones que sean implementadas utilizando un estilo arquitectónico en capas, propuesto anteriormente, o en el módulo encargado de la interacción con el usuario o que provea las interfaces de usuario, para otros estilos arquitectónicos.

Este estilo ayudará a que cuando la interfaz de una aplicación necesite ser cambiada, ya sea cuando dicha aplicación se encuentre en su desarrollo o los clientes lo soliciten; los cambios realizados en la vista no afecten al modelo. Ya que los requerimientos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas del negocio.

2.3 Patrones de Diseño

Definir patrón, simplemente, en lenguaje llano, pudiera ser: manera de hacer las cosas siguiendo un estándar.

Ahora bien, Christopher Alexander, promotor de los patrones, los cuales creó en 1977 plantea que “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”.

Según otros autores los patrones capturan la experiencia existente y probada para promover buenas prácticas. (24)

Aunque también se pudiera agregar de manera más simple que un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas

situaciones y discusiones sobre sus compromisos. Con el fin de reutilizar la experiencia humana ante problemas ya probados y solucionados.

Existen varias categorías de patrones según el nivel de abstracción entre los que se encuentran:

- **Los patrones arquitecturales:** Aquellos que expresan un esquema organizativo estructural fundamental para sistemas de software.
- **Los idiomas:** Patrones de bajo nivel específicos para un lenguaje de programación o un entorno completo. (Estándares de código)
- **Los patrones de diseño:** Aquellos que expresan un esquema para definir estructuras de diseño (o sus relaciones) con las que construir sistemas de software.

Dividir un problema en partes siempre ha sido uno de los objetivos de una buena programación orientada a objetos. Para ello utilizaremos **Patrones de Diseño** (Design Patterns) que son modelos de trabajo enfocados a dividir un problema en partes de modo que nos sea posible abordar cada una de ellas por separado para simplificar una solución. Un patrón de diseño es un conjunto de reglas que describen como afrontar tareas y solucionar problemas que surgen durante el desarrollo de software.

También se pueden ver como la solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). Por lo tanto están basados en la recopilación del conocimiento de los expertos en desarrollo de software.

Entre los objetivos de los patrones de diseño se encuentran:

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

Estos patrones se pueden clasificar según su propósito:

Patrones creacionales: Abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas.

Dentro de esta clasificación se encuentran los patrones:

- **Singleton** (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.
- **Factory Method** (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.
- **Abstract Factory** (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- **Prototype** (Prototipo): Crea nuevos objetos clonándolos de una instancia ya existente.
- **Builder** (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.

Patrones estructurales: Tratan de conseguir que cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Estudian cómo se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos.

Dentro de esta clasificación se encuentran los patrones:

- **Adapter** (Adaptador): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- **Bridge** (Puente): Desacopla una abstracción de su implementación.
- **Composite** (Objeto compuesto): Permite tratar objetos compuestos como si se tratase de uno simple.
- **Decorator** (Envoltorio): Añade funcionalidad a una clase dinámicamente.

- **Facade** (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- **Flagweight** (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
- **Proxy** (Apoderado) Mantiene un representante de un objeto.

Patrones de comportamiento: Los patrones de comportamiento estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal.

Dentro de esta clasificación se encuentran los patrones:

- **Chain of Responsibility** (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.
- **Command** (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- **Iterator** (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- **Mediator** (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- **Observer** (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- **State** (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- **Strategy** (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- **Template Method** (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- **Memento** (Recuerdo): Permite volver a estados anteriores del sistema.

- **Interpreter** (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- **Visitor** (Visitante): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.

Argumentar que los patrones contenidos dentro de estas tres clasificaciones también son conocidos como Patrones GOF (Gang Of Four o Banda de los 4)

Patrones GRASP (General Responsibility Assignment Software Patterns): Estos son patrones generales de software para asignación de responsabilidades. Describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

Dentro de esta clasificación se encuentran los patrones:

- **Experto:** Garantiza que una clase contenga toda la información necesaria para realizar la labor que tiene encomendada, o sea que sea la experta en esa información.
- **Creador:** Permite asignar la responsabilidad de que una clase cree un objeto de otra bajo ciertas condiciones.
- **Controlador:** Permite asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.
- **Alta Cohesión:** Garantiza que cada elemento de nuestro diseño realice una labor única dentro del sistema, no desempeñada por el resto de los elementos.
- **Bajo Acoplamiento:** Garantiza que haya poca dependencia entre las clases.

Además existen otros cuatro los cuales se consideran adicionales y son:

- Polimorfismo
- Fabricación Pura
- Indirección
- No Hables con Extraños.

Algunos autores consideran que estos patrones de asignación de responsabilidades están contenidos dentro de los patrones de diseño por describir los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Por lo tanto fueron incluidos como patrones de diseño en esta investigación.

Se relacionan a continuación los patrones de diseño propuestos a utilizar en el desarrollo de la plataforma de servicios de aplicaciones:

Singleton

Este patrón se utiliza con el propósito de garantizar que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Este patrón surge cuando se cree importante que algunas clases tengan exactamente una instancia y que sean fácilmente accesibles. Además, cuando una variable global no previene de crear múltiples instancias de objetos, una solución mejor es hacer que sea la propia clase la responsable de su única instancia, quien deberá garantizar que no se pueda crear ninguna otra (interceptando las peticiones para crear nuevos objetos) y proporcione un modo de acceder a ella.

Este patrón ha sido escogido para ponerlo en uso en la conexión a la base de datos por los componentes a desarrollar para la plataforma, logrando así tener una instancia global para lograr este propósito. Creando una clase conexión y con un objeto de la misma establecer dicha conexión, se logra establecer una instancia global a ser utilizada por cada funcionalidad del negocio del sistema en el momento de acceder a los datos.

Este patrón va a proveer como ventajas para la plataforma de servicios de aplicaciones las siguientes:

- Acceso controlado a la única instancia. Encapsula su única instancia, puede tener un control estricto sobre cómo y cuándo acceden a ella los clientes.
- Espacio de nombres reducido. Es una mejora sobre las variables globales. Evita contaminar el espacio de nombres con variables globales que almacenen las instancias.

Factory Method

Este patrón se utiliza con el propósito de definir una interfaz para crear un objeto, dejando que las subclases decidan qué clases instanciar. Permite a una sub clase delegar la instanciación a sus subclases.

Este patrón se recomienda usar cuando una clase no puede anticipar qué objetos va a crear. También se usa cuando una clase quiere que sus subclases indiquen qué objetos crea. Es llamado fabricación por tener la responsabilidad de fabricar un objeto.

Este patrón se propone utilizarlo en la implementación del componente propuesto para realizar las conversiones de unidades de medida. Por la razón de que la plataforma manejará diferentes tipos de unidades de medida como son longitud, densidad, entre otras tantas. Se propone entonces crear una clase Factory donde las sub clases sean las encargadas de garantizar el cálculo de las conversiones de las unidades de medida en dependencia de las mismas. Esta clase Factory será la encargada de decidir según los parámetros que se le pasen el tipo de objeto que devolverá.

Este patrón va a proveer como ventajas que:

- Se gane en flexibilidad, pues crear los objetos dentro de una clase con un “Método de Fábrica” es siempre más flexible que hacerlo directamente, debido a que se elimina la necesidad de atar nuestra aplicación unas clases de productos concretos.
- Se facilitan futuras ampliaciones, puesto que se ofrece a las subclases la posibilidad de proporcionar una versión extendida de un objeto.

Facade

Este patrón se utiliza para proporcionar una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

Este patrón se recomienda utilizarlo cuando es necesario estructurar un sistema en subsistemas, ayuda a reducir la complejidad. Un típico objetivo de diseño es minimizar la comunicación y dependencias entre

subsistemas. Un modo de lograr esto es introduciendo un objeto fachada que proporcione una interfaz única y simplificada para los servicios más generales del subsistema.

Este patrón se propone utilizarlo para establecer una Fachada de comunicación entre la plataforma y los componentes de la misma, en el momento de integrarse o entre cada uno de los componentes a desarrollar para la plataforma. Proveyendo así una interfaz de comunicación entre los componentes, tan necesaria como se ha explicado anteriormente. Esta Fachada permitirá la comunicación entre cada uno de los componentes que necesiten de otro para su funcionamiento.

Se propone usar esta Fachada para proporcionar una interfaz simple para un subsistema complejo como será el caso de los componentes de la plataforma de servicios a pozos ya que los subsistemas suelen volverse más complicados a medida que van evolucionando.

Las principales ventajas que va a proveer este patrón serán que:

- Oculta a los clientes los componentes del subsistema, reduciendo así el número de objetos con los que tratan los clientes y haciendo que el subsistema sea más fácil de usar.
- Promueve un débil acoplamiento entre el subsistema y sus clientes (permite modificar los componentes de un subsistema sin que sus clientes se vean afectados).

Experto

Lo que plantea este patrón es asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con la responsabilidad. El problema que resuelve el patrón experto está referido al principio más básico mediante el cual las responsabilidades son asignadas en el diseño orientado a objetos.

El modelo de clases puede definir docenas o cientos de clases de software, y una aplicación puede definir cientos o miles de responsabilidades a ser cumplidas. Durante el diseño orientado a objetos, cuando las interacciones entre objetos son definidas, se asignan las responsabilidades a las clases. Haciéndolo bien, los sistemas tienden a ser fáciles de entender, mantener y extender, y hay una oportunidad de re usar los componentes en aplicaciones futuras. (25)

Para verlo más claramente, 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.

El experto es usado más que cualquier otro patrón en la asignación de responsabilidades, es un principio usado continuamente en el diseño orientado a objetos.

Este patrón provee las siguientes ventajas:

- Desde que los objetos usan sus propias informaciones para realizar tareas. Esto permite poco acoplamiento, lo cual conduce a sistemas más robustos y de mantenimiento mucho más fácil.
- El comportamiento está distribuido a lo largo de clases que tienen la información requerida. La alta cohesión también es soportada.

Alta Cohesión

La cohesión 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. (26)

Una baja cohesión hace muchas cosas no afines o realiza trabajo excesivo. Esto presenta los siguientes problemas para las clases:

- Son difíciles de comprender.
- Difíciles de reutilizar.
- Difíciles de conservar.
- Las afectan constantemente los cambios.

La solución es asignar una responsabilidad de modo que la cohesión siga siendo alta. Al asignar responsabilidades en el diseño, se buscan soluciones que asignen los métodos a las clases de forma coherente, completa y relacionada. De esta forma, se obtendrán clases cohesionadas. En fin, se propone que la información que almacena una clase sea coherente y esté (en la medida de lo posible) relacionada con la clase.

En la práctica, el nivel de cohesión no puede ser considerado independiente de los otros patrones Experto y Bajo Acoplamiento.

Este patrón provee las siguientes ventajas:

- Mejora la claridad y facilidad con que se entiende el diseño.
- Se simplifica el mantenimiento y las mejoras de funcionalidad.
- A menudo, se genera un bajo acoplamiento.
- Soporta mayor capacidad de reutilización.

Bajo Acoplamiento

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. Acoplamiento bajo significa que una clase no depende de muchas clases. (27)

Por otro lado, acoplamiento alto significa que una clase recurre a muchas otras clases. Esto presenta los siguientes problemas:

- Los cambios de las clases afines ocasionan cambios locales.
- Difíciles de entender cuando están aisladas.
- Difíciles de reutilizar puesto que dependen de otras clases.

Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto a la herencia.

La solución a este problema es asignar una responsabilidad para mantener bajo el acoplamiento. El grado de acoplamiento no puede considerarse aisladamente de otros principios como Experto y Alta Cohesión.

Resumiendo, la idea de este patrón es tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases.

Este patrón provee las siguientes ventajas:

- Disminuye la dependencia entre clases.
- Clases fáciles de entender por separado.
- Clases fáciles de reutilizar.

2.4 Metodología de Desarrollo

Para desarrollar un software se necesita seguir una metodología para lograr una organización en el trabajo por parte del equipo de desarrollo y para lograr clientes satisfechos con el producto. La metodología de desarrollo se considera un conjunto de técnicas, procedimientos y soporte documental que ayuda a los desarrolladores a realizar el software. También se puede decir que representa el camino para desarrollar un software de manera sistemática.

Con las metodologías de desarrollo se pueden lograr mejores aplicaciones que conduzcan a una mejor calidad debido a un desarrollo controlado ya que son un proceso normalizado en una organización no dependiente del personal.

Se propone utilizar una metodología robusta o pesada como el caso de **RUP** (Rational Unified Process) para regir el desarrollo. Para aplicar esta metodología no se necesita tener contacto frecuente con el cliente, lo cual es muy conveniente por la lejanía de sus instalaciones. Además, esta metodología produce gran cantidad de documentación por lo que se considera lenta pero con buenos resultados lo cual no afecta porque el cliente no se encuentra apresurado con el producto.

Esta es una metodología de software que permite el desarrollo de aplicaciones a gran escala como la nuestra, mediante un proceso continuo de pruebas y retroalimentación, garantizando el cumplimiento de ciertos estándares de calidad. Aunque con el inconveniente de generar mayor complejidad en los controles de administración del mismo. Sin embargo, los beneficios obtenidos recompensan el esfuerzo invertido en este aspecto.

El proceso de desarrollo constituye un marco metodológico que se define en términos de metas estratégicas, objetivos, actividades y artefactos (documentación) requeridos en cada fase de desarrollo.

RUP es un proceso de desarrollo orientado a objetos, utiliza el Lenguaje Unificado de Modelado (UML) como lenguaje de representación visual. Este proceso unificado define “Quién”, “Cómo”, “Cuándo” y “Qué” debe hacerse en el proyecto.

Tiene tres características fundamentales: es iterativo e incremental, centrado en la arquitectura y dirigido por casos de usos. (28)

Dirigido por casos de uso: Los casos de uso representan los requisitos de software capturados durante el flujo de trabajo de requisitos, la planificación del proyecto se hace en términos de casos de uso, los desarrolladores crean realizaciones de casos de uso en términos de clases y subsistemas, los componentes se incorporan en los incrementos y cada uno realiza un conjunto de casos de uso, y por último se verifica que el sistema implementa los casos de uso correctos para el usuario. En otras palabras los casos de uso guían la arquitectura del sistema, enlazan todas las actividades del desarrollo y dirigen el proceso de desarrollo.

Centrado en la arquitectura: La arquitectura representa la forma del futuro sistema en términos de vistas arquitectónicas, sobre la cual el equipo de desarrollo y usuarios deben estar de acuerdo, ya que estas describen los elementos del modelo más importantes para su desarrollo, la arquitectura va madurando en las interacciones comenzando con los casos de uso relevantes desde el punto de vista arquitectónico.

Iterativo e incremental: “El Proceso Unificado propone que cada fase se desarrolle en iteraciones, ya que el incremento en la complejidad de los sistemas actuales hace que sea factible dividir el trabajo en partes más pequeñas o mini-proyectos. Cada mini-proyecto es una iteración que resulta en un incremento. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros”.

Dentro de las principales ventajas de RUP para el desarrollo de la plataforma de servicios a pozos se encuentran:

- Mitigación temprana de posibles riesgos
- Progreso visible en las primeras etapas
- El conocimiento adquirido en una iteración puede aplicarse de iteración a iteración

- Abordar las cuestiones de alto riesgo y valor en las primeras iteraciones
- Verificar continuamente la calidad desde el principio y con frecuencia
- Aplicar casos de uso
- Modelar el Software visualmente
- Gestión cuidadosa de requisitos

Pero a mi entender la principal ventaja que posee esta metodología para ser usada en el desarrollo de la plataforma de servicios a pozos es la de que es la que domina el equipo de desarrollo.

2.5 Lenguaje de Modelado

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar un diseño de software orientado a objetos.

Algunas organizaciones los usan extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo. Esto puede suponer también que las interacciones entre partes del programa den lugar a sorpresas cuando el modelo ha sido convertido en un software funcional. (24)

Para el desarrollo de un sistema de software se utilizan varios diagramas debido a la cantidad de procesos o métodos con que este cuenta, ya que todos no se pueden incluir en un solo diagrama. En el caso específico de la arquitectura, para que se convierta en una herramienta precisa y útil en el desarrollo de los sistemas de software se hace necesario que cuente con una manera precisa de representarla, especificarla, visualizarla y documentarla.

Con este fin es que se usan los lenguajes de modelado, no solo para los arquitectos si no para el equipo de desarrollo en general para que exista una buena representación de todas las ideas y características del sistema. Esto conlleva además al entendimiento del equipo de desarrollo.

Uno de los lenguajes de modelado más utilizado en el mundo por las empresas productoras de software es el Lenguaje Unificado de Modelado o **UML** del inglés respaldado por el OMG (Object Management Group).

UML unifica principalmente los métodos de Booch, Rumbaugh y Jacobson. 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. Estos elementos se agrupan en diagramas preestablecidos que corresponden a diferentes proyecciones del sistema. (29)

UML permite que se represente de manera semi-formal la estructura general del sistema y presenta como principales características las siguientes:

- Es un lenguaje consolidado, fácil de aprender y permite la documentación de todo el ciclo de desarrollo del sistema.
- Puede ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios o entre el equipo de desarrollo.
- 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 modelado de los sistemas.

A continuación se muestran algunas de las ventajas que proporciona usar este lenguaje de modelado en el desarrollo de nuestro sistema:

- Mejor entendimiento del riesgo del proyecto antes de construir el sistema
- Mejores tiempos totales de desarrollo
- Podremos especificar la estructura y el comportamiento del sistema y comunicarlo a todos los integrantes del proyecto
- Se documentarán las decisiones de la arquitectura del proyecto
- Se obtendrá el "plano" del sistema
- Mejor soporte al planeamiento y control del proyecto
- Un aumento en la calidad del desarrollo

- Reducción en los costos económicos

2.6 Herramienta CASE

La realización de un nuevo software requiere que las tareas sean organizadas y completadas en forma correcta y eficiente. Las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) fueron desarrolladas para automatizar esos procesos y facilitar las tareas de coordinación de los eventos que necesitan ser mejorados en el ciclo de desarrollo de software.

Las herramientas CASE son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Las mismas dan asistencia a los analistas, ingenieros de software y desarrolladores durante el desarrollo del software.

La herramienta CASE propuesta para el desarrollo de la plataforma de servicios a pozos es **Visual Paradigm**. Esta es una galardonada herramienta de modelado UML que soporta hasta la versión 2.x de UML, utilizada para el diseño de las aplicaciones visualmente, la generación de documentación, código fuente y las bases de datos. (30) Se encuentra respaldado por la licencia Berkeley Software Distribution (BSD) una licencia de software libre permisiva ya que permite el uso de código fuente en software no libre.

Dentro de las características que ofrece para el equipo de desarrollo se encuentran las siguientes:

- Capacidades de ingeniería directa (versión profesional) e inversa.
- Disponibilidad de múltiples versiones, para cada necesidad.
- Permite la exportación de los diagramas en formato de imagen.
- Permite generar código en múltiples lenguajes.
- Es una herramienta multiplataforma.

Esta herramienta brinda grandes ventajas como son::

- La facilita la comunicación entre el equipo de desarrollo ya que utiliza un lenguaje común. (UML)
- Disponibilidad de integrarse con los principales IDE.

- Disponibilidad en múltiples plataformas. Característica que otras no poseen.
- Permite la conexión a repositorios como el CVS y el Subversion.
- Gran usabilidad ya que los diagramas se agrupan por categorías y fomenta la organización. Es un software muy funcional y fácil de usar.

La herramienta CASE Visual Paradigm fue seleccionada por poseer estas ventajas y además por ser la herramienta con que más se encuentra familiarizado el equipo de desarrollo.

2.7 Lenguajes de Programación

Un lenguaje de programación es un lenguaje que puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Consiste en un conjunto de reglas sintácticas y semánticas que define su estructura y el significado de sus elementos.

Python

Se propone Python como lenguaje de programación principal para la implementación de la plataforma de servicios de aplicaciones.

Este lenguaje es orientado a objetos de propósito general, soporta herencia múltiple y polimorfismo, además es interpretado o de script, o sea, que se ejecuta utilizando un programa intermedio llamado intérprete en lugar de compilar su código. Se considera además un lenguaje multiparadigma, o sea, que así como C++ no fuerza a los programadores a adoptar un estilo particular de programación, permite usar estilos como los de Programación Orientada a Objetos (POO), Programación estructurada, funcional, orientada a aspectos, entre otros.

Los lenguajes de scripts se conocen con el nombre de middleware ya que son capaces de gestionar transferencias de datos entre aplicaciones o entre aplicaciones y el sistema operativo. Habitualmente se pueden ampliar mediante plug - ins o módulos (en el caso de Python) e incluso pueden ser empotrados en otras aplicaciones.

Posee un fuerte tipado dinámico, lo cual se refiere a que no es necesario declarar el tipo de dato que va a contener una determinada variable, sino que su tipo se determinará en tiempo de ejecución según el tipo

del valor al que se asigne, pero su tipo se mantiene de manera consistente si no hay una nueva asignación. (31)

El intérprete de Python está disponible en varias plataformas, UNIX, Solaris, Linux, DOS, Windows, OS/2, Mac OS, etc. Por lo que si no se utilizan librerías específicas de cada plataforma el programa podrá correr en todos estos sistemas sin grandes cambios.

Existen varias implementaciones distintas del intérprete, entre las más utilizadas se encuentran:

- CPython: Es la más utilizada, la más rápida y la más madura. Cuando se habla de Python normalmente se hace referencia a esta implementación. En este caso tanto el intérprete como los módulos están escritos en C.
- Jython es la implementación en Java de Python, mientras que IronPython es su contrapartida en C# (.NET). Su interés estriba en que utilizando estas implementaciones se pueden utilizar todas las librerías disponibles para los programadores de Java y .NET respectivamente.
- PyPy se trata como su nombre lo indica de una implementación en Python de Python.

La versión propuesta para el intérprete a utilizar en el desarrollo de la plataforma de servicios de aplicaciones será la de CPython 2.6

Entre las principales ventajas de Python se encuentran su sintaxis simple y clara, el código no es muy difícil de entender ya que es muy similar al pseudocódigo, por lo que llevar un programa del papel a la implementación es muy sencillo y los desarrolladores lo aprenden muy rápido, pero no solo eso, los programas en Python suelen ser entre 2 y 4 veces más cortos que sus equivalentes en Java, C, Pascal y otros. Posee una gran cantidad de librerías y módulos disponibles. Es de código libre bajo licencia GPL y posee muy buena documentación tanto en inglés como en español.

Este lenguaje será usado para desarrollar todos los componentes que requieran accesos frecuentes a la base de datos o que no requieran una respuesta relativamente inmediata del sistema. Se propone utilizar el estilo de código de Python pep8. Este estilo propone una buena organización del código además de hacerlo muy legible para los desarrolladores.

C++

El otro lenguaje de programación a utilizar en el desarrollo de la plataforma será C++, para cuando sea necesaria una respuesta relativamente rápida del sistema donde no sea necesario acceder a la base de datos y además para desarrollar las interfaces de usuario mediante el framework especificado.

Este es un lenguaje derivado del C que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos, por lo que se considera un lenguaje híbrido multiparadigma.

Al ser un lenguaje compilado se ahorra tiempo ya que no es necesario recompilar la aplicación completa al hacer un solo cambio. Además, esta característica permite vincular código C++ con código que se ha producido en otros idiomas, como el ensamblador o C. Algunas de las particularidades de C++ son la posibilidad de redefinir los operadores con la sobrecarga de los mismos, una forma de hacer polimorfismo y de poder crear nuevos tipos que se comporten como tipos fundamentales. Con este lenguaje se puede implementar la herencia simple y múltiple, se pueden definir clases abstractas y definir plantillas como mecanismo para implantar el paradigma de la programación genérica permitiendo que una clase o función trabaje con tipos de datos abstractos.

La principal ventaja que presenta este lenguaje sin restarle importancia a las anteriores es que el equipo de desarrollo tiene un vasto conocimiento de este lenguaje.

2.8 Ambiente de Desarrollo Integrado

Un ambiente de desarrollo integrado (Integrated Development Environment: IDE) es un programa compuesto por un conjunto de herramientas para un programador, puede dedicarse a uno o varios lenguajes de programación tales como C++, Java, C#, Python, Visual Basic, entre otros. 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).

El IDE propuesto para la implementación de la plataforma es **Eclipse**. Este es un entorno de desarrollo integrado de código abierto multiplataforma para desarrollar aplicaciones de cliente enriquecido. Eclipse

fue desarrollado originalmente por IBM pero ahora es desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios. Este IDE se encuentra bajo la Eclipse Public License (EPL) la cual es una licencia de software de código abierto.

Este IDE es una extensible plataforma de código abierto para desarrollar herramientas que a pesar de estar escrito mayoritariamente en Java se puede programar en un considerable número de lenguajes entre los que se encuentran los propuestos, Python y C++. El soporte para desarrollo en varios lenguajes es proveído por un componente enchufado o plug-in para proporcionar toda su funcionalidad al frente de la plataforma de cliente rico, a diferencia de otros entornos monolíticos donde las funcionalidades están todas incluidas, las necesite el usuario o no.

En el desarrollo de la plataforma se utilizará PyDEV, un plug-in para Eclipse que permite programar en Python; además de CDT, otro plug-in para Eclipse que permite programar en C++.

Dentro de las ventajas que posee este IDE se encuentran las herramientas y tecnologías que provee para el desarrollo de aplicaciones. A continuación se muestran algunas que serán útiles en el desarrollo de la plataforma de servicios a pozos:

- Posee herramientas de pruebas y medición de rendimientos para que los desarrolladores puedan monitorizar sus aplicaciones y hacerlas más productivas.
- Brinda soporte para tecnologías centradas en el manejo de datos.
- Posee herramientas para el desarrollo de aplicaciones destinadas a ser ejecutadas en dispositivos limitados en hardware.
- Brinda soporte para la conexión a repositorio mediante plug-in. (Subclipse, plug-in para Subversion)
- Es un IDE multiplataforma, entre las que se encuentran Windows y GNU/Linux.

2.9 Frameworks

Un framework es un esquema (un esqueleto, un patrón) para el desarrollo y la implementación de una aplicación, o sea, la forma de estructurar y normalizar la información de un modo conocido para poder

manejarla: almacenarla, recuperarla, etc. Pueden llegar al detalle de definir los nombres de ficheros, su estructura y las convenciones de programación. (32)

Existen innumerables frameworks, su elección está dada por el tipo de aplicación a desarrollar, el lenguaje de programación y otras tecnologías como bases de datos, sistemas operativos, etc. A continuación se describen los frameworks propuestos para el desarrollo de la plataforma.

QT

Qt4 es una plataforma de desarrollo que incluye clases, librerías y herramientas para la producción de aplicaciones de interfaz gráfica en C++ entre otras funcionalidades. Está distribuida bajo los términos de GNU Lesser General Public License, es software libre y de código abierto. Qt4 es multiplataforma e incluye soporte de nuevas tecnologías como OpenGL, XML, bases de datos, programación para redes y mucho más. Qt dispone de una amplia gama de herramientas que facilitan entre otras cosas la creación de formularios, botones y ventanas de dialogo con el uso del ratón. Además provee soporte para la internacionalización de su contenido.

Qt dispone de cuatro grandes ventajas:

- Qt es completamente gratuito para aplicaciones de código abierto aunque también se utiliza para aplicaciones comerciales.
- Las herramientas, librerías y clases están disponibles para casi todas las plataformas Unix y sus derivados (como Linux, MacOS X, Solaris, etc) como también para la familia Windows, por lo que una aplicación puede ser compilada y utilizada en cualquier plataforma sin necesidad de cambiar el código.
- Qt tiene una extensa librería con clases y herramientas para la creación de elegantes aplicaciones. De esta librería utilizaremos la herramienta QtDesigner para el diseño de las interfaces gráficas.
- El API de la biblioteca cuenta con métodos para acceder a bases de datos mediante SQL, así como uso de XML, gestión de hilos, soporte de red, una API multiplataforma unificada para la manipulación de archivos, de ficheros y de estructuras de datos tradicionales.

A continuación se muestran ejemplos de aplicaciones que fueron desarrolladas con QT:

- Adobe Photoshop Album
- Google Earth
- Mathematica
- Skype
- VLC Media Player

Además de esta librería se debe usar PyQT, que no es más que una adaptación o enlace de Python con el framework QT de Nokia y funciona en todas las plataformas soportadas por QT. De este enlace se debe usar la versión 4 la cual soporta QT versión 4 y superiores. Los enlaces se implementan como un conjunto de módulos de Python que contienen alrededor de 300 clases y 6000 funciones y métodos.

SQLAlchemy

SQLAlchemy es una herramienta de Python para el Mapeo Objeto Relacional (ORM del inglés Object Relational Mapper) que provee a los desarrolladores de aplicaciones un poder absoluto y flexibilidad sobre SQL. Se encuentra bajo la licencia del Instituto de Tecnología de Massachusetts (MIT), una licencia de software libre permisiva ya que permite el uso de código fuente en software no libre.

SQLAlchemy es muy fácil de usar y muy rápido en la devolución de los resultados. Este ORM no considera las bases de datos solo como una colección de tablas, las considera un motor de álgebra relacional y les posibilita a las clases ser mapeadas contra la base de datos de más de una manera, mediante las tablas, las consultas, uniones y otras.

Así, las relaciones de base de datos y modelos de objetos de dominio pueden ser disociadas limpiamente desde el principio, lo que permite a ambas partes desarrollar todo su potencial. Se considera más que un ORM ya que es una capa de abstracción de datos que permite la construcción y manipulación de datos de expresiones SQL. Ofrece resultados fácil de usar y muy rápido.

SQLAlchemy se puede integrar con gestores de bases de datos como PostgreSQL, SQLite, MySQL, Oracle, MS SQL, Firebird, Sybase y otros.

Este ORM organiza las operaciones pendientes tales como crear, insertar, actualizar y eliminar en colas, para ello realiza una clasificación de dependencias topológicas de todos los elementos modificados en la cola y agrupa todas las sentencias redundantes juntas. Por lo tanto, el mapeo de la base de datos y el diseño de clases está totalmente separado. Esto produce una eficiencia máxima y una transacción segura y reduce al mínimo las probabilidades de interbloqueo.

La construcción de las consultas basadas en funciones permite a las cláusulas SQL ser construidas por funciones y expresiones de Python. La gama completa de lo que es posible incluye expresiones booleanas, operadores, funciones, alias de tablas, sub consultas seleccionables, sentencias create, insert, update, delete, actualizaciones correlacionadas, cláusulas Exists correlacionadas, cláusulas Union, Inner y Outer Joins. (33)

2.10 Sistema Gestor de Base de Datos

Una base de datos es un “almacén” que permite guardar grandes cantidades de información de forma organizada para que luego se pueda encontrar y utilizar fácilmente. Se puede definir como un conjunto de información relacionada que se encuentra agrupada o estructurada.

Un Sistema Gestor de Base de Datos (SGBD) es un conjunto de programas que permiten crear y mantener una base de datos, asegurando su integridad, confidencialidad y seguridad. Por tanto, debe permitir:

- Definir una base de datos: especificar tipos, estructuras y restricciones de datos.
- Construir la base de datos: guardar los datos en algún medio controlado por el mismo SGBD
- Manipular la base de datos: realizar consultas, actualizarla, generar informes.

El SGBD propuesto para el desarrollo de la plataforma de servicios de aplicaciones de pozos es **PostgreSQL** en su versión 8.3. Este es un sistema de gestión de base de datos relacional orientada a objetos, multiplataforma y libre, publicado bajo la licencia Berkeley Software Distribution (BSD) una licencia de software libre permisiva ya que permite el uso de código fuente en software no libre. (34)

Como muchos otros proyectos de código abierto, el desarrollo de PostgreSQL no es manejado por una sola empresa sino que es dirigido por una comunidad de desarrolladores y organizaciones comerciales las cuales trabajan en su desarrollo. Dicha comunidad es denominada el PGDG (PostgreSQL Global Development Group).

Ofrece soporte para el lenguaje SQL92\SQL3, integridad de transacciones y extensibilidad de tipos de datos. Además, sirve de soporte a los lenguajes de programación más populares como PHP, C, C++, Java, Python, entre otros. El número de base de datos que puede contener es ilimitado. Posee una gran escalabilidad y rendimiento bajo grandes cargas de trabajo.

PostgreSQL incorpora cuatro conceptos adicionales básicos de manera tal que los usuarios puedan extender fácilmente el sistema, estos son los de clases, herencia, tipos y funciones. Además, con características como las restricciones (constraints), disparadores (triggers), reglas (rules) e integridad transaccional que aporta PostgreSQL se puede aportar potencia y flexibilidad adicional a este SGBD.

Estas características colocan a Postgres en la categoría de las Bases de Datos identificadas como objeto-relacionales.

Se puede concluir apuntando que posee grandes ventajas para el desarrollo de la plataforma de servicios a pozos tales como:

- Es un software libre.
- Es un sistema muy estable.
- Es un sistema multiplataforma.
- Soporta una alta concurrencia.
- Posee una gran escalabilidad.
- Posee un alto rendimiento bajo grandes cargas de trabajo.
- Es un lenguaje muy potente.
- Ofrece garantía de integridad en los datos.

2.11 Herramientas para el Control de Versiones

Subversion

Subversion es un sistema de control de versiones de código fuente abierto y libre. Es software libre bajo una licencia de tipo Apache/BSD. (35)

Subversion maneja ficheros y directorios a través del tiempo. Hay un árbol de ficheros en un repositorio central. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones; si se ha hecho un cambio incorrecto a los datos, simplemente se deshace. Subversion es un sistema general que puede ser usado para administrar cualquier conjunto de ficheros. Subversion proporciona:

Versionado de directorios: Implementa un sistema de ficheros versionado “virtual” que sigue los cambios sobre árboles de directorios completos a través del tiempo. Ambos, ficheros y directorios, se encuentran bajo el control de versiones.

Verdadero historial de versiones: Puede añadir, borrar, copiar, y renombrar ficheros y directorios. Y cada fichero nuevo añadido comienza con un historial nuevo, limpio y completamente suyo.

Envíos atómicos: Una colección cualquiera de modificaciones o bien entra por completo al repositorio, o bien no lo hace en absoluto. Esto permite a los desarrolladores construir y enviar los cambios como fragmentos lógicos e impide que ocurran problemas cuando sólo una parte de los cambios enviados lo hace con éxito.

Versionado de metadatos: Cada fichero y directorio tienen un conjunto de propiedades, claves y sus valores, asociados a él. Usted puede crear y almacenar cualquier par arbitrario de clave/valor que desee. Las propiedades son versionadas a través del tiempo, al igual que el contenido de los ficheros.

Manipulación consistente de datos: Expresa las diferencias del fichero usando un algoritmo de diferenciación binario, que funciona idénticamente con ficheros de texto (legibles para humanos) y ficheros binarios (ilegibles para humanos). Ambos tipos de ficheros son almacenados igualmente comprimidos en el repositorio, y las diferencias son transmitidas en ambas direcciones a través de la red.

Ramificación y etiquetado eficientes: El coste de ramificación y etiquetado no necesita ser proporcional al tamaño del proyecto. Crea ramas y etiquetas simplemente copiando el proyecto, usando un mecanismo similar al enlace duro. De este modo estas operaciones toman solamente una cantidad de tiempo pequeña y constante.

La herramienta además es compatible con los siguientes protocolos: http://; https://; svn://; svn+ssh://.

TortoiseSVN

Es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. Es software libre liberado bajo la licencia GNU GPL. (36) Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. El repositorio es prácticamente lo mismo que un servidor de ficheros ordinario, salvo que recuerda todos los cambios que se hayan hecho a sus ficheros y directorios. Esto permite que pueda recuperar versiones antiguas de sus ficheros y examinar la historia de cuándo y cómo cambiaron sus datos, y quién hizo el cambio. Es fácil de usar, permite ver el estado de los archivos desde el explorador de Windows y permite también crear gráficos de todas las revisiones asignadas. Este cliente será utilizado por los desarrolladores que escojan Windows como sistema operativo.

RaridSVN

Es un cliente gratuito de código abierto y multiplataforma para el sistema de control de versiones Subversion, liberado bajo la licencia GNU (GPL). (37) Provee una interfaz para Subversion muy fácil de usar para principiantes y muy flexible, suficiente como para aumentar la productividad para usuarios de Subversion con experiencia. Está completamente escrito en C++ por lo tanto es muy rápido y eficiente. Además, está disponible en varios lenguajes. Los ficheros al estar almacenados en un repositorio central se pueden recuperar versiones antiguas de esos ficheros y examinar la historia de cómo y cuándo

cambiaron sus datos, además de quien lo hizo. Este cliente será utilizado por los desarrolladores que escojan Linux como sistema operativo.

2.12 Conclusiones Parciales

En este capítulo se describieron las principales herramientas y tecnologías a utilizar en el desarrollo de la plataforma. Así como también se dan argumentos del porqué de esta selección. Se espera con estas decisiones satisfacer las necesidades del equipo de desarrollo para la implementación de la plataforma de servicios a pozos del CEINPET. Además, se propusieron los estilos arquitectónicos y patrones de diseño que darán sustento a la arquitectura de la plataforma.

CAPÍTULO 3: Presentación y Evaluación de la Solución Propuesta.

3.1 Introducción

En este capítulo se hará una propuesta de las metas y restricciones arquitectónicas para la plataforma de servicios a pozos donde se proponen los requisitos funcionales y no funcionales de la misma. Se presentará la línea base de la arquitectura propuesta describiendo sus aspectos más representativos. Se mostrarán y detallarán las vistas de la arquitectura y se evaluará la arquitectura de software propuesta obteniendo los posibles riesgos producidos con la evaluación de la misma.

3.2 Representación Arquitectónica

La plataforma de servicios a pozos consiste en la integración de un conjunto de aplicaciones relacionadas con la industria del petróleo. Estas aplicaciones gestionan la información referente a yacimientos y pozos del crudo, además de analizar y evaluar propiedades de los mismos. Con la plataforma se centralizarán todas estas aplicaciones en un mismo sistema y se logrará un rápido acceso a los datos gestionados.

Para la implementación de la plataforma se propone hacer uso del estilo arquitectónico orientado a componentes de la familia de estilos de llamada y retorno. Se propone para su desarrollo los lenguajes de programación Python y C++, haciendo uso de los frameworks QT y SQLAlchemy para el desarrollo de las interfaces gráficas y el mapeo objeto relacional respectivamente.

Para confeccionar la representación de la arquitectura se tomó como guía el proceso de desarrollo de RUP, que define la representación arquitectónica mediante las 4+1 vistas arquitectónicas. Los diagramas fueron modelados con la herramienta CASE Visual Paradigm, usando el Lenguaje Unificado de Modelado.

3.3 Metas y Restricciones Arquitectónicas

El objetivo de la ingeniería del software es el desarrollo de sistemas adecuados a las necesidades del cliente, pero también ajustados a otros criterios, como el modelo de negocio, los recursos disponibles y el tiempo de entrega. Es obvio que la ingeniería del software no solo ha de cumplir con la funcionalidad

(escribir código ajustado a los requisitos funcionales) sino también con las cualidades suplementarias (requisitos no funcionales) o de lo contrario no cumplirá con su misión: Desarrollar el software que se necesita en el momento y condiciones que se tienen disponibles; o dicho de otra manera, desarrollar software de calidad.

Requisitos No Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Son importantes para que clientes y desarrolladores puedan valorar las características no funcionales del producto.

Por lo tanto, un requisito no funcional es una característica requerida del sistema, del proceso de desarrollo, del servicio prestado o de cualquier otro aspecto del desarrollo, que señala una restricción del mismo.

A continuación se enuncian las categorías y su descripción de los requisitos no funcionales propuestos para la plataforma de servicios de aplicaciones:

Requisitos de Software

Estaciones de Trabajo para el Equipo de Desarrollo

Las computadoras que se utilizarán como puestos de trabajo para el equipo de desarrollo deberán tener instalado:

- Sistema operativo Windows XP o superior, o sistema operativo GNU-Linux en cualquier distribución
- Intérprete Python 2.6
- Para el Mapeo Objeto Relacional (ORM) SQLAlchemy en su versión 0.5
- Para la implementación de las interfaces de usuario el framework QT en su versión 4.5
- Para enlazar la biblioteca de QT y usarla en Python el binding PyQT 4
- Como Entorno de Desarrollo Integrado (IDE) Eclipse en su versión 3.5

- Como herramienta CASE Visual Paradigm en su versión 6.4
- Como Sistema Gestor de Base de Datos (SGBD) PostgreSQL en su versión 8.3

Estaciones de Trabajo para el Usuario Final

Las computadoras que se utilizarán como puesto de trabajo para el usuario final deberán tener instalado:

- Sistema operativo Windows 2000 o superior, o sistema operativo GNU-Linux en cualquier distribución
- Intérprete Python 2.6
- PyQT 4
- SQLAlchemy 0.5

Servidor de Base de Datos

La computadora que se utilizará como servidor de base de datos deberá tener instalado:

- Sistema operativo Windows 2000 o superior, o sistema operativo GNU-Linux en cualquier distribución (recomendado)
- Sistema gestor de base de datos PostgreSQL.

Requisitos de Hardware

Estaciones de Trabajo para el Equipo de Desarrollo

Las computadoras que se utilizarán como puesto de trabajo para el equipo de desarrollo deberán contar con:

Requisitos mínimos:

- Periféricos Teclado y Mouse PS2 o USB
- 512 MB de memoria RAM
- CPU Pentium IV a 1.00 GHz

- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 10 Mb/s
- 9 GB de espacio libre en disco para Windows
- 5 GB de espacio libre en disco para Linux

Requisitos recomendados:

- Periféricos Teclado y Mouse PS2 o USB
- 1 GB de memoria RAM
- CPU Pentium IV a 2.00 GHz o superior
- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 100 Mb/s
- Más de 9 GB de espacio libre en disco

Estaciones de Trabajo para el Usuario Final

Las computadoras que se utilizarán como puesto de trabajo para el usuario final deberán contar con:

Requisitos mínimos:

- Periféricos Teclado y Mouse PS2 o USB
- 256 MB de memoria RAM
- CPU Pentium III a 1.00 GHz
- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 10 Mb/s
- 7 GB de espacio libre en disco para Windows
- 4 GB de espacio libre en disco para Linux
- Impresora de cualquier tipo y marca

Requisitos recomendados:

- Periféricos Teclado y Mouse PS2 o USB
- 512 MB de memoria RAM
- CPU Pentium IV a 2.00 GHz
- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 100 Mb/s

- Más de 7 GB de espacio libre en disco
- Impresora de cualquier tipo y marca

Servidor de Base de Datos

La computadora que se utilizará como servidor de base de datos deberá contar con:

Requisitos mínimos:

- 1 GB de memoria RAM.
- Disco duro de 80 GB.
- CPU Pentium IV a 1.00 GHz.
- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 10 Mb/s

Requisitos recomendados:

- 2 GB de memoria RAM.
- Disco duro de más de 80 GB.
- CPU Pentium IV a 2.00 GHz o superior.
- Adaptador de red, ya sea una tarjeta o integrado en la placa madre con velocidad de 100 Mb/s

Restricciones en el Diseño y la Implementación

El sistema será una aplicación escritorio donde el diseño de los componentes y el sistema en general se realizará bajo los principios y uso de la Programación Orientada a Objetos (POO) empleando:

- Los lenguajes de programación Python y C++
- Como estilo de código para Python pep8 el cual se adjunta a este documento
- Como estilo de código para C++ uno definido por los líderes del proyecto que se adjunta a este documento
- El framework QT para la implementación de las interfaces de usuario
- El Mapeador Objeto-Relacional (ORM) SQLAlchemy para la gestión de los datos con la base de datos

Las aplicaciones a integrar se desarrollarán en forma de componentes para integrar como plug-in con el objetivo de la reutilización, fácil instalación y fácil actualización.

Requisitos de Apariencia o Interfaz Externa

- La interfaz gráfica de la aplicación debe ser amigable y sencilla y proveer de forma ordenada y prolija las funcionalidades del sistema.
- Incluir atajos para los elementos del menú principal y para otros elementos usados frecuentemente.
- La aplicación debe verse correctamente en todas las resoluciones y profundidades de color.
- Utilizar en la aplicación fuentes básicas que vengan con el sistema operativo y que sean relativamente grandes.
- La interfaz gráfica debe diseñarse predominando los colores, verde por el logo de la empresa y gris para combinar preferentemente.

Requisitos de Seguridad

Los usuarios del sistema deben autenticarse antes de realizar cualquier actividad para garantizar que las funcionalidades del sistema se muestren de acuerdo con los permisos del o los roles asignados al usuario que esté autenticado.

Requisitos de Disponibilidad

La disponibilidad de la información del sistema al personal autorizado se le garantizará después de su acceso en todo momento, mientras no haya un fallo (fluido eléctrico, conexión de red u otros). Los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán al personal para obtener los datos deseados en un momento dado.

Requisitos de Usabilidad

La plataforma está concebida para ser usada por los especialistas del CEINPET u otros de la rama del petróleo con conocimientos del negocio que estén calificados para gestionar e interpretar los resultados de las aplicaciones que se tengan instaladas.

- La plataforma será utilizada por cualquiera del equipo de desarrollo con conocimientos medios o avanzados de programación en los lenguajes Python o C++ y diseño de base de datos con PostgreSQL para futuras actualizaciones.
- La documentación de la arquitectura y la documentación generada por el sistema en general, deberá ser realizada de manera detallada y clara tal que pueda ser comprensible para futuros desarrollos.
- El sistema una vez implementado debe contar con un manual de usuario.

Requisitos de Soporte

- Los líderes del proyecto en la medida que se vayan liberando las versiones del producto deberán ir al CEINPET a instalar y configurar el mismo.
- Los líderes del proyecto gestionarán con el CEINPET el mantenimiento del software después del despliegue.

Requisitos de Confiabilidad

Las salidas del sistema deben de ser exactas evitando los errores ya que con las aplicaciones se evaluarán las condiciones del terreno, yacimiento, pozo u otros factores y cualquier mala decisión tomada por los especialistas a partir de un dato inconsistente afectaría la producción del yacimiento y a su vez la economía del país.

Requisitos de Portabilidad

El sistema debe ser implementado de tal forma que pueda ser ejecutado en cualquier ambiente de sistema operativo que se esté usando.

Requisitos de Escalabilidad

El sistema debe ser escalable de modo que futuras aplicaciones desarrolladas sobre esta línea para el CEINPET puedan ser integradas como plug-in en la plataforma.

3.4 Organigrama de la Arquitectura

La plataforma de servicios de aplicaciones a desarrollar para el CEINPET se encuentra dentro de la categoría de software de gestión. La misma debe poseer una base de datos que permita el almacenamiento de información para posibilitar el funcionamiento de las aplicaciones que de ella dependan y para la toma de decisiones por parte de los especialistas que conlleven a resultados óptimos en la industria del petróleo. Este sistema debe poseer un alto grado de confiabilidad y garantía ya que sus resultados repercuten en las producciones de petróleo de las empresas de nuestro país y por ende en su economía.

Esta plataforma contará con aplicaciones que gestionen información de los yacimientos del crudo y los pozos que existan en los mismos. Las aplicaciones a integrar serán las relacionadas con el/la:

- Gestión de lechadas para la cementación de pozos petroleros (ANACEM)
- Gestión de información de pérdida de circulación durante la perforación de pozos petroleros (GESCOM)
- Evaluación y control del trabajo con los fluidos de perforación (MUDMAN)
- Identificación de aguas en pozos petroleros (SIAPP)
- Gestión de la información de muestras de núcleos (SIGNUC)
- Diagnóstico de daños en pozos petroleros (SISDAP)

Por estas razones el estilo arquitectónico propuesto para el desarrollo de la plataforma de servicios de aplicaciones fue el de orientado a componentes. Cada aplicación de las mencionadas anteriormente se implementará como un componente con interfaces bien definidas a integrar en la plataforma y cada uno de ellos podrá implementar en sí mismo la arquitectura que más le acomode. Para finalmente integrarse como un plug-in en la plataforma de servicios de aplicaciones.

Por lo tanto, como macro-arquitectura se propone la de orientada a componentes y como arquitectura específica para cada uno de ellos la seleccionada más factible para cada aplicación.

Solo que cada componente proveerá una interfaz con las funcionalidades que brinda, las cuales serán gestionadas desde la plataforma para ser mostradas en el menú principal. Desde donde el cliente accederá a las funcionalidades propias de cada aplicación integrada a la plataforma.

Por ejemplo MUDMAN, la cual se desarrollará para la primera versión y en su implementación se utilizará el estilo arquitectónico en capas, específicamente, tres capas como se muestra a continuación:

Figura 1 Estilo arquitectónico 3 capas.

Capa de presentación: Es la que ve el usuario (también se le denomina “capa de usuario”), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). Esta capa se comunica únicamente con la capa de negocio.

En la capa de presentación se debe aplicar el estilo arquitectónico Modelo-Vista-Controlador, propuesto por los desarrolladores del framework QT para el desarrollo de las interfaces de usuario. En la Vista se encontrarán todas las interfaces de usuario. En el Controlador se encontrarán las clases de lógica de interfaz necesarias para manejar eventos u otros aspectos necesarios para mostrar la información al usuario. En el Modelo se encontrarán un conjunto de datos agrupados en listas para ser mostrados en la Vista.

Capa de negocio: Es donde residen las funcionalidades que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él.

En la capa de lógica de negocio no se propone la utilización de ningún framework por lo que se contará solamente con las entidades del negocio.

Capa de datos: Es donde residen los datos y es la encargada de acceder a los mismos. Se accede al gestor de bases de datos y se realiza todo el almacenamiento de datos, se reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Para la gestión de los datos en la capa de acceso a datos se propone utilizar el ORM SQLAlchemy, el cual mapeará las tablas de la base de datos en clases para conformar el paquete Domain. Además, en un paquete DAO (Data Access Object) estarán las entidades necesarias para la creación, obtención, actualización y eliminación (CRUD: Create, Retriave, Update, Delete) de datos de la base de datos.

Por lo tanto, así quedaría organizado el sistema:

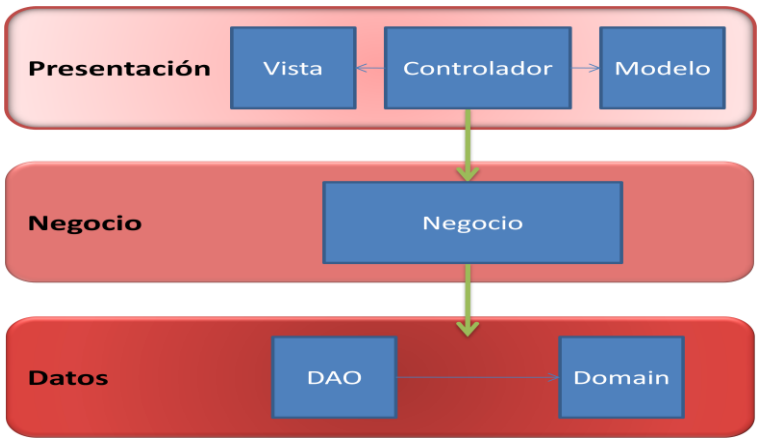

Figura 2 Representación arquitectónica de MUDMAN.

3.5 Vistas de la Arquitectura

Con el objetivo de lograr una mejor comprensión del sistema, organizar el desarrollo, fomentar la reutilización, contribuyendo de esta forma a una evolución del sistema más rápida y eficaz; se diseñan las vistas de la arquitectura definidas por la metodología RUP. Utilizando las vistas arquitectónicas, que muestran las diferentes características del sistema se puede capacitar a los desarrolladores, clientes y otros usuarios en la compresión al detalle del sistema propuesto, facilitando su participación en el mismo.

La representación de la arquitectura propuesta se hará utilizando las vistas definidas por RUP:

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

Vista Lógica

En esta vista se describen los paquetes más abstractos que conforman el sistema. Esta vista apoya principalmente a los requisitos funcionales, o sea, lo que el sistema debe brindar en forma de servicios a sus usuarios. Esta descomposición no sólo se hace para potenciar el análisis funcional, sino también sirve para identificar mecanismos y elementos de diseño comunes a diversas partes del sistema.

La siguiente figura muestra la división del sistema en subsistemas. Esto facilita el mantenimiento y la reusabilidad de la plataforma. Pues cualquier cambio en el negocio, solo afectaría al subsistema al cual pertenece la responsabilidad. Facilita y organiza el trabajo en equipo, dejando abierta a consideración la opción de desarrollar en paralelo y garantizar una terminación ágil y eficaz de la plataforma.

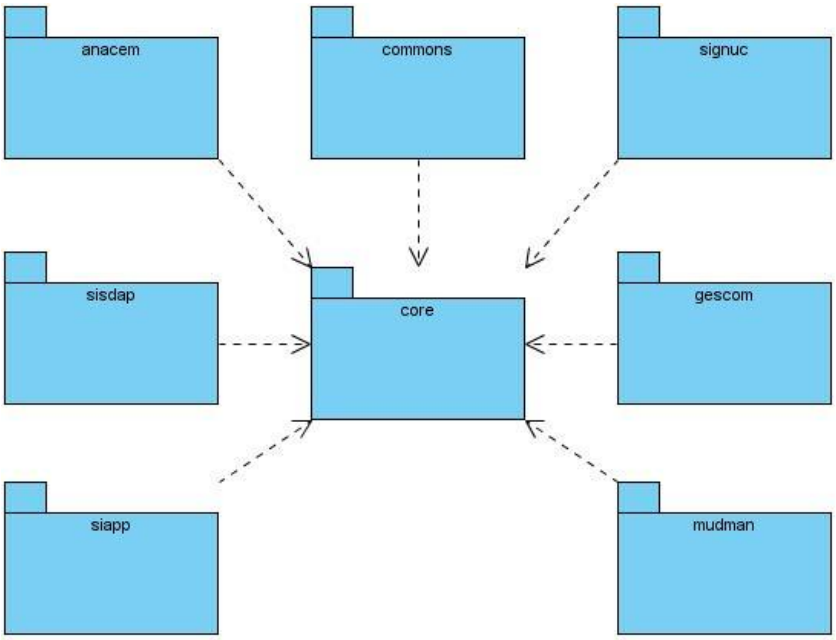

Figura 3 Principales subsistemas identificados.

Los principales subsistemas identificados son:

- anacem: Contiene la realización de los casos de uso (CU) referentes a la gestión de lechadas para la cementación de pozos petroleros
- gescom: Contiene la realización de los CU referentes a la gestión de información de pérdida de circulación durante la perforación de pozos petroleros
- mudman: Contiene la realización de los CU referentes a la evaluación y control del trabajo con los fluidos de perforación
- siapp: Contiene la realización de los CU referentes a la identificación de aguas en pozos petroleros
- signuc: Contiene la realización de los CU referentes a la gestión de la información de muestras de núcleos
- sisdap: Contiene la realización de los CU referentes al diagnóstico de daños en pozos petroleros
- commons: Contiene la realización de los CU referentes a las funcionalidades comunes que tengan las aplicaciones de la plataforma, ya sea la gestión de pozos o yacimientos del crudo, entre otros. Su estructura es la siguiente:

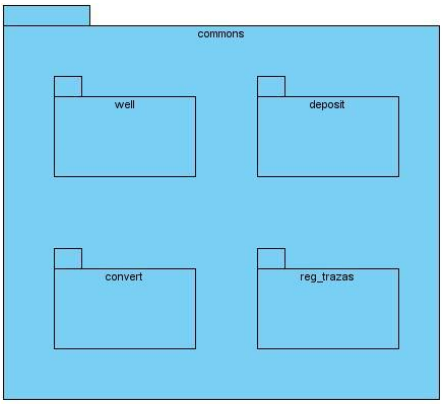

Figura 4 Subsistema commons de la plataforma.

Los principales paquetes identificados son:

- well: Contiene la realización del CU referente a la gestión de pozos.
- deposit: Contiene la realización del CU referente a la gestión de yacimientos.
- convert: Contiene la realización del CU referente a la conversión de unidades de medida.
- reg_trazas: Contiene la realización del CU referente al registro de las trazas de los usuarios.

➢ core: Contiene la realización de los CU referentes a la gestión de usuarios y roles, los permisos de cada uno, entre otros. Los mismos hacen que funcione de forma segura y eficiente cada una de las aplicaciones de la plataforma. Su estructura es la siguiente:

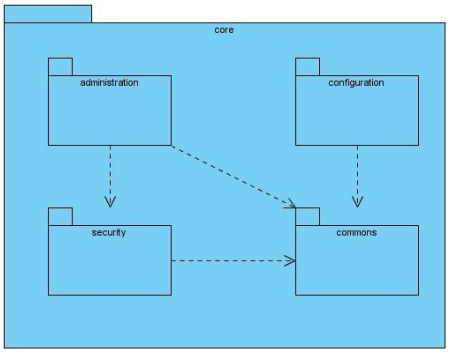

Figura 5 Subsistema core

Los principales componentes identificados dentro del core son:

- administration: Contiene la realización de los CU correspondiente con la gestión de usuarios y roles del sistema
- configuration: Contiene la realización del CU correspondientes a la administración de plug-in.
- security: Contiene la realización del CU correspondiente con la asignación de permisos a los usuarios y roles del sistema.
- commons: Este componente está conformado por paquetes que poseen funcionalidades comunes para cualquier sistema como la conexión a la base de datos.

Vista de Implementación

La vista de implementación proporciona una descripción de los principales subsistemas de componentes de la aplicación. Esta vista muestra las organizaciones y dependencias lógicas entre componentes software, sean estos componentes de código fuente, binarios o ejecutables. En este diagrama se tiene en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Con la representación de esta vista se ayuda a los desarrolladores a visualizar el camino de la implementación y permite tomar decisiones respecto a las tareas del desarrollo.

La figura muestra los paquetes definidos para cada uno de los subsistemas y componentes del sistema:

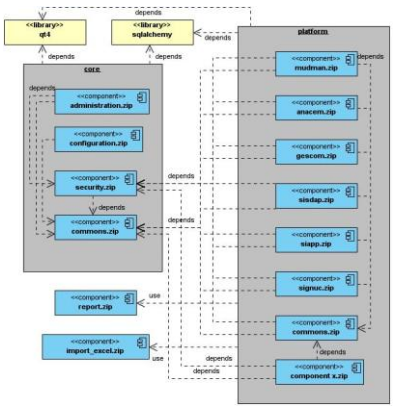

Figura 6 Vista de implementación

A continuación se muestra como se implementan los componentes físicos del sistema agrupados en paquetes:

- core: Contiene los componentes fundamentales para que funcionen de forma segura y eficiente cada una de las aplicaciones
- platform: Contiene cada una de las aplicaciones a integrar en forma de componentes así como un componente commons donde se implementan las funcionalidades comunes de las aplicaciones
- qt4: Librería usada para el desarrollo de las interfaces gráficas
- sqlalchemy: Librería usada para el mapeo objeto relacional
- report: Componente desarrollado anteriormente para generar reportes
- import_excel: Componente a desarrollar para la importación de documentos Excel

Vista de Despliegue

Mediante esta vista se describe el mapeo del software en el hardware y se reflejan los aspectos de distribución. Esta vista es un conjunto de nodos unidos por conexiones de comunicación. Un nodo puede contener instancias de componentes software, objetos y procesos. Así quedaría la Vista de Despliegue:

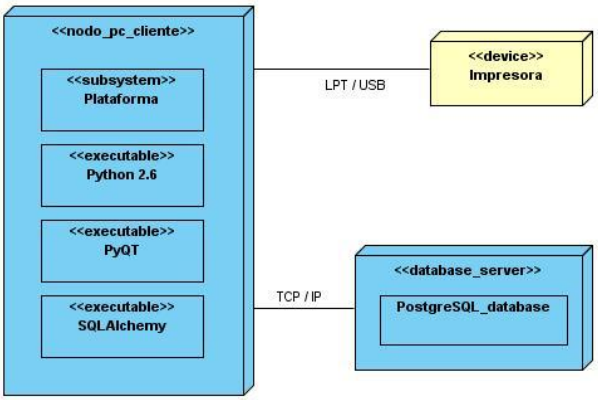

Figura 7 Vista de despliegue

A continuación se muestra la descripción de cada uno de los nodos y dispositivos propuestos en la figura anterior:

- nodo_pc_cliente: Constituye un nodo para la utilización de la plataforma por el cliente. Este nodo contendrá la plataforma de servicios a pozos como subsistema, el intérprete de Python versión 2.6, el PyQT 4 y el SQLAlchemy 0.5
- database_server: Constituye un nodo servidor de base de datos central en el cual se encuentra la información centralizada referente a cada una de las aplicaciones. Este nodo contendrá el sistema gestor de base de datos PostgreSQL v8.3
- Dispositivo Impresora: Satisface los requisitos de los clientes de impresión de reportes generados por la aplicación. La forma de conectarse al dispositivo es por puerto USB o en Paralelo lo que no es significativo para la aplicación, aunque influye en la rapidez de la impresión pero no constituye una exigencia del cliente

Elementos e interfaces de comunicación

Para el despliegue del sistema el cliente ya tiene establecido la configuración de las redes de datos en la empresa. Por lo que el sistema solamente haría uso de las mismas para su ejecución utilizando para la conexión entre el Nodo_PC_Cliente y el Nodo_DataBase_Server el siguiente protocolo:

Protocolo TCP/IP

TCP/IP no es un único protocolo, sino un conjunto de protocolos que cubren los distintos niveles del modelo OSI. Los dos protocolos más importantes son el TCP y el IP, que son los que dan nombre al conjunto. TCP/IP es compatible con cualquier sistema operativo y con cualquier tipo de hardware.

Una de las funciones y ventajas principales del TCP/IP es proporcionar una abstracción del medio, de forma que sea posible el intercambio de información entre medios diferentes y tecnologías que inicialmente son incompatibles.

Vista de Casos de Uso del Sistema

Los Casos de Uso (CU) son una técnica para especificar el comportamiento y responder a las necesidades del sistema. La vista de CU brinda información de escenarios y CU arquitectónicamente significativos o críticos con funcionalidades imprescindibles para el sistema. Estos CU son aquellos que

sirven para validar la arquitectura propuesta y describen las funcionalidades imprescindibles para el sistema. Esta vista se utiliza como entrada al hacer la planificación de lo que deba desarrollarse dentro de una iteración.

He aquí algunos conceptos esenciales para comprender esta vista:

Actores: Un actor representa un conjunto coherente de roles que los usuarios de casos de usos desempeñan cuando interaccionan con estos casos de uso.

Casos de Uso (CU): 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.

A continuación se muestra la clasificación de los CU del sistema:

Críticos	Secundarios	Auxiliares	Opcionales
Gestionar información muestra de agua	Autenticar	Imprimir reportes	Gestionar plug-in
Gestionar información muestra de núcleos	Convertir unidades de medida	Exportar documentos	Internacionalización
Gestionar información pérdida de circulación	Gestionar roles		
Diagnosticar daños	Gestionar usuario		
Gestionar información lechada cementación	Gestionar trazas		
Gestionar información			

fluidos de perforación			
Gestionar yacimiento			
Gestionar pozo			
Generar reporte			
Graficar información			
Importar documento Excel			

Tabla 1 Clasificación de los CU del sistema.

Los casos de usos que se presentan y comentan a continuación son la base para el posterior funcionamiento del sistema, no se puede realizar ninguna de las operaciones propuestas si no se cuenta con la base conceptual para realizarlas. A partir de estas funcionalidades, se puede comprobar que la arquitectura funciona para los elementos fundamentales, además de su comportamiento estable.

Sería notable aclarar que el cliente decidió el desarrollo para la primera versión de la aplicación MUDMAN por gestionar un gran cúmulo de información, posteriormente se integrarán las demás aplicaciones. Por lo tanto, con el desarrollo del proyecto hasta este momento se proponen para la Vista de CU del sistema los CU significativos referentes a esta aplicación donde el actor que los inicia va a ser el Analista de Fluidos. A continuación se proponen los CU críticos para esta versión:

- Gestionar pozo
- Gestionar yacimiento
- Graficar información
- Generar reporte
- Gestionar información fluidos de perforación

Así quedaría la Vista de CU del sistema:

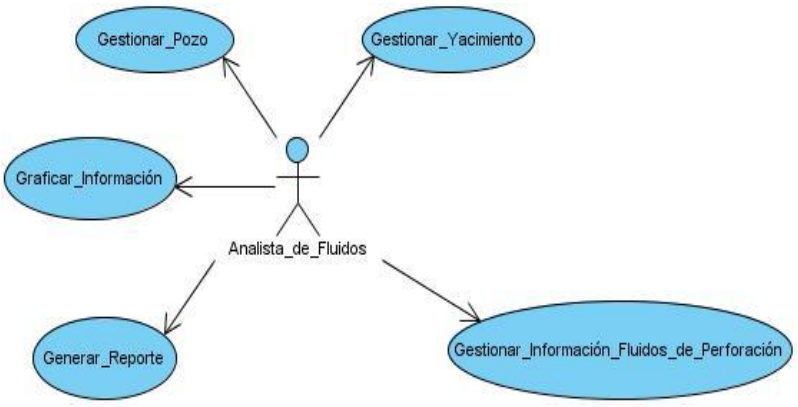

Figura 8 Vista de CU del sistema

Vista de Procesos

Esta vista suministra una base para la comprensión de la organización de los procesos del sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos o hilos concurrentes.

En este caso la solución propuesta no presenta procesos concurrentes por tanto no se ha hecho representación de esta vista.

3.6 Evaluación de la Arquitectura de Software

Evaluar una arquitectura de software sirve para prevenir todos los posibles problemas de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la arquitectura diseñada para el sistema.

Cuanto más temprano se encuentre un problema en un proyecto de software, mejor. El costo de arreglar un error durante las fases tempranas, es mucho menor al costo de arreglar ese mismo error cuando ya sea tarde. Dado que la arquitectura es un producto temprano de la fase de diseño, esta tiene un profundo efecto en el sistema y en el proyecto. Una mala arquitectura puede llevar a un proyecto al fracaso. Es

mejor cambiar la arquitectura antes que otros artefactos, que están basados en ella, se estabilicen. Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres.

Una evaluación de una Arquitectura de Software no te da un SI o un NO, si es buena o mala, o una calificación. Te dice dónde está el riesgo, es decir fortalezas y debilidades identificadas de la Arquitectura de Software. (38)

La evaluación de las arquitecturas de software puede ser realizada mediante el uso de diversas técnicas y métodos. Este conjunto de técnicas y métodos se basan fundamentalmente en la medida del cumplimiento de los atributos de calidad requeridos para un sistema de software, ya sea, evaluando una arquitectura de forma integral, o a través de la evaluación de arquitecturas candidatas. (39)

Técnicas de Evaluación de Arquitecturas de Software

Las técnicas existentes para evaluar la arquitectura se pueden clasificar en dos grupos, cualitativas y cuantitativas:

Entre las técnicas de evaluación cualitativas se encuentran:

- Escenarios
- Cuestionarios
- Listas de verificación

Estas técnicas normalmente son usadas cuando la arquitectura se encuentra en construcción en fases tempranas del desarrollo. Estas son técnicas que requieren poca información detallada y pueden conducir a resultados relativamente precisos. Además aplicarlas no acarrea un alto costo para el equipo de desarrollo, por lo que son las más usadas por los arquitectos.

Entre las técnicas de evaluación cuantitativas se pueden emplear:

- Métricas
- Simulación
- Experimentos

- Prototipos
- Modelos matemáticos

Estas técnicas normalmente son usadas cuando la arquitectura ya ha sido implantada, o implementada. Están encaminadas a obtener valores para la toma de decisiones en cuanto a los atributos de calidad. Son técnicas que aplicarlas conlleva a un alto costo para el equipo de desarrollo por lo que no son tan utilizadas por los arquitectos.

Métodos de Evaluación de Arquitecturas de Software

Para la evaluación de arquitecturas de software también son utilizados métodos de evaluación. Los cuales sirven de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos o riesgos que pueda presentar una arquitectura de software y sus soluciones.

Entre los métodos más conocidos y utilizados están:

- **Método de Análisis de Arquitecturas de Software** (Software Architecture Analysis Method, **SAAM**) Este método está basado en escenarios y permite evaluar una arquitectura o comparar varias. Normalmente es usado cuando el atributo de calidad Modificabilidad es el de mayor interés.
- **Método de Análisis de Acuerdos de Arquitectura** (Architecture Trade-off Analysis Method, **ATAM**) Este método está basado en escenarios y no solo evalúa atributos de calidad, también especifica cómo pueden interactuar entre ellos. Normalmente es usado cuando los atributos de calidad rendimiento y confiabilidad son los de mayor interés.
- **Método de Análisis de Diseños Parciales** (Active Reviews for Intermediate Designs, **ARID**) Este método es usado para la evaluación de diseños preliminares en etapas tempranas del desarrollo y normalmente es usado para evaluar la factibilidad de la arquitectura.

Resultados de la Evaluación de una Arquitectura de Software

Generalmente, la evaluación de la arquitectura ocurre después que esta ha sido especificada, pero antes que empiece la implementación. En un proceso iterativo e incremental, la evaluación se puede realizar al final de cada ciclo. Sin embargo, uno de los atractivos de la evaluación de arquitecturas es que se puede

efectuar en cualquier etapa de la vida de una arquitectura. En particular, existen dos variaciones útiles: temprana y tardía.

En cualquiera de estas dos variantes se produce como resultado en la evaluación de una arquitectura la captura y priorización de las metas que la arquitectura debe cumplir para poder ser considerada adecuada. La evaluación de una arquitectura no produce resultados cuantitativos. La evaluación ayuda a encontrar debilidades. Es proveer un espíritu de mitigación de riesgos. Las salidas de una evaluación arquitectónica son información e ideas sobre la arquitectura. Pudieran ser:

- Lista priorizada de los atributos de calidad requeridos
- Riesgos y no riesgos

El mayor beneficio que brinda la evaluación de una arquitectura, es que descubre los problemas que si se hubiesen dejado sin descubrir, habría sido mucho más costoso corregirlos luego. En breve, una evaluación produce una mejor arquitectura.

Evaluación de la Arquitectura de Software Propuesta

Para la evaluación de la arquitectura de software en cuestión el método más conveniente a utilizar es el **ATAM** (Método de Análisis de Acuerdos de Arquitectura), este método es considerado el más completo, porque revela la forma en que una arquitectura específica satisface ciertos atributos de calidad y provee una visión de cómo los atributos de calidad interactúan con otros, además utiliza la técnica de evaluación basada en escenarios poco costosa para el equipo de desarrollo.

Aclarar además que por la calidad de software estar definida como el grado en el cual el software posee una combinación deseada de atributos. Tales atributos son requerimientos adicionales del sistema que hacen referencia a características que éste debe satisfacer, diferentes de los requerimientos funcionales. Estas características o atributos se conocen con el nombre de atributos de calidad, los cuales se definen como las propiedades de un servicio que presta el sistema a sus usuarios.

Como principal desventaja presenta que solo es aplicable en cualquier momento después de que el diseño de la arquitectura ha sido establecido, como es el caso. Por lo que no es útil a la hora de elegir

entre arquitecturas candidatas, acción muchas veces necesarias en equipos de poca experiencia. El cual no es el caso, ya que arquitecturas similares han sido aplicadas anteriormente por el equipo de desarrollo con resultados satisfactorios.

Para la evaluación de la arquitectura de software propuesta fueron identificados los siguientes atributos de calidad como de alta prioridad:

Requisitos de Escalabilidad

El sistema debe ser escalable de modo que futuras aplicaciones desarrolladas sobre esta línea para el CEINPET puedan ser integradas como plug-in en la plataforma.

Requisitos de Confiabilidad

Las salidas del sistema deben de ser exactas evitando los errores ya que con las aplicaciones se evaluarán las condiciones del terreno, yacimiento, pozo u otros factores y cualquier mala decisión tomada por los especialistas a partir de un dato inconsistente afectará la producción del yacimiento y a su vez la economía del país.

Requisitos de Disponibilidad

La disponibilidad de la información del sistema al personal autorizado se le garantizará después de su acceso en todo momento, mientras no haya un fallo (fluido eléctrico, conexión de red u otros). Los dispositivos o mecanismos utilizados para lograr la seguridad no ocultarán o retrasarán al personal para obtener los datos deseados en un momento dado.

Los siguientes fueron identificados como importantes, pero con menos prioridad:

Requisitos de Seguridad

Los usuarios del sistema deben autenticarse antes de realizar cualquier actividad para garantizar que las funcionalidades del sistema se muestren de acuerdo con los permisos del o los roles asignados al usuario que esté autenticado.

Requisitos de Portabilidad

El sistema debe ser implementado de tal forma que pueda ser ejecutado en cualquier plataforma (sistema operativo) que se esté usando.

A partir de estos atributos de calidad se seleccionaron los siguientes escenarios:

Atributo de Calidad	Escenario
Escalabilidad	El desarrollo de una nueva aplicación para el CEINPET debe ser integrada en la plataforma
Confiabilidad	Con una de las aplicaciones de la plataforma se debe calcular un dato decisivo para la toma de decisiones de los especialistas
Disponibilidad	La plataforma está disponible 24 horas al día los 7 días de la semana para que los usuarios puedan acceder a la información
Seguridad	El analista de fluidos solo tendrá acceso a la aplicación para la gestión de información de este tema
Portabilidad	Se debe sustituir el sistema operativo. Se debe cambiar el gestor de bases de datos. Se debe trasladar la plataforma a otra estación de trabajo.

Tabla 2 Escenarios y atributos de calidad

A continuación se procede a analizar cada uno de los escenarios.

- Desarrollar una nueva aplicación que deba ser integrada en la plataforma mientras provea las interfaces necesarias para su integración es simple. Esto consiste solo en utilizar la interfaz que provea la aplicación e integrarla como un plug-in. La escalabilidad del sistema se registra como un

punto de sensibilidad, que afecta positivamente este atributo. Se identifica como un punto de **no riesgo**.

- Que las salidas del sistema sean exactas y garantizar la correcta toma de decisiones por parte de los especialistas no se puede garantizar con la arquitectura de la plataforma. Las salidas del sistema están basadas en cálculos y leyes matemáticas y de otras disciplinas que algún error en esos cálculos podría afectar algún dato y así proveer un resultado que no fuese confiable para los especialistas. Este es un punto de **riesgo** ya que no se puede garantizar que los resultados sean confiables desde la arquitectura, por lo que no es dependiente de la arquitectura.
- Que el sistema esté disponible las 24 horas, 7 días a la semana para que los usuarios puedan acceder a la información no se garantiza con la arquitectura ya que la disponibilidad se refiere a la falla del sistema y sus consecuencias pero pueden existir factores externos que afecten este atributo como pudieran ser la falla de fluido eléctrico o conexión de red, entre otros. Este es un punto de **riesgo** ya que no se puede garantizar que el sistema esté disponible con solo tener la máquina encendida, por lo que no es dependiente de la arquitectura.
- Que el analista de fluidos o cualquier otro especialista del CEINPET tenga acceso solo a la aplicación que gestione información referente a su especialidad es simple. Esto se garantiza desde los módulos de administración y seguridad de la plataforma, propuestos en la arquitectura por lo que la seguridad se registra como un punto de sensibilidad, que afecta positivamente este atributo. Se identifica como un punto de **no riesgo**.
- Sustituir un sistema operativo por otro sería un cambio sencillo, ya que las herramientas propuestas para el desarrollo de la plataforma en su totalidad son multiplataforma. Tanto el intérprete de Python, como el gestor de base de datos o los frameworks utilizados son herramientas multiplataforma bajo licencias libres. La dependencia del sistema operativo se registra como un punto de sensibilidad, que afecta positivamente la modificabilidad y portabilidad. Se identifica como un punto de **no riesgo**.
- Sustituir el gestor de bases de datos es simple ya que la utilización del ORM SQLAlchemy genera una capa de abstracción de la base de datos, el uso de una base de datos u otra es transparente. El cambio del gestor de bases de datos se registra como un punto de sensibilidad, que afecta positivamente la modificabilidad. Se identifica como un punto de **no riesgo**.

- Trasladar la plataforma a otra estación de trabajo es simple ya que la plataforma está diseñada para eso, para ser instalada en diferentes estaciones de trabajo dentro del CEINPET, de ahí que se considere de gran portabilidad y se registre como un punto de sensibilidad, que afecta positivamente este atributo. Se identifica como un punto de **no riesgo**.

Con el estudio realizado a partir de los atributos y escenarios escogidos se obtuvieron siete puntos de sensibilidad, cinco puntos de no riesgo y dos puntos de riesgo no dependientes de la arquitectura.

3.7 Conclusiones Parciales

En este capítulo se describió la solución arquitectónica propuesta y fue evaluada mediante un método para este fin dejando al descubierto los riesgos surgidos de este análisis. Se espera esta solución sea clara para los entendidos en el tema y el equipo de desarrollo centre sus esfuerzos en desarrollar una plataforma de servicios a pozos siguiendo esta línea base. Si se apoyan en las vistas, diagramas y los requisitos presentados en este capítulo para resolver la problemática se espera desarrollar un software de calidad.

CONCLUSIONES GENERALES

La investigación realizada arrojó como resultado una propuesta de arquitectura de software que soporte el desarrollo de una plataforma de servicios de aplicaciones para el CEINPET. Cumpliéndose así el objetivo planteado al inicio de la investigación. Se seleccionaron además las herramientas y tecnologías más factibles a utilizar, y los estilos arquitectónicos y patrones de diseño que sustentarán el desarrollo de dicha plataforma. Se diseñaron las vistas de la arquitectura que guiarán el desarrollo de los componentes del sistema, así como los elementos necesarios para el despliegue de la plataforma de servicios a pozos. Por último, se evaluó esta propuesta de arquitectura mediante un método para este fin, identificando los riesgos y no riesgos de la misma.

Con la investigación realizada se espera haber dejado sentadas las bases para el desarrollo de la plataforma de servicios de aplicaciones para el CEINPET.

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible arribar a las siguientes recomendaciones:

- Poner en práctica el diseño arquitectónico propuesto.
- Refinar la arquitectura propuesta a partir de la puesta en práctica del diseño realizado.
- Garantizar que las salidas del sistema sean exactas para una óptima toma de decisiones por parte de los especialistas del CEINPET.
- Valorar por parte de la Universidad de las Ciencias Informáticas la reutilización del diseño arquitectónico propuesto para el desarrollo de nuevas aplicaciones informáticas con características similares.

BIBLIOGRAFÍA

1. Dewayne E. Perry y Alexander L.Wolf. Foundations for the study of software architecture. 1992.
2. Mary Shaw y Paul Clements. A field guide to Boxology: Preliminary classification of architectural styles for. 1996.
3. Robert Allen y David Garlan. The Wright Architectural Description Language. 1996.
4. Mark Klein y Rick Kazman. Attribute-based architectural styles. 1999.
5. Larman, Craig. UML y Patrones Introducción al análisis y diseño orientado a objetos . 2da Edición.
6. Grady Booch, James Rumbaugh e Ivar Jacobson. The Unified Modeling Language User Guide. Rational Software Corporation. Addison-Wesley, 1999.
7. Rumbaugh, Jacobson Booch. El proceso unificado de desarrollo de software. 2000.
8. Emanuel Blasi. Resumen de Patrones de Diseño.
9. Unidad Docente de Ingeniería del software. Patrones del Gang of Four.
10. Raúl González Duque. Python para todos. España
11. David Garlan y Mary Shaw. “An introduction to software architecture”. 1994.
12. Steve Burbeck. Application programming in Smalltalk-80: How to use Model-View Controller (MVC). University of Illinois in Urbana-Champaign.
13. Addison Wesley. El lenguaje de programación C++. España. 1998
14. Peter Müller. Introduction to Object-Oriented Programming Using C++. 1997.
15. Matthias Kalle Dalheimer. Qt vs. Java A Comparison of Qt and Java for Large-Scale, Industrial-Strength GUI Development.
16. Christian Pirchheim. Visual Programming of User Interfaces for Distributed Graphics Applications. 2006.
17. Roger S. Pressman. Ingeniería Del Software. Un Enfoque Práctico.
18. Emanuel Blasi. Resumen de Patrones de Diseño.
19. Tomás Javier Robles Prado. Introducción a Python v1.01.
20. Magnus Lie Hetland. Beginning Python. From Novice to Professional. 2005.
21. Thomas Lockhart. Tutorial de PostgreSQL. El equipo de desarrollo de PostgreSQL. 1996.
22. Mike Bayer. SQLAlchemy Documentation. 2009.
23. M.Barbacci, M. Klein,T. Longstaff. 2005. Quality Atributes. s.l: Carnegie Mellon University, 2005.

TRABAJOS CITADOS

1. Zayas, Carlos Álvarez de. Metodología de la Investigación Científica. Santiago de Cuba: s.n., 1995.
2. Reynoso, Carlos. Introducción a la arquitectura de software. Buenos Aires: s.n., 2004.
3. Shaw, David Garlan & Mary. An introduction to software architecture. Pitsburg : World Scientific, 1994.
4. IBM. Rational Unified Process. Ayuda RUP. 2006.
5. Almenares, Kiosmy. Diseño e implementación del proceso de inscripción del modulo mercantil. Caracas: s.n., 2007.
6. Plataforma de desarrollo. [Online] http://www.rational.com.ar/herramientas/softwaredevelopmentplatform.htm
7. Intel.com. [Online] Intel. http://www.intel.com/technology/index.htm?iid=gg_work+home_technology.
8. Giacomantonio, M. Dove vanno le piattaforme di e-Learning. 2004.
9. Benedetto, I. "Dalla valutazione dell'apprendimento alla valutazione dell'ambiente di apprendimento", en Form@re Erickson 2001.
10. Belyk, D. F., D. Software evaluation criteria and terminology, en The Center of Distance Education (Athabasca University)
11. Román Mendoza, E. C., P. and Pérez, P. Web-based Instructional Environments: Tools and Techniques for effective Second Language Acquisition. Servicio de Publicaciones de la Universidad de Murcia.
12. Roger S. Pressman. Ingeniería Del Software. Un Enfoque Práctico.
13. Clemens Alden Szyperski. Component software: Beyond Object-Oriented programming. Reading, Addison-Wesley, 2002
14. Shlumberguer. [Online] Shlumberguer. http://www.slb.com/
15. CMG. [Online] CMG. http://www.cmgl.ca/
16. Kappa. [Online] Kappa. http://www.kappaeng.com/
17. Leandris De la Cruz - Castillo Maikel Sánchez. Sistema de Gestión de Lechadas para la Cementación de Pozos Petroleros. La Habana: s.n., 2009.
18. Elizabeth Sánchez Suárez - Angel García. Sistema para la Gestión de Información de Pérdida de Circulación durante la Perforación de Pozos Petroleros. La Habana: s.n., 2009.

19. Alegna Martínez García - Johnny Cedeño. Sistema para la Evaluación y Control del trabajo con los Fluidos de Perforación. La Habana: s.n., 2009.
20. Ariagna Rodríguez Donatien - Carlos Enrique Ramírez. Sistema para la Identificación de Aguas en Pozos Petroleros. La Habana: s.n., 2009.
21. Carelys Suárez Arencibia - Anibal Santos. Sistema para la gestión de la información de muestras de núcleos. La Habana: s.n., 2009.
22. Belinda Revés Sotolongo. Análisis y diseño de un sistema para el diagnóstico de daños en pozos petroleros. La Habana: s.n., 2009.
23. Steve Burbeck. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). University of Illinois in Urbana-Champaign, Smalltalk Archive.
24. Craig Larman. UML y Patrones. México. 1999.
25. Patrones GRASP [Online] http://www.ldc.usb.ve/~teruel/ci3711/patron3a/index.html
26. Patrón alta cohesión [Online] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf
27. Patrón bajo acoplamiento [Online] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf
28. Rumbaugh, Jacobson Booch. El proceso unificado de desarrollo de software. 2000.
29. Grady Booch, James Rumbaugh e Ivar Jacobson. The Unified Modeling Language User Guide. Rational Software Corporation. Addison-Wesley, 1999.
30. Visual Paradigm [Online] http://www.visual-paradigm.com/product/vpuml/
31. Python [Online] http://www.slideshare.net/jpadillaa/python-parte1
32. Frameowrk [Online] http://jordisan.net/blog/2006/que-es-un-framework/
33. Sqlalchemy [Online] http://sqlalchemy.com
34. PostgreSQL [Online] http://postgresql.org/docs/
35. Subversion [Online] http://tigris.org/servlets/OpenCollabNet
36. Tortoisesvn [Online] http://tortoisesvn.tigris.org/
37. Rapidsvn [Online] http://rapidsvn.tigris.org/
38. Evaluación arquitectura [Online] http://www.cimat.mx/~trejov/Lemus2/SoftwareArchitectureAssesment.pdf
39. Kazman, Clements, Klein. 2001. Evaluating Software Architectures: Methods and Case Studies. Addison-Wesley, 2001.