Universidad de las Ciencias Informáticas

Facultad # 9

Título

Propuesta de Arquitectura de Software para la Empresa de Perforación y Extracción de Petróleo de Occidente

Trabajo de diploma para optar por el Título de

Ingeniero en Ciencias Informáticas

Autor: Yunier Alexander Pimienta Fernández

Tutor: Ing. Yadeny Aquino Jiménez

Co-tutor: Ing. Yulien Figueredo Guzmán

Esp. María Valdés Ramos

Lic. Jorge Emilio Escala Maceo

Asesor: Ing. Julio Alberto Leyva Duran

Dedicatoria

Dedico especialmente este trabajo a mi abuelo Ruben y a mi madre Bernardina por ser mi razón de ser.

Agradecimientos

Llegue este profundo agradecimiento a mis padres Bernardina y Antonio, a mis abuelos Esnerida y Rubén, mi hermana y al resto de la familia por haberme guiado por el buen camino y haber hecho posible que la escalada a esta cima no hubiese sido en vano.

A mi novia Lisandra por su apoyo incondicional, sus desvelos, ayuda y amor.

A mis más que amigos, a mis hermanos, Yulien Figueredo Guzmán y Jorge Emilio Escala Maceo por su inigualable amistad, su confianza y por estar ahí todos y cada uno de los días más importantes que he pasado en esta escuela.

A mis amigos Maggie, Mailin, Lisbet, Alejandro y Raudel por los buenos y malos momentos vividos a lo largo de estos 5 años.

A Magalis que ha sido como otra madre para mí.

Mi más profundo agradecimiento a la tutora que me apoyó y confió en mi en todo momento.

Mi más profunda gratitud a la cotutora María Valdés Ramos por su ayuda y confianza.

A todos los que han sido mis compañeros de estudio y de trabajo, por la ayuda ofrecida y por haber tenido la suerte de compartir con ellos.

Agradecer a todos los profesores que de una forma u otra aportaron su grano de arena a la preparación que he adquirido en los años de estudiante.

En fin, para no olvidar a nadie, a todos aquellos que me han apoyado durante este largo andar, a todos…,

Muchas gracias

Resumen

En los últimos años, las nuevas Tecnologías de la Información y las Comunicaciones (TIC) han provocado un fuerte movimiento en las comunidades científicas de todo el mundo. Cuba no está exenta a este auge surgido en la nueva “era de la información”, por lo que ha identificado la necesidad de informatizar la sociedad cubana.

En la Empresa de Perforación y Extracción de Petróleo de Occidente perteneciente a la Unión CUBAPETROLEO (CUPET), se manipula una gran cantidad de información, la cual necesita ser centralizada y que sea accesible uniformemente por los usuarios.

El presente trabajo de diploma contiene un estudio de las principales tendencias, tecnologías y metodologías actuales. Igualmente se definen los principales requerimientos del sistema y se obtiene como resultado el diseño de la arquitectura de software para un sistema, el cual permitirá optimizar el proceso de gestión de información. La solución propuesta delimita el estilo arquitectónico, patrones de arquitectura, lenguaje de programación y gestor de base de datos, que deben ser usados en la futura implementación del sistema.

El éxito de la aplicación depende en gran medida de las decisiones arquitectónicas tomadas durante su desarrollo, definiendo las características fundamentales de las tecnologías y aspectos esenciales de diseño. Todo esto proporciona a los miembros del equipo una idea clara y objetiva de que se está desarrollando.

Palabras Claves:

Gestión de información y Arquitectura de software.

Abstract

In the last years, the new Information and Communication Technologies (ICT) has encouraged a strong movement in the scientific communities in the whole world. Cuba is not exempt from this practice emerged in the new “information age”, so it has identified the need to automate the Cuban society.

In the Enterprises of Oil Extraction and Production of the West, belonging to the union CUBAPETROLEO (CUPET), it is managed a great amount of information, which needs to be centralized and equally accessible for the users.

The present work contains a study of the principal trends, technologies, and today’s methodologies. It is likely defined the main requirements of the system and it is obtained, as a result, a design of a software architecture for a system, that will allow optimizing the information management process. The solution proposed defines the architectonic style, patterns, programming language, and database management system that should be further used to implement the system.

The success of the application depends largely on the architectonic decisions made during its development, establishing the main characteristics of the technologies as well as essential aspects of the design. All these elements give the members of the team a clear and objective idea of what it’s being developed.

Key words:

Information Management and Software Architecture

Tabla de contenido

Introducción

Actualmente los procesos de gestión de la información y los conocimientos adquieren un papel protagónico cuando de lograr eficiencia y calidad se trata, ya sea en el campo investigativo, de formación o producción. Gracias al constante desarrollo de las Tecnologías de la Información y las Comunicaciones (TICs) estos procesos son automatizados, propiciando una superación y mejoras en todo el ámbito socioeconómico y cultural.

A nivel internacional muchas empresas y países desarrollados basan su crecimiento en la aplicación y la programación estratégica de las herramientas computacionales. Por lo que se han definido políticas que los inducirán a su permanencia en el dinamismo mundial de los próximos años. Lo que quiere decir, que los objetivos (o deseos) de avance y triunfo de estas empresas, giran alrededor del tratamiento automático de la información por medio de ordenadores. Con lo cual se logra proporcionar la información necesaria de una manera más rápida y segura a fin de tomar decisiones.

La informática cubana está enmarcada en la migración y desarrollo de aplicaciones de código abierto con vistas a propiciar un mayor desarrollo en el campo de la producción de software. El avance de las tecnologías de la información, cataliza las mejoras en metodologías y arquitecturas que se utilizan para automatizar procesos en las diferentes esferas y áreas de la vida. En el caso de empresas e instituciones, se han definido y personalizado gran cantidad de metodologías y herramientas que aportan ventajas de desarrollo e integración.

La Unión CUBAPETROLEO (CUPET) vinculada al Ministerio de la Industria Básica (MINBAS), es la encargada de satisfacer las necesidades del mercado nacional de hidrocarburos de forma competitiva, a partir del incremento de la producción y la optimización del uso de los combustibles nacionales, como contribución a la independencia económica del país. CUPET se basa fundamentalmente en la aplicación de tecnología de avanzada y un potencial humano altamente calificado.

CUBAPETROLEO cuenta con dos empresas encargadas de la Perforación y Extracción de Petróleo, cuya función es garantizar el incremento de la producción de petróleo y gas (Propia y de Compañías) a partir del empleo de mejores métodos de recuperación secundaria y terciaria, en la explotación de los

yacimientos petrolíferos, así como el control de la medición eficaz del 100 % del total de producción del fondo de pozos activos. Estas empresas se encargan además, de recolectar el total del fluido de producción del yacimiento, separar el agua libre y el gas acompañante entregando al proceso de tratamiento y comercialización la materia prima en valores de calidad, bajo el estricto cumplimiento de los indicadores técnicos, económicos y de calidad establecidos para el cumplimiento de las políticas de gestión ambiental.

La Empresa de Perforación y Extracción de Petróleo de Occidente (EPEPO) no cuenta con un sistema automatizado para manipular la gran cantidad de información referente a la gestión de indicadores de producción de dicha empresa. Se constató que todo el procesamiento y consolidación de los datos de estos indicadores se realiza de forma manual o semiautomática, empleando diferentes herramientas ofimáticas como son: Microsoft Excel, Microsoft Word, Microsoft Access, lo que trae como consecuencia:

- ✓ Demoras en la entrega de informaciones.
- ✓ Los reportes generados no reflejan la claridad requerida.
- ✓ Bases de datos simples, locales y de poco alcance.
- ✓ Carente informatización de la gestión de datos.
- ✓ Información inconsistente y descentralizada.

Luego del análisis de la situación actual, surge el siguiente **problema científico**: La inadecuada gestión del flujo de información dificulta el control diario de los datos que se generan a partir del proceso de producción en la EPEPO.

Según el problema identificado anteriormente se plantea como **objeto de estudio:** Procesos de gestión del flujo de información de la producción en empresas petroleras.

Para resolver el problema se propone el siguiente **objetivo general:** Diseñar e implementar una arquitectura para un sistema que permita el control de los procesos de extracción y producción en la EPEPO.

Para dar cumplimiento al objetivo general, se definieron los siguientes **objetivos específicos:**

- ✓ Documentar el flujo de información del proceso de gestión de la producción en empresas petroleras cubanas.
- ✓ Definir la Línea Base de la Arquitectura.

El objetivo trazado delimita el siguiente **campo de acción:** El proceso de gestión de información de producción en la EPEPO.

Para cumplir con los objetivos de esta investigación y resolver la situación problémica planteada, se proponen las siguientes **tareas:**

1. Realizar un estudio de las principales características del proceso de producción en empresas petroleras cubanas.
2. Investigar el estado del arte de sistemas que controlen el flujo de información de producción de empresas petroleras.
3. Determinar las cualidades y propiedad que debe tener el sistema.
4. Definir la metodología de software a utilizar.
5. Definir las herramientas a utilizar para el desarrollo de la aplicación.
6. Definir el estilo arquitectónico a utilizar en el desarrollo del software.
7. Definir patrones a utilizar, así como su aplicación.
8. Definir la Línea Base de la Arquitectura.

Todo lo antes expuesto conduce a plantear la siguiente **Hipótesis:** Si se define la arquitectura de software para un sistema que controle el flujo informativo generado a partir del proceso de producción de la Empresa de Perforación y Extracción de Petróleo de Occidente, se logrará una mayor eficiencia en los procesos de gestión de información de la producción.

Se espera como **posible resultado** lograr una arquitectura completa y flexible para un sistema que controle el flujo de información que se genera a partir del proceso de producción en la Empresa de Perforación y Extracción de Petróleo de Occidente.

Para la realización de estas tareas se utilizaron los siguientes métodos de investigación:

- ✓ Métodos Teóricos.
  - Análisis y síntesis: Para conocer, reflexionar y aumentar los conocimientos acerca de la gestión de información identificada en los procesos de extracción y perforación en la EPEPO a partir de la consulta de la literatura científica correspondiente y la experiencia personal de la visita a dicha empresa, y luego, para confeccionar un cuerpo coherente en el cual se resuman los resultados obtenidos del análisis.
  - Modelación: Con el objetivo de modelar el diseño de la arquitectura propuesta.
- ✓ Métodos Empíricos
  - Observación: Se utilizó para comprobar el estado real del personal de la EPEPO en consecuencia con las necesidades de obtención de información precisa.
  - Entrevistas: Se entrevistaron técnicos y especialistas que laboran en la EPEPO para obtener informaciones precisas y confiables sobre aspectos específicos inherentes a la solución del problema de investigación.

Capítulo 1: Fundamentación Teórica

1.1 Introducción

Existen conceptos asociados a los procesos de gestión de la información petrolera que pueden resultar desconocidos y son de vital importancia para la comprensión de la investigación. Ayuda en este tema, el conocimiento del estado actual de estos, el análisis y comparación de tendencias, metodologías, tecnologías y herramientas de actualidad que fundamentan la selección que se presenta en capítulos posteriores.

1.2 Conceptos asociados

El *proceso de extracción*, *recolección*, *y separación de petróleo y gas* tiene como misión, garantizar el incremento de la producción de petróleo y gas (Propia y de Compañías), a partir del empleo de mejores métodos de recuperación secundaria y terciaria en la explotación de los yacimientos petrolíferos, así como el control de la medición eficaz del 100 % del total de producción del fondo de Pozos Activos.

Además de recolectar el total del fluido de producción del yacimiento, separar el agua libre y el gas acompañante entregando al proceso de tratamiento y comercialización la materia prima en valores de calidad de emulsión petróleo – agua (BSW) máximo del 15%. Todo esto se debe hacer bajo el estricto cumplimiento de los indicadores técnicos, económicos y de calidad establecidos para el cumplimiento de las políticas de gestión ambiental y de calidad.

La *información* se define como un conjunto de datos organizados, significativos y pertinentes que describen sucesos o entidades. En cuanto al universo de la computadora, la información es un factor fundamental que se representa a través de símbolos, específicamente en forma de datos binarios[1].

La *gestión de información* es el proceso que se encarga de suministrar los recursos necesarios para la toma de decisiones, así como para mejorar los procesos, productos y servicios de las organizaciones. Es el proceso mediante el cual se obtiene, despliega o utiliza una variedad de recursos básicos para apoyar los objetivos de la organización. Su utilidad está dada en función de su aporte a los procesos de

toma de decisiones, creación de productos y solución de problemas, entre otros aspectos [2]. Es todo lo relacionado con la obtención de la información adecuada, en la forma correcta, para la persona indicada, al costo adecuado, en el tiempo oportuno, en el lugar apropiado y para tomar la acción correcta [3].

Un *sistema* es un conjunto de funciones que operan en armonía o con un mismo propósito, y que puede ser ideal o real. Por su propia naturaleza, un sistema posee reglas o normas que regulan su funcionamiento, por lo que puede ser entendido, aprendido y enseñado [4].

Cualquier sistema es más o menos complejo pero debe poseer una coherencia discreta acerca de sus propiedades y operación. En general, los elementos o módulos de un sistema interactúan y se interrelacionan entre sí.

El desarrollo de software incluye un gran estudio y acumulación de una idea en la realidad, también supone la aplicación de la inventiva y del ingenio para desarrollar una cierta actividad. Esto, por supuesto, no implica que no se utilice el método científico para llevar a cabo los objetivos. Es decir, a través de técnicas, diseños, modelos y con el conocimiento proveniente de las ciencias, se puede resolver problemas y satisfacer necesidades de los clientes.

El término *Ingeniería* se define como el conjunto de conocimientos y técnicas que permiten aplicar el saber científico a la utilización de la materia y de las fuentes de energía. Además como conjunto de conocimientos y técnicas cuya aplicación permite la utilización racional de los materiales y de los recursos naturales, mediante invenciones, construcciones u otras realizaciones provechosas para el hombre [5].

Hoy día es cada vez más frecuente la consideración de la *Ingeniería del Software (ISW)* como una nueva área de la ingeniería. Esta es una disciplina formada por un conjunto de métodos, herramientas y técnicas que se utilizan en el desarrollo de los programas informáticos (software). Incluye el análisis previo de la situación, el diseño del proyecto, el desarrollo del software, las pruebas necesarias para confirmar su correcto funcionamiento y la implementación del sistema [6].

La Ingeniería del Software es la aplicación práctica del conocimiento científico en el diseño y construcción de programas de computadora. Es al mismo tiempo la documentación asociada requerida para desarrollar, operar (funcionar) y mantenerlos [5].

Hay muchas formas de enfocar un problema utilizando una solución basada en el software. Un enfoque muy utilizado es la visión orientada a objetos. El dominio del problema se caracteriza mediante un conjunto de objetos con atributos y comportamientos específicos. Los objetos son manipulados mediante una colección de funciones (llamadas métodos, operaciones o servicios), se comunican entre ellos mediante un protocolo de mensajes y se clasifican mediante clases y subclases.

Dentro de las principales características de un objeto está encapsular tanto datos, como procesos que se aplican a esos datos, permitiendo construir clases de objetos e inherentemente construir bibliotecas de objetos y clases reutilizables.

La *Ingeniería del Software Orientado a Objetos (ISOO)* sigue los mismos pasos que el enfoque convencional. El análisis identifica las clases y objetos relevantes en el dominio del problema. El diseño proporciona detalles sobre la arquitectura, las interfaces y los componentes. La implementación (utilizando un lenguaje orientado a objetos) transforma el diseño en código, y las pruebas chequean tanto la arquitectura como las interfaces y los componentes. Todo lo anteriormente planteado da como solución un conjunto de modelos orientados a objetos los cuales describen los requisitos, el diseño, el código y los procesos de pruebas para un sistema o producto.

Las tecnologías de objetos llevan a reutilizar, y la reutilización (de componente de software) lleva a un desarrollo de software más rápido y a programas de mejor calidad. El software orientado a objetos es más fácil de mantener debido a que su estructura es inherentemente poco acoplada. Esto lleva a menores efectos colaterales cuando se deben hacer cambios, y provoca menos frustración en el ingeniero del software y en el cliente. Además, los sistemas orientados a objetos son más cómodos de adaptar y poseen mayor escalabilidad [5].

En la actualidad, los sistemas complejos y de alta calidad basados en computadora se deben construir en períodos de tiempo muy cortos. Esto se logra con un enfoque de reutilización más organizado.

La *Ingeniería del Software Basada en Componentes (ISBC)* es un proceso que se centra en el diseño y construcción basado en computadora que utilizan componentes de software reutilizables. Proporciona beneficios inherentes en lo respecto a calidad del software, productividad del desarrollador y coste general del sistema.

El proceso ISBC acompaña a dos subprocesos concurrentes: ingeniería del dominio y desarrollo basado en componentes. El objetivo de la ingeniería del dominio es identificar, construir, catalogar y diseminar un conjunto de componentes de software en un determinado dominio de aplicación. A continuación, el desarrollo basado en componentes cualifica, adapta e integra estos componentes para su reutilización en un sistema nuevo. Conjuntamente, el desarrollo basado en componentes diseña componentes nuevos que se basan en los requisitos personalizados de un sistema nuevo.

La ingeniería del software basada en componentes hace uso de un modelo de intercambio de datos, herramientas, almacenamiento estructurado y un modelo objeto subyacente para construir aplicaciones. El modelo de objetos suele ajustarse a uno o más estándares de componentes (por ejemplo, OMG/CORBA) que definen la forma en que una aplicación puede acceder a los objetos reutilizables.

A medida que crece la complejidad de las aplicaciones y que se extiende el uso de sistemas distribuidos y sistemas basados en componentes, los aspectos arquitectónicos del desarrollo de software están recibiendo un interés cada vez mayor, tanto desde la comunidad científica como desde la propia industria del software.

Las innumerables definiciones de la *Arquitectura de Software* (AS) se agrupan en cuatro grandes grupos: modernas, clásicas, bibliográficas y las que brindan personas dedicadas al tema, agrupadas en la comunidad del Instituto de Ingeniería de Software “Carnegie Mellon”. La mayoría poseen puntos de coincidencia que giran alrededor de: elementos de software, componentes, propiedades y relaciones. A continuación se enumeran algunas de ellas:

- ✓ La AS de un programa o sistema computacional es la estructura del sistema, la cual comprende: elementos de software, propiedades externamente visibles de esos elementos y relaciones entre ellos [7].

- ✓ La AS es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos, el ambiente y los principios que orientan su diseño y evolución [7].

Se puede concluir que la AS aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos para su desarrollo posterior en el diseño. Su objetivo principal es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permita la comunicación entre los equipos que participen en un proyecto. Esta no solo está relacionada con la estructura y el comportamiento del sistema, sino que está también dictada por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones, los compromisos económicos y tecnológicos, además de la estética.

La relación siguiente hace una recopilación de los objetivos más importantes que debe cubrir un diseño de arquitectura:

- ✓ Dar cobertura a la funcionalidad: El diseño del software debe, en primer lugar, centrarse en dar cabida a todas las funcionalidades que se especifican, o sea, debe incluir dentro de su diseño la solución a los requisitos que le dan funcionalidad al sistema.
- ✓ Facilitar el desarrollo del proyecto de software: Un software puede ser un proyecto complejo donde intervengan docenas de personas y se extienda durante años. Es un objetivo crucial que el reparto de funciones entre los distintos módulos permita un desarrollo independiente, pero coordinado de los mismos. La programación orientada a objetos y a componentes permite por su filosofía de origen, una programación modular y a la vez independiente pero coordinada.
- ✓ Optimizar el uso de los recursos del hardware: Los recursos se refieren a la capacidad de cálculo de los procesadores, al uso de la memoria, al almacenamiento y acceso a los discos, a la capacidad de generación gráfica y a la capacidad de comunicaciones.
- ✓ Permitir el intercambio entre módulos: Para mejorar y ampliar las capacidades del software en el futuro es preciso que la arquitectura sea lo suficientemente abierta para poder sustituir un módulo sin que se vean afectados demasiados los demás.

En la ingeniería de software al igual que en la construcción; los estilos arquitectónicos describen categorías, conjunto de componentes, cooperación entre ellos, conectores y modelos, por lo que se

pudiera delimitar los *estilos arquitectónicos* como conceptos descriptivos que definen una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales [8].

Además, los estilos arquitectónicos se pudieran describir como un conjunto de reglas de diseño que identifican las clases de componentes y conectores que se pueden utilizar para formar un sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo [9].

Desde la etapa más temprana de concepción del software se encuentran patrones. *Los patrones* se refieren a prácticas de re-utilización y los estilos conciernen a teorías sobre las estructuras de los sistemas a veces más formales que concretas [8].

Los *patrones* pueden dividirse o clasificarse en (ver anexo 1):

Patrones de arquitectura que son los relacionados con la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, rendimiento, modularidad y acoplamiento. Expresan un paradigma fundamental para estructurar u organizar un sistema software.

Los *patrones de diseño,* los cuales proporciona un esquema para refinar los subsistemas o componentes de un sistema software y las relaciones entre ellos. Describe estructuras recurrentes para comunicar componentes que resuelven un problema de diseño en un contexto particular. Son patrones de un nivel de abstracción menor que los patrones de arquitectura. Están por lo tanto más próximos a lo que sería el código fuente final. Su uso no se refleja en la estructura global del sistema.

Además los *patrones de análisis* describen un conjunto de prácticas destinadas a elaborar modelos de los conceptos principales de la aplicación que se va a construir.

Los *patrones de proceso u organización* tratan todo lo concerniente con el desarrollo o procesos de administración de proyectos, técnicas y estructuras de organización.

Los *patrones de idioma* reglan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas. Son específicos a un lenguaje de programación determinado. Describe como implementar aspectos particulares de los componentes de un patrón de diseño usando las características y potencialidades de un lenguaje de programación concreto.

Se suele argumentar que los patrones ofrecen tres ventajas fundamentales:

- ✓ Están ya probados: Son soluciones que han sido utilizadas con anterioridad de manera repetida y se ha comprobado que funcionan.
- ✓ Son reutilizables: Corresponden con problemas que no son específicos de un caso concreto, sino que se presentan una y otra vez en distintas aplicaciones.
- ✓ Son expresivos: Cuando un equipo de desarrolladores tiene un vocabulario común de patrones, se puede comunicar de manera fluida y precisa las ideas fundamentales sobre el diseño de una aplicación.

Según señala Mary Shaw y Neno Medvidovic en sus revisiones de los lenguajes de descripción arquitectónica, el uso que se da en la práctica y en el discurso a conceptos tales como arquitectura de software o estilos suele ser informal y ad hoc. En la práctica industrial, las configuraciones arquitectónicas se suelen describir por medio de diagramas de cajas y líneas, con algunos añadidos diacríticos. Los entornos para esos diagramas suelen ser de espléndida calidad gráfica, comunican con relativa efectividad la estructura del sistema y siempre brindan algún control de consistencia respecto de qué clase de elemento se puede conectar con cuál otra; pero a la larga, proporcionan escasa información sobre la computación efectiva representada por las cajas, las interfaces expuestas por los componentes o la naturaleza de sus computaciones [10].

En la década de 1990 y en lo que va del siglo XXI, se han materializado diversas propuestas para describir y razonar en términos de arquitectura de software. Muchas de ellas han asumido la forma de *lenguajes de descripción arquitectónica (ADLs)*. Estos ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento [8].

Conjuntamente suministran construcciones para especificar abstracciones arquitectónicas y mecanismos para descomponer un sistema en componentes y conectores, especificando de qué manera estos elementos se combinan para formar configuraciones y definiendo familias de arquitectura o estilos. Contando con un ADL, un arquitecto puede razonar sobre las propiedades del sistema con precisión, pero a un nivel de abstracción convenientemente genérico. Algunas de esas propiedades podrían ser protocolos de interacción, anchos de banda y latencia, localización del almacenamiento, conformidad con estándares arquitectónicos y previsiones de evolución después del sistema [10].

El modelo omite detalles que no resultan esenciales para la comprensión del original y por lo tanto facilita dicha comprensión. El modelado no es más que la construcción de un modelo a partir de una especificación.

En la actualidad, la mayoría de las herramientas que se utilizan para el modelado, tanto en los medios empresariales como académicos, tienen como base o lenguaje de modelado, al *Lenguaje Unificado de Modelado (UML - Unified Modeling Language).* Este lenguaje permite que se represente de manera semiformal la estructura general del sistema, con la ventaja de que este mismo pueda ser usado en todas las etapas de desarrollo del sistema y su representación gráfica puede ser usada para comunicarse con los usuarios.

El lenguaje UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, etc., hasta la implementación y configuración con los diagramas de despliegue.

UML es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema de software orientado a objetos. Ofrece un estándar para describir un "plano" del sistema, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

En la informática, las aplicaciones son los programas con los cuales el usuario final interactúa, es decir, son aquellos programas que permiten la interacción entre el usuario y la computadora. Dentro de estos

programas se ubican dos grandes grupos: *aplicaciones desktop o de escritorio* y las *aplicaciones web*. Las primeras son aplicaciones que se encuentran instaladas en la computadora donde se está trabajando, o sea, esta estará ocupando parte del espacio disponible en la computadora el cual pudiera utilizarse para otros recursos o servicios. A la vez son más seguras que las aplicaciones web, por sus facilidades de uso, manejo y son más rápidas en cuanto al funcionamiento y cumplimiento de las peticiones. Las últimas son páginas web especializadas, que permiten mostrar o captar información desde la interfaz de usuario la cual puede ser almacenada en bases de datos; también son populares debido a lo práctico del navegador web como cliente ligero.

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. Presenta una interfaz, mediante la cual el usuario puede comunicarse con el sistema físico y realizar operaciones como almacenar o recuperar datos de ella. Entre las principales funciones que debe cumplir un SGBD están la creación y mantenimiento de la base de datos, el control de accesos, la manipulación de datos de acuerdo con las necesidades del usuario, el cumplimiento de las normas de tratamiento de datos y evitar redundancias e inconsistencias [11].

En la actualidad las grandes empresas de desarrollo de software con el fin de obtener un proceso de desarrollo más predecible, eficiente y mucho más disciplinado, utilizan las *metodologías*. Esto se logra desarrollando un proceso detallado con un fuerte énfasis en la planificación e inspirado por otras disciplinas de la ingeniería. Las *metodologías de desarrollo de software* son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.

Toda metodología debe ser adaptada al contexto del proyecto (recursos técnicos y humanos, tiempo de desarrollo, tipo de sistema, etc.). Históricamente, las metodologías tradicionales han intentado abordar la mayor cantidad de situaciones de contexto del proyecto, exigiendo un esfuerzo considerable para ser adaptadas, sobre todo en proyectos pequeños y con requisitos muy cambiantes. Las metodologías ágiles por su parte, ofrecen una solución casi a medida para una gran cantidad de proyectos que tienen estas características.

El uso de las herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) es visto, por una parte, como una solución para los mayores problemas relacionados con las formas tradicionales de desarrollo de sistemas de software: demoras, errores, inconsistencias, dificultad para las pruebas e inadecuada documentación [12] y por otra parte, como un soporte para el desarrollo de sistemas de calidad a través de la automatización del ciclo de vida de desarrollo. Al adoptar una herramienta CASE, las organizaciones tienen la gran expectativa de que ello implica grandes mejoras en la calidad y velocidad en el proceso de desarrollo de los sistemas [13].

Cuando de simplificar el desarrollo de aplicaciones se trata se deben tener en cuenta los framework, los cuales suelen ser implementaciones de patrones de diseño conocidos, aderezados con funciones que asisten al desarrollador [14].

En el contexto de la programación un framework es un set de funciones o código genérico que realiza tareas comunes y frecuentes en todo tipo de aplicaciones (creación de objetos, conexión a base de datos, limpieza de strings, etc.). Esto brinda una base sólida sobre la cual desarrollar aplicaciones concretas y permite obviar los componentes más triviales y genéricos del desarrollo.

Un *framework* en el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Conocer un vocabulario y una gramática no equivale a saber un idioma. Conocer un idioma implica además el hábito de combinar sus elementos de forma semiautomática para producir frases que se quieran decir. Conocer las palabras, las sentencias de un lenguaje no equivale a saber programar, pero son condiciones necesarias para estar listo y empezar a hacerlo, o entender cómo funcionan programas ya hechos. Un *lenguaje de programación* es la notación para la descripción precisa de algoritmos o programas informáticos. Es el conjunto de instrucciones que permiten al programador pensar claramente sobre la complejidad del problema a resolver, de manera que pueda ordenarlas

convenientemente para la creación de un programa ejecutable por la computadora. Se dividen en lenguajes de alto y bajo nivel según se acerquen más o menos a las formas de comunicación humana, respectivamente [15].

1.3 Estado actual de los sistemas de información petrolera.

En la actualidad las entidades encargadas de la producción de petróleo, tanto a nivel nacional como internacional, son más exigentes a la hora de mejorar y buscar software destinados para controlar la producción de petróleo. En general, todas las empresas en su afán de obtener una mayor calidad y rapidez en su funcionamiento, aspiran a los mejores sistemas informáticos y, simultáneamente las empresas desarrolladoras de software realizan productos que satisfacen en gran medida las expectativas de sus clientes.

De los sistemas de información petrolera, existen disímiles productos y versiones pertenecientes a varias empresas de desarrollo de software. En Cuba, se han hecho intentos, aunque no existe la suficiente difusión e intercambio de experiencias como para realizar y estandarizar un producto que cubra las expectativas de CUPET. Ejemplos de estos intentos es el Sistema de Producción (SisProd) implantado y funcionando en la EPEP-Centro, ubicada en el yacimiento de Varadero. Este sistema a pesar de abarcar todas las funcionalidades para el proceso de extracción y recolección del petróleo en este lugar, no puede ser utilizado en las demás empresas de este tipo en el país debido a que fue diseñado específicamente para esta empresa por las características que presenta.

A nivel mundial figuran una serie de compañías que producen software destinados al control de la producción de petróleo, las cuales ya tienen un alto grado de experiencia en dicha materia y cuentan además con un grupo de especialistas altamente capacitados, entre ellas están:

Infoil International empresa argentina, cuyo objetivo se centra en construir software para la producción de petróleo y gas desde el 1992. Aplican pleno compromiso con los costos efectivos de soluciones para la gestión de las operaciones de petróleo, con base en la experiencia real y excelencia en la técnica. Una de las aplicaciones que está dedicada a la producción de datos de administración en la industria de petróleo y gas es **InfoProd.** Este sistema permite acceso simultáneo a toda la información, ofrece una

amplia recolección de datos, la integración y el análisis de la producción, ubicación de yacimientos y la formación o el campo. Es una excelente plataforma tecnológica con funcionalidades específicas basadas en las necesidades de cada cliente.

Field development (FDC) es una compañía consultora en servicios integrados para la industria del gas y el petróleo. Está compuesta por un equipo de profesionales de las ramas de Ingeniería en Petróleo, Geología, Sistemas y Técnicos Laboratoristas especializados en las disciplinas de ingeniería de producción, reservorios, modelado geológico, desarrollo de sistemas y en la ejecución de experiencias de laboratorio. Uno de los productos desarrollados por esta compañía es **BACO 6.0** que constituye un modelo de balance composicional de materiales específicamente orientado a yacimientos de gas y condensado. Con él es posible programar en el tiempo la entrada en producción de nuevos pozos, así como también simular procesos de reciclado de gas en forma total o parcial. Otra de sus aplicaciones es **PetroPlan** que es un conjunto de más de 80 funciones de uso frecuente en la industria del petróleo y el gas. Estas funciones constituyen la base de cálculos de Ingeniería de Reservorios, Ingeniería de Producción, diseño de instalaciones de superficie y sistemas de transporte de fluidos.

Wonderware es una empresa que ofrece una plataforma de software SCADA para empresas petrolíferas y de gas, dotada de una impresionante flexibilidad, de fácil mantenimiento y de gran fiabilidad. Las soluciones Wonderware están construidas e integradas con una arquitectura de software sencilla, abierta y escalable que permite la conexión con prácticamente todos los sistemas de automatización, unidades terminales remotas (RTU), servidores de medición de campo electrónico (EFM), bases de datos, sistemas históricos o empresariales que se utilizan hoy día. Esto permite a los usuarios expandir sus sistemas SCADA de Petróleo y Gas sin necesidad de comprar nuevos hardware o sistemas de control.

Varec FuelsManager ha sido un líder innovador en el petróleo y sector químico durante ocho décadas que trabajan con las industrias líderes en la instrumentación de medición de tanques combustible y los sistemas de gestión de combustible, en el establecimiento de normas y la introducción de nuevas tecnologías. **FuelsManager ® Oil & Gas** es la solución industrial para gestión de inventarios, tanques de medición, SCADA y el seguimiento del movimiento de los productos para la mayoría de los grandes las compañías petroleras. FuelsManager Oil & Gas está disponible en cuatro ediciones: Standard,

Professional, Terminal Automation y Enterprise. Esto permite elegir una edición que se adapta al nivel de funcionalidad que se requiere.

Estos sistemas, en su mayoría diseñados e implementados usando el estilo arquitectónico cliente/servidor en tres capas, han sido desarrollados utilizando software privativo, lo que eleva en gran medida los costos de producción e implantación. Además de imposibilitar la reutilización de estas experiencias en nuevas versiones, atentando entre otras cosas contra la estandarización. La gran mayoría de estas soluciones están implementadas en aplicaciones de escritorio brindando compatibilidad con el sistema operativo Windows solamente.

A pesar de los diversos servicios que brindan estas empresas y la eficacia de sus aplicaciones se puede afirmar que ninguno cumple con las características específicas que desea CUPET para el control de la producción de petróleo en Cuba. Además la obtención de alguno de ellos significaría costos adicionales en licencias de sistemas operativos, software, etc.

1.4 Tendencias, Herramientas y Tecnologías

1.4.1 Metodologías de Desarrollo

Microsoft Solutions Framework (MSF)

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

El modelo de proceso MSF combina los mejores principios del modelo en cascada y del modelo en espiral. Combina la claridad que planea el modelo en cascada y las ventajas de los puntos de transición del modelo en espiral.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación [16].

Rational Unified Process (RUP)

El proceso unificado de desarrollo (RUP) es una metodología para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software [17].

RUP es un proceso en el que se define claramente quien, cómo, cuándo y qué debe hacerse en el proyecto. Como tres características esenciales está dirigido por los Casos de Uso: que orientan el proyecto a la importancia para el usuario y lo que este quiere, está centrado en la arquitectura: que relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y es iterativo e incremental: donde divide el proyecto en mini-proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada [18].

Como filosofía RUP maneja seis principios clave:

- ✓ Adaptación del proceso: El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.
- ✓ Balancear prioridades: Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.
- ✓ Colaboración entre equipos: El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc.
- ✓ Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.
- ✓ Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o esquemas (framework) por nombrar algunos. Estos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

- ✓ Enfocarse en la calidad: El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

Programación Extrema (Extreme Programming, XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, utilizadas para proyectos de corto plazo. Es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en el desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. Su particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto [19]. Dentro de sus principales características se destacan:

- ✓ Pruebas Unitarias: Se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándose en algo hacia el futuro, puedan hacerse pruebas de las fallas que pudieran ocurrir.
- ✓ Re fabricación: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- ✓ Programación en pares: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento.

Esta metodología propone además comenzar el desarrollo con algo pequeño y añadir funcionalidad con retroalimentación continua. El manejo de cambios se convierte en una parte sustantiva del proceso y su costo no depende de la etapa o fase en la que se encuentre. Las funcionalidades son solamente introducidas cuando son necesarias y el usuario o cliente se convierte en miembro del equipo.

1.4.2 Estilos arquitectónicos

La relación de estilos arquitectónicos que a continuación se muestra, no incluye a todos los que existen. Se mencionan aquellos que son más utilizados en la actualidad, los cuales se agrupan en clases, las que engloban una serie de estilos arquitectónicos que comparten características.

Estilos de flujo de datos

Filtros y Tuberías (Pipes & Filters): El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. Cada componente tiene un conjunto de entradas y un conjunto de salidas. Un componente lee entradas y las transforma en salida. Los componentes son programas independientes; lo que sucede es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente [20].

Este estilo soporta una buena manera de reutilización, facilita el mantenimiento y mejora de sus componentes así como, soporta la ejecución concurrente. Sin embargo no es aconsejable para cuando se necesita una gran interactividad con los datos puesto que presenta problemas de rendimiento debido a que los datos se transmiten de forma completa entre filtros.

Estilos centrados en datos

Arquitecturas de Pizarra o Repositorio: La estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él son los dos componentes principales en esta arquitectura. Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados [20].

Estilos llamada y retorno

Model-View-Controller (MVC): El patrón 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. El modelo que se encarga de administrar el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado y a instrucciones de cambiar el estado (habitualmente desde el controlador); la vista que maneja la visualización de la información y el controlador que interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado [20].

Debido a la separación del modelo de la vista y la inexistencia de dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos

simultáneamente. Asimismo esta interfaz puede cambiar y adecuarse a interfaces para nuevos dispositivos como teléfonos celulares o PDAs, estos nuevos requerimientos de la interfaz no afectaría al modelo en casi nada. Sin embargo, es necesario tener en cuenta que si es el modelo el que cambia constantemente, podría desbordar a la vista con requerimientos de actualización haciendo colapsar el sistema.

Arquitecturas en capas: Es una arquitectura con 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. Los componentes son cada una de las capas y los conectores son los protocolos de interacción entre las capas [21].

Las interacciones entre las capas ocurren generalmente por invocación de métodos, por definición los niveles más bajos no deben poder utilizar funcionalidad ofrecida por las capas superiores. Este estilo facilita la modularidad del sistema, la localización de errores y mejora considerablemente el soporte del sistema. Pero en ocasiones resulta complicado encontrar la separación de las capas adecuadas.

Arquitecturas orientadas a objetos: Los componentes del estilo se basan en principios OO o sea el encapsulamiento, la herencia y el polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación; las interfaces están separadas de las implementaciones; en cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases.

Entre las cualidades de este estilo, la más básica concierne a que se puede modificar la implementación de un objeto sin afectar a sus clientes. Asimismo es posible descomponer problemas en colecciones de agentes en interacción. Además, por supuesto, un objeto es ante todo una entidad reutilizable en el entorno de desarrollo [20].

Arquitecturas basadas en componentes: Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica. Los componentes son las unidades de modelado, diseño e implementación, las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico. Los

componentes soportan algún régimen de introspección, de modo que su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución [20].

El marco arquitectónico estándar para la tecnología de componentes está constituido por los cinco puntos de vista de Reference Model for Open Distributed Procesing RM-ODP (Empresa, Información, Computación, Ingeniería y Tecnología).

Estilos de Código Móvil

Arquitectura de máquinas virtuales: Se ha llamado también intérpretes basados en tablas o sistemas basados en reglas, este estilo incluye un seudo-programa a interpretar y una máquina de interpretación. Ambas variedades abarcan, sin duda, un extenso espectro que va desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación [9].

Formado por un motor de simulación o interpretación, una memoria que contiene el código a interpretar, una representación del estado de la interpretación y otra del estado del programa que se está simulando. Esta arquitectura siempre está reducida al lenguaje de programación y no siempre es aplicable.

Estilos heterogéneos

Sistemas de control de procesos: Desde el punto de vista arquitectónico, mientras casi todos los demás estilos se pueden definir en función de componentes y conectores, los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos rangos especificados, llamados puntos fijos o valores de calibración [20].

Estilos Peer-to-Peer

Arquitecturas basadas en eventos: Se han llamado también de invocación implícita. Se vinculan históricamente con sistemas basados publicación suscripción. Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos.

Arquitecturas orientadas a servicios (SOA): Es lo suficientemente flexible, elegante y ágil garantizando las soluciones que las empresas han anhelado siempre. Es una arquitectura de software que construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces; es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.

1.4.3 Herramientas CASE

Visual Paradigm

Esta herramienta está desarrollada por Visual Paradigm Internacional una de las principales compañías de herramientas CASE. Su mayor éxito consiste en la capacidad de ejecutarse sobre diferentes sistemas operativos lo que le confiere la característica de ser multiplataforma. Visual Paradigm utiliza UML como lenguaje de modelado ofreciendo soluciones de software que permiten a las organizaciones desarrollar las aplicaciones de calidad más rápido, bien y más barato. Es muy fácil de usar y presenta un ambiente gráfico agradable para el usuario.

Su notación es muy parecida a la estándar, permite configurar las líneas de redacción, el modelado de base de datos, el modelado de requerimientos, el modelado del proceso de negocio, la interoperabilidad, la generación de documentación y la generación de código base para diferentes lenguajes de programación como Java, C# y PHP. Igualmente permite la integración con herramientas de desarrollo (IDE´s) [22].

Rational Rose

Rational Rose es una de las más poderosas herramientas de modelado visual para el análisis y diseño de sistemas basados en objetos. Se utiliza para modelar un sistema antes de proceder a construirlo. Además Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML 1.1. Rational propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática, otra dinámica de los modelos del sistema, una lógica y otra física; que permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software. Rose es una plataforma

independiente que ayuda a la comunicación entre los miembros de equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas.

1.4.4 Sistemas Gestores de Base Datos (SGBD ó DBMS)

Los Sistemas Gestores de Base de Datos, como se explico anteriormente, son un tipo de software muy específico, dedicado a servir de interfaz entre la Base de Datos, el usuario y las aplicaciones que la utilizan. Se componen de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. A continuación se mencionan algunos de ellos y sus características.

SQLServer

Microsoft SQLServer es uno de los mejores SGDB para Windows, es el ampliamente elegido por una amplia gama de clientes corporativos y proveedores independientes de software que construyen aplicaciones de negocios. Las necesidades y requerimientos de los clientes han llevado a la creación de innovaciones de producto significativas para facilitar la utilización, escalabilidad, confiabilidad y almacenamiento de datos. Está basado en el lenguaje Transact-SQL, y específicamente en Sybase IQ, capaz de poner a disposición de muchos usuarios grandes cantidades de datos de manera simultánea.

Oracle

Oracle es una herramienta cliente/servidor para la gestión de Base de Datos relacional. Esta herramienta hace uso de los recursos del sistema informático en todas las arquitecturas de hardware para garantizar su aprovechamiento al máximo en ambientes cargados de información. Con nuevas funciones de autogestión, Oracle elimina las tareas administrativas lentas y propensas a errores, lo que permite a los administradores de las BD concentrarse en los objetivos estratégicos de la empresa en lugar de prevenir problemas de rendimiento y disponibilidad [23].

PostgreSQL

El SGBD relacional orientado a objetos conocido como PostgreSQL está derivado del paquete Postgres escrito en Berkeley, distribuida bajo licencia BSD. PostgreSQL es el gestor de bases de datos de código abierto más avanzado hoy día, ofreciendo control de concurrencia multi-versión, soportando casi toda la

sintaxis SQL (incluyendo subconsultas, transacciones, tipos y funciones definidas por el usuario), contando también con un amplio conjunto de enlaces con lenguajes de programación como pueden ser C, C++, Java, Perl, TCL y Python [24]. Al mismo tiempo ofrece una potencia adicional sustancial al incorporar los siguientes cuatro conceptos adicionales básicos en una vía en la que los usuarios pueden extender fácilmente el sistema: clases, herencia, tipos y funciones.

1.4.5 Lenguajes de Programación

C++

Aunque en un principio C++ se plantea como una mejora de C ('C con clases'), en la actualidad es un lenguaje independiente, versátil, potente y general. Mantiene la ventaja de C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además ha eliminado algunas de las dificultades y limitaciones del C original.

Este es a la vez un lenguaje procedural (orientado a algoritmos) y orientado a objetos. Como lenguaje procedural se asemeja al C y es compatible con él. Como lenguaje orientado a objeto se basa en una filosofía completamente diferente, que exige del programador un completo cambio de mentalidad. Las características propias de la programación orientada a objetos (Object Oriented Programming, u OOP) de C++ son modificaciones mayores que si cambian radicalmente su naturaleza [25].

C#

C# fue diseñado por Microsoft, y posteriormente estandarizado por el organismo ECMA. Ese lenguaje intenta aprovechar, en la medida de lo posible, las características de la plataforma .NET en la cual es posible escribir código en muchos otros lenguajes, pero C# es el único que ha sido diseñado específicamente para ser utilizado en ella.

Además este lenguaje elimina muchos elementos añadidos por otros lo que facilitan su uso y compresión. También dentro de sus características está ser orientado a objetos y a componentes, ser seguro, moderno y de alto rendimiento [26].

La sintaxis y estructuración de C# es muy parecida a la de C++ o Java, puesto que la intención de Microsoft es facilitar la migración de código y su aprendizaje a los desarrolladores habituados a ellos. Sin embargo, su sencillez y el alto nivel de productividad son comparables a los de Visual Basic.

En general, C# es un lenguaje de programación que toma las mejores características de lenguajes preexistentes como Visual Basic, Java o C++ y las combina en uno solo. El hecho de ser relativamente reciente no implica que sea inmaduro, pues Microsoft ha escrito la mayor parte de la BCL (Basic Class Library) usándolo, por lo que su compilador es el más depurado y optimizado de los incluidos en el .NET Framework SDK.

Java

Java es un lenguaje de programación desarrollado por la empresa Sun Microsystems en 1991. Los programas en Java generalmente son compilados a un lenguaje intermedio llamado bytecode, que luego son interpretados por una máquina virtual de java (Java Virtual Machine “JVM”). Estos bytecodes fueron diseñados para cumplir varios requisitos:

- ✓ Compactos: Un programa traducido a bytecodes es de 2 a 3 veces más compacto que el mismo programa traducido a lenguaje de máquina de un procesador tradicional.
- ✓ Fáciles de interpretar: Un intérprete de bytecodes es sencillo y puede ser muy eficiente.
- ✓ Fáciles de traducir: Convertir de Java bytecodes a lenguaje de máquina optimizado para un procesador Intel ó RISC es relativamente sencillo. Algunas implementaciones de la JVM en vez de interpretar los bytecodes a la hora de ejecución, los traducen a lenguaje de máquina del procesador en el que se está ejecutando. A esto se le llama “Just In Time compilation” (JIT). Empleando tecnología JIT los programas en Java son tan rápidos como programas en C++.
- ✓ Fáciles de implementar en hardware: La JVM tiene una arquitectura sencilla, lo cual simplifica su implementación en hardware. Un procesador de Java puede ser barato.
- ✓ Verificables: Es posible verificar una secuencia de bytecodes antes de ejecutarlos para validar que cumplan con las reglas de acceso a memoria de Java.

Una de las cualidades más atractivas de Java, es ser un lenguaje orientado a objetos diseñado para ser multiplataforma y poder ser empleado el mismo programa en diversos sistemas operativos. Esta

característica, junto con la posibilidad de emplearlo para crear applets e insertarlos en páginas HTML, o mediante servlets y páginas jsp generar código HTML dinámico, así como la capacidad de acceder a bases de datos, hacen de Java uno de los lenguajes más utilizados en la actualidad.

1.4.6 Lenguaje Extensible de Marcas (XML)

XML es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C). El lenguaje XML es un lenguaje de marcas, basado en SGML, capaz de describir cualquier tipo de información en forma personalizada, aunque también es un metalenguaje de mercado capaz de describir lenguajes de marcas adecuadas para aplicaciones concretas [27]. Es una simplificación y adaptación del SGML, lo cual permite definir la gramática de lenguajes específicos. Por lo tanto XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. XML además de su aplicación en Internet, propone un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en: bases de datos, editores de texto, hojas de cálculo, etc.

Cada documento XML tiene una estructura tanto lógica como física:

- ✓ Físicamente, el documento está compuesto de unidades llamadas entidades. Una entidad puede referirse a otras entidades con el fin de causar su inclusión en el documento. Un documento comienza en una "raíz" o entidad documento.
- ✓ Lógicamente, el documento está compuesto por declaraciones, elementos, comentarios, referencias de carácter e instrucciones de proceso. Todas estas estructuras lógicas son indicadas mediante las correspondientes marcas.

1.4.7 Línea de Productos Software

Desarrollar aplicaciones de software de la forma tradicional es un proceso que resulta costoso en cuanto a la cantidad de recursos humanos necesarios, tiempo de desarrollo invertido y calidad del producto final. Para dar respuesta a estos problemas, han surgido propuestas que buscan reducir el esfuerzo en tiempo de codificación y dar paso al modelamiento de los sistemas en un mayor nivel de abstracción.

A Software Product Line (SPL) o Línea de Productos Software es un conjunto de sistemas intensivos en software que comparten un conjunto común, administrado de prestaciones para satisfacer las necesidades específicas de un segmento de mercado o misión y que son desarrollados a partir de un conjunto en común de activos centrales de un modo prescrito [28]. Estos activos centrales forman la base para la Línea de Productos y en ellos se incluyen, entre otros: la arquitectura, las especificaciones de requisitos, los planes y casos de prueba y componentes de software reutilizables.

Para ser exitosa con una Línea de Productos Software una organización debe alterar sus prácticas y técnicas de gestión, su estructura organizacional y su enfoque del negocio.

Las SPL´s permiten el ahorro en tiempo de desarrollo y proveen calidad a las aplicaciones, aprovechando el conjunto de características comunes que estas comparten. De igual modo, puede manejarse la variabilidad dentro de una SPL, permitiendo a los productos que pertenecen a una misma familia, tengan características particulares y que los hagan diferentes de otros productos [29].

Las principales ventajas de establecer estas SPL además de las citadas anteriormente se materializan en:

- ✓ Reutilización de la arquitectura: Todas las aplicaciones de la línea comparten básicamente la misma arquitectura, lo que admite reutilizar su diseño, su documentación, asegurar el cumplimiento de determinadas propiedades, etc.
- ✓ Reutilización de los componentes: Como se ha indicado, parte de los componentes son comunes a los distintos productos, o bien pueden obtenerse parametrizando componentes genéricos. Esto facilita notablemente la implementación de nuevos integrantes de la línea de productos.
- ✓ Fiabilidad: Los componentes empleados están exhaustivamente probados, al ser compartidos con aplicaciones que ya están en funcionamiento, lo que aprueba corregir defectos, aumentar su fiabilidad y calidad. Por otro lado, el uso de dichos componentes en aplicaciones que comparten la misma arquitectura permite garantizar que no haya pérdida de fiabilidad en la reutilización.
- ✓ Planificación: La experiencia repetida en el desarrollo de productos de la línea facilita hacer una planificación más fiable del proyecto y con mayores posibilidades de ser cumplida.
- ✓ Mejora del proceso: También es posible adaptar y especializar los propios procesos de desarrollo para la línea de productos.

1.5 Conclusiones

A lo largo de este capítulo, se han expuesto conceptos asociados a los procesos de gestión del flujo de información de la producción en empresas petroleras, que facilitan la comprensión de la investigación. Se fundamentan y caracterizan un conjunto de tecnologías, herramientas y tendencias asociadas a la solución propuesta para el problema científico identificado.

Capítulo 2: Tecnología a Utilizar

2.1 Introducción

Este capítulo es el resultado de la búsqueda y el análisis de la información vinculada al objeto de estudio, procesos a automatizar. Definiendo el patrón arquitectónico asociado y los de diseño, que se ajustan a la solución adecuada; también se definirá el lenguaje de programación y el gestor de Base de Datos, así como se tendrán en cuenta las herramientas CASE automáticas e IDE´s para el desarrollo.

2.2 Metodología de Desarrollo Seleccionada

Con el propósito de asegurar la producción de un software de alta calidad que se ajuste a las necesidades de los usuarios finales con unos costos y calendario predecibles. Se utilizará Rational Unified Process (RUP) como metodología de desarrollo debido a que es un proceso que proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades. RUP es una metodología de desarrollo de software que intenta integrar todos los aspectos a tener en cuenta durante todo el ciclo de vida del software, con el objetivo de hacer abarcables tanto pequeños como grandes proyectos software. Sin embargo los verdaderos aspectos definitorios de RUP tenidos en cuenta para su elección se resumen en:

- ✓ Guiado por casos de uso: La razón de ser del Sistema de Producción para Empresas Petroleras es servir a usuarios ya sean humanos u otros sistemas. Por otro lado, un caso de uso es una facilidad que el software debe proveer a sus usuarios. Estos reemplazan la antigua especificación funcional tradicional y constituyen la guía fundamental establecida para las actividades a realizar durante todo el proceso de desarrollo incluyendo el diseño, la implementación y las pruebas del sistema.
- ✓ Centrado en arquitectura: La arquitectura involucra los elementos más significativos del sistema y está influenciada entre otros por plataformas software, sistemas operativos, manejadores de bases de datos, protocolos, consideraciones de desarrollo como sistemas heredados y requerimientos no funcionales. Es similar a una radiografía del sistema que se quiere desarrollar, lo suficientemente completa como para que todos los implicados en el desarrollo tengan una idea clara de qué es lo que está construyendo, pero lo suficientemente simple como para que si se quita algo, el sistema siga funcionando. Se representa mediante varias vistas, 4+1, recibe este nombre porque lo forman

las vistas lógica, de implementación, proceso y despliegue, más la de casos de uso que es la que da cohesión a todas.

- ✓ Iterativo e Incremental: Para hacer más manejable este proyecto software se recomienda dividirlo en ciclos. Para cada ciclo se establecen fases de referencias, cada una de las cuales debe ser consideradas como un mini proyecto cuyo núcleo fundamental está constituido por una o más iteraciones de las actividades principales básicas. En concreto, con esta metodología se divide el proceso de desarrollo en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable y en las que se hace un mayor o menor hincapié en los distintas actividades.

RUP también es idóneo pues utiliza el lenguaje unificado de modelado (UML) para preparar todos los diseños del sistema de software. Las cuatro fases de vida de desarrollo de software tienen un objetivo esencial:

- ✓ Inicio: El objetivo en esta etapa es determinar la visión del proyecto.
- ✓ Elaboración: En esta etapa el objetivo es determinar la arquitectura óptima del sistema.
- ✓ Construcción: En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.
- ✓ Transición: El objetivo es llegar a obtener el despliegue del proyecto.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. El ciclo de vida que se desarrolla por cada iteración, es llevado a cabo bajo dos disciplinas: la disciplina de desarrollo y la disciplina de soporte. Estas disciplinas agrupan las principales actividades que se desarrollan en el desarrollo del sistema software.

2.3 Patrón de arquitecturas

En la solución propuesta se utilizará patrón de arquitectura Layers (Capas), con el objetivo de la separación de responsabilidades en cada una de las capas que conforman la aplicación y así lograr facilidades de desarrollo, enfatizando en la modificabilidad y escalabilidad del sistema.

Este patrón soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.

Además admite muy naturalmente optimizaciones, refinamientos y proporciona amplia reutilización, o sea, al igual que los tipos de datos abstractos se puede reutilizar un mismo nivel en varias aplicaciones. Admite también la estandarización, ya que el cambio de nivel no afecta el resto.

En la vista de la arquitectura se plantean tres capas verticales:

Capa de Presentación: Se limita a la navegabilidad y a gestionar todos aquellos aspectos relacionados con la lógica de presentación de la aplicación como: comprobación de datos de entrada, formatos de salida, internacionalización de la aplicación, etc.

Capa de Negocio o Dominio: Se encarga de contener todo el código que define las reglas de negocio (cálculo, validaciones). Surge del resultado del análisis funcional de la aplicación viene a ser la identificación del conjunto de reglas de negocio que abstraen el problema real a tratar. Estas son las que realmente tiene el peso del motor del sistema, dado que se basan en el funcionamiento del modelo real.

Capa de Acceso a Datos: Es la encargada de persistir las entidades que se manejan en el negocio, el acceso a los datos almacenados, la actualización, etc. Además puede ofrecer servicios relacionados con la persistencia o recuperación de información más complejos.

2.4 Patrones de Diseño Seleccionados

El Modelo de Diseño podría definir cientos o miles de clases software, y la aplicación podría requerir que se realicen cientos o miles de responsabilidades. Durante el diseño orientado a objetos, cuando se definen las interacciones entre los mismos, se toman decisiones sobre la asignación de responsabilidades a las clases del software. Con la selección de los patrones de diseño, el sistema tienden a ser más fácil de entender, mantener y ampliar, y existen más oportunidades para reutilizar componentes en futuras aplicaciones.

Los patrones de diseño permiten:

- ✓ Una mayor extensibilidad del software.
- ✓ Lograr un software más flexible y reutilizable.

- ✓ Alcanzar una mayor organización.
- ✓ Mayor facilidad para hacer cambios.
- ✓ Alcanzar una gran refactorización.
- ✓ Idioma común de intercambio de soluciones.
- ✓ Buenas prácticas en el desarrollo de software.

A continuación se exponen algunos de los patrones de diseño que estarán presente en el modelado del software.

2.4.1 Patrón Experto

Permite asignar una responsabilidad al experto en información.

Entre sus ventajas están:

- ✓ Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide.
- ✓ El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas, las cuales son fáciles de comprender y mantener.

2.4.2 Patrón Creador

Define quien debería ser el responsable de la creación de una nueva instancia de alguna clase. Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. Dentro de sus ventajas están:

- ✓ Guiar la asignación de responsabilidades relacionadas con la creación de objetos.
- ✓ Se encuentra un creador que necesite conectarse al objeto creado en alguna situación, eligiéndolo como el creador, se favorece el bajo acoplamiento.

- ✓ Se brinda un soporte al bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización. Es probable que el acoplamiento no aumente, pues la clase creada tiende a ser visible a la clase creador, debido a las asociaciones actuales que llevaron a elegirla como el parámetro adecuado.

2.4.3 Patrón Bajo Acoplamiento

Este patrón define como dar soporte a una dependencia escasa y a un aumento de la reutilización, enfocándose en asignar una responsabilidad para mantener bajo acoplamiento.

El bajo acoplamiento estimula la asignación de responsabilidades de forma tal que, la inclusión de estas no incremente el acoplamiento, creando clases más independientes y con mayor resistencia al impacto de los cambios, que aumentan la productividad y la posibilidad de reutilización.

Es válido aclarar que este patrón no puede verse de forma independiente a los patrones Experto y Alta Cohesión, sino más bien incluirse como otro de los principios del diseño que influyen de forma determinante a la hora de la asignación de responsabilidades.

Ventajas

- ✓ No afectan los cambios en otros componentes.
- ✓ Fácil de entender de manera aislada.
- ✓ Conveniente para reutilizar.

2.4.4 Patrón Alta Cohesión

El propósito de este patrón es asignar una responsabilidad de manera que la cohesión permanezca alta. Posibilitando mucha facilidad de mantenimiento, comprensión y uso. Brinda un alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos de las clases que componen el software.

Ventajas

- ✓ Se incrementa la claridad y facilita la comprensión del diseño.
- ✓ Se simplifican el mantenimiento y las mejoras.
- ✓ Se soporta a menudo bajo acoplamiento.

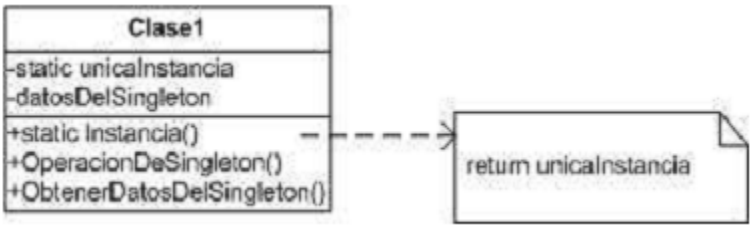
2.4.5 Patrón Singleton

Con la utilización del patrón Singleton se garantiza que una clase sólo tenga una instancia y proporciona un punto de acceso global a ella.

Usarlo cuando:

- ✓ Deba existir exactamente una instancia de una clase y ésta debe ser accesible a los clientes desde un punto de acceso conocido.
- ✓ La única instancia debería ser extensible mediante herencia y los clientes deberían ser capaces de usar una instancia extendida sin modificar su código.

Estructura



Ventajas

- ✓ 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 almacenan las instancias.

- ✓ Permite el refinamiento de operaciones y la representación: Se puede crear una subclase de la clase Singleton, y es fácil configurar una aplicación con una instancia de esta clase extendida, incluso en tiempo de ejecución.
- ✓ Permite un número variable de instancias: Hace que sea fácil permitir más de una instancia de la clase. Solo se necesitaría cambiar la operación que otorga acceso a la instancia del Singleton.

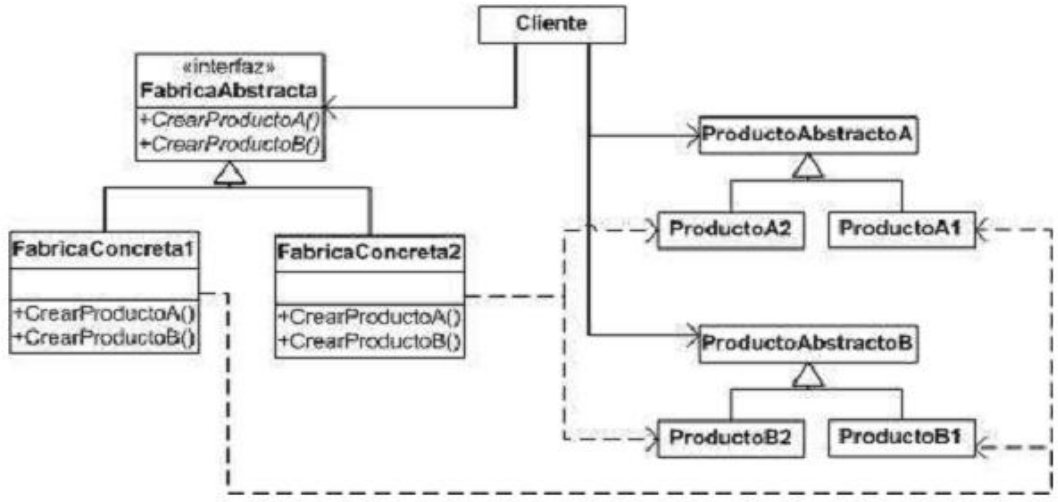
2.4.6 Patrón Abstract Factory

Con la utilización de este patrón se proporciona una interfaz para crear familias de objetos relacionados o que dependen entre sí, sin especificar sus clases concretas.

Usarlo cuando:

- ✓ Un sistema debe ser independiente de cómo se crean, componen y representan sus objetos.
- ✓ Una familia de objetos relacionados está diseñada para ser usada conjuntamente, y es necesario hacer cumplir esta restricción.
- ✓ Quiere proporcionar una biblioteca de clases de productos y solo quiere revelar sus interfaces, no sus implementaciones.

Estructura



Dentro de su estructura se encuentran diferentes clases participantes:

- ✓ FabricaAbstracta: Declara una interfaz para operaciones que crean objetos producto, abstractos.
- ✓ FabricaConcreta: Implementa las operaciones para crear objetos producto concreto.
- ✓ ProductoAbstracto: Declara una interfaz para un tipo de objeto producto.
- ✓ ProductoConcreto: Define un objeto producto para que sea creado por la fábrica correspondiente. implementa la interfaz ProductoAbstracto.
- ✓ Cliente: Sólo usa interfaces declaradas por las clases FabricaAbstracta y ProductoAbstracto.

Ventajas

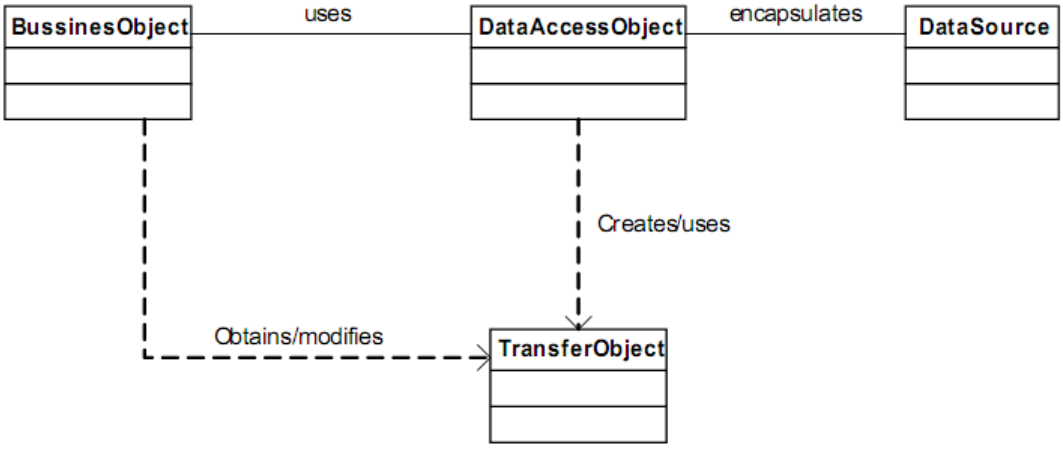
- ✓ Aísla las clases concretas: Ayuda a controlar las clases de objetos que crea una aplicación, encapsula la responsabilidad y el proceso de creación de objetos. Aísla a los clientes de las clases de implementación, manipulan las instancias a través de sus interfaces abstractas. Los nombres de las clases quedan aisladas en la implementación de la fábrica concreta; no aparecen en el código cliente (Ventaja).
- ✓ Facilita el intercambio de familias de productos: La clase de una fábrica concreta solo aparece una vez en una aplicación, cuando se crea. (Ventaja).
- ✓ Promueve la consistencia: Se diseñan objetos en una familia para trabajar juntos (Ventaja).

2.4.7 Patrón Data Access Object (DAO)

Es un patrón de diseño directamente basado en la “separation of concerns”, en el que se separa la persistencia de datos del resto de funcionalidades del sistema.

Este patrón permite recoger los datos y almacenarlos en una base de datos. También, posibilita hacer una aplicación lo más independiente posible de una base de datos concreta.

Estructura



El DAO es el punto de entrada al almacén de datos y proporciona operaciones de tipo CRUD (Create-Read-Update-Delete), las cuales implementa según su necesidad.

Ventajas

- ✓ Proporciona la independencia del almacén de datos, debido a que el cambio de motor de base de datos o el paso de usar un pequeño archivo XML a usar una base de datos relacional para almacenar datos solo afectará al DAO y no a las clases encargadas de la lógica de negocio o de presentación. Se suele usar el patrón Factory para poder instanciar los DAOs reduciendo al máximo la dependencia del DAO concreto a crear.
- ✓ DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes.
- ✓ La interfaz expuesta por DAO no cambia cuando cambia la implementación de la fuente de datos subyacente (diferentes esquemas de almacenamiento).

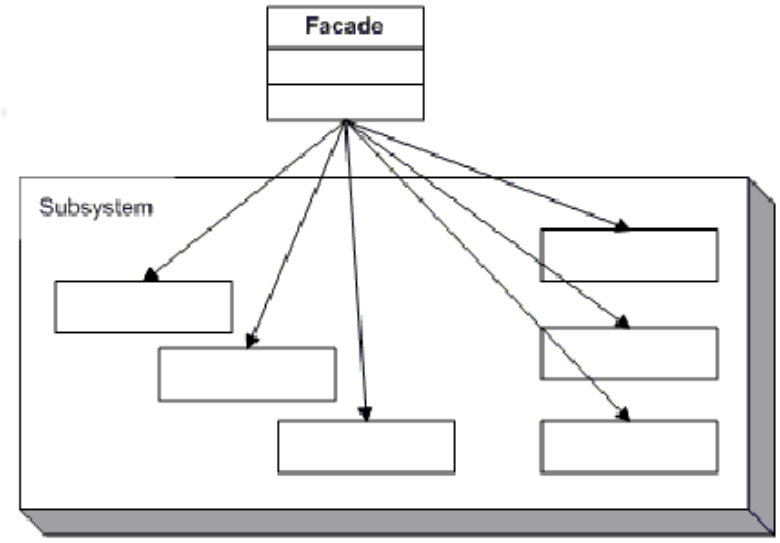
2.4.8 Patrón Facade o Fachada

Su objetivo es proporcionar una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La “fachada” satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas.

Cuando usarlo:

- ✓ Cuando se pretende proporcionar una interfaz simple para un subsistema complejo.
- ✓ Cuando existen muchas dependencias entre los clientes y las clases que implementan una abstracción. Una “fachada” proporciona al subsistema independencia y portabilidad.
- ✓ Cuando se pretende estructurar en capas el subsistema (cada capa tendrá su propia “fachada”).

Estructura



Ventajas

- ✓ Al separar al cliente de los componentes del subsistema, se reduce el número de objetos con los que el cliente trata, facilitando así el uso del subsistema.
- ✓ Se promueve un acoplamiento débil entre el subsistema y sus clientes, eliminándose o reduciéndose las dependencias.
- ✓ No existen obstáculos para que las aplicaciones usen las clases del subsistema que necesiten. De esta forma se puede elegir entre facilidad de uso y generalidad.

2.5 Patrones de idioma

Se utilizan en los flujos de implementación, mantenimiento y despliegue, comúnmente reconocidos como estándares de codificación. Describen como codificar y representar operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo, brindan legibilidad y claridad.

A continuación, se describen instrucciones a seguir en el desarrollo y codificación del sistema SISPEP.

Tipos de Identificadores	Reglas para nombres	Ejemplos
Paquetes	El prefijo del nombre de un paquete se escribe siempre con letras ASCII en minúsculas, y debe ser uno de los nombres de dominio de alto nivel, actualmente com, edu, gov, mil, net, org, o uno de los códigos ingleses de dos letras que identifican cada país como se especifica en el ISO Standard 3166, 1981.	com.sun.eng com.apple.quicktime.v2
Clases	Los nombres de las clases deben ser sustantivos, cuando son compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas. Intentar mantener los nombres de las clases simples y descriptivas. Usar palabras completas, evitar acrónimos y abreviaturas (a no ser que la abreviatura sea mucho más conocida que el nombre completo).	class Cliente; class ImagenAnimada;
Interfaces	Los nombres de las interfaces siguen la misma regla que las clases.	interface ObjetoPersistente; interface Almacen;

Métodos	Los métodos deben ser verbos, cuando son compuesto tendrán la primera letra en minúscula, y la primera letra de las siguientes palabras que lo forma	ejecutar();
		
		ejecutarRapido();

	en mayúscula.	
Variables	Excepto las constantes, todas las instancias y variables de clase o método empezarán con minúscula. Las palabras internas que lo forman (si son compuestas) empiezan con su primera letra en mayúsculas. Los nombres de variables no deben empezar con los caracteres subguión "_" o signo del dólar "$", aunque ambos están permitidos por el lenguaje. Los nombres de las variables deben ser cortos pero con significado. La elección del nombre de una variable debe ser un mnemónico, designado para indicar a un observador casual su función. Los nombres de variables de un solo carácter se deben evitar, excepto para variables índices temporales. Nombres comunes para variables temporales son i, j, k, m, y n para enteros; c, d, y e para caracteres.	int i; char c; float miAnchura;

Constantes	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un subguión ("_"). (Las constantes ANSI se deben evitar, para facilitar su depuración.)	static final int ANCHURA_MINIMA = 4;
		
		static final int ANCHURA_MAXIMA=99;

2.6 UML como Lenguaje de Modelado

El objetivo del modelado del sistema SISPEP es capturar las partes esenciales del mismo. Para facilitar este modelado, se realiza una abstracción y se plasma en una notación gráfica, esto se conoce como modelado visual. El modelado visual permite manejar la complejidad del sistema, analizar y diseñar, pues es necesario abstraer la complejidad en modelos que los desarrolladores puedan entender.

UML sirve para el modelado completo de sistemas complejos, tanto en el diseño de los sistemas software como para la arquitectura hardware donde se ejecuten.

Otra característica de este modelado visual es que sea independiente del lenguaje de implementación, de tal forma que los diseños realizados usando UML se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML (principalmente lenguajes orientados a objetos). Permitiendo así una mayor escalabilidad del software y reutilización del diseño.

UML es además un método formal de modelado. Esto aporta las siguientes ventajas:

- ✓ Mayor rigor en la especificación.
- ✓ Permite realizar una verificación y validación del modelo realizado.

Dentro de sus principales funciones se encuentran:

- ✓ UML permite **visualizar** de una forma gráfica un sistema de forma que otro lo puede entender.
- ✓ UML permite **especificar** cuáles son las características de un sistema antes de su construcción.
- ✓ A partir de los modelos especificados se pueden **construir** los sistemas diseñados.
- ✓ Los propios elementos gráficos sirven como **documentación** del sistema desarrollado que pueden servir para su futura revisión.

2.7 Herramienta CASE Seleccionada

Para el modelamiento del sistema se utilizará Visual Paradigm la cual es una poderosa herramienta CASE, que al igual que Rational Rose, utiliza UML para el modelado. Esta herramienta posibilita el desarrollo completo del software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de la aplicación proporcionando una alta calidad, mejoras y a un menor tiempo de desarrollo.

Visual Paradigm permite además, dibujar todos los tipos de diagramas de clases, generar código desde diagramas y viceversa, también generar documentación la cual puede ser utilizada para la conformación de los documentos del Expediente de Proyecto. La herramienta UML CASE también facilita abundantes

tutoriales de UML, demostraciones interactivas y proyectos UML. Es igualmente una herramienta multiplataforma, muy sencilla de usar, fácil de instalar y actualizar. Posibilita un entorno de creación de diagramas para UML 2.x.

Para el modelado del sistema la versión que se utilizará será Enterprise Edition v3.4, debido a las características que brinda la misma:

- ✓ Modelamiento de los Proceso de Negocio.
- ✓ Modelamiento de Base de Dato.
- ✓ Modelo Relacional de Objetos.
- ✓ Modelamiento Visual.
- ✓ Diseño de Interfaz de Usuario.
- ✓ Integración a Entornos de Desarrollo.
- ✓ Ingeniería Directa e Inversa.
- ✓ Generación de Código.
- ✓ Interoperabilidad.
- ✓ Integración con CVS (Concurrent Version System) y Subversion.

2.8 Lenguaje de Programación Seleccionado

Java se distingue de otros lenguajes, por ser una plataforma completa de desarrollo. Consta de un gran conjunto de componentes que se pueden reutilizar y mecanismos para extenderlos, facilitando la vida a los desarrolladores. Aunque al mismo tiempo obliga a tener buenas prácticas, buenos patrones de diseño a diversos problemas recurrentes de desarrollo.

La plataforma Java se ha establecido en la industria como una de las principales herramientas de construcción de aplicaciones en las corporaciones, otorgándoles diversos beneficios, así como: un universo de aplicaciones, framework y estándares generados alrededor de la plataforma, que la complementan y extienden.

Al ser un lenguaje libre, dispone de un sin número de características que lo convierte en la herramienta ideal para la creación aplicaciones distribuidas.

Características de Java:

- ✓ Simple: El único requerimiento para aprender Java es tener una comprensión de los conceptos básicos de la programación orientada a objetos. Java es más complejo que un lenguaje simple, pero más sencillo que cualquier otro entorno de programación. El único obstáculo que se puede presentar, es conseguir comprender la programación orientada a objetos, aspecto que, al ser independiente del lenguaje, se presenta como insalvable.
- ✓ Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos.
- ✓ Robusto: Este lenguaje realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo. Además obliga a la declaración explícita de métodos, reduciendo así las posibilidades de error. Igualmente maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de memoria.
- ✓ Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets para establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.
- ✓ Indiferente a la arquitectura: Está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. Para acomodar requisitos de ejecución tan variopintos, el compilador de Java genera bytecodes: un formato intermedio indiferente a la arquitectura diseñada para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.
- ✓ Portable: Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.
- ✓ Es multihilo: El beneficio de ser multihilo consiste en un mejor rendimiento interactivo y mejor comportamiento en tiempo real. Aunque el comportamiento en tiempo real está limitado a las capacidades del sistema operativo subyacente (Unix, Windows, etc.), aún supera a los entornos de flujo único de programa (single-threaded) tanto en facilidad de desarrollo como en rendimiento.

2.9 Framework de Desarrollo Seleccionados

Los framework son construidos teniendo en cuenta lenguajes orientados a objetos, esto permite una mejor modularización de los componentes y óptima reutilización de código. Conjuntamente, en la mayoría de los casos, un framework implementa uno o más patrones de diseño de software que aseguran la escalabilidad del producto.

A partir de la organización estructural del sistema en tres capas verticales, se propone la utilización de tres framework de código abierto, distribuidos en cada capa de la aplicación con funcionalidades bien definidas y componentes implementados, los cuales permiten desarrollar componentes propios para cada capa de la aplicación:

Capa de Presentación. Framework Swing:

Características:

- ✓ Arquitectura Modelo-Vista-Controlador (MVC): Swing usa MVC como principal diseño detrás de cada uno de sus componentes. Esta arquitectura divide los componentes en tres elementos y cada uno juega un rol fundamental en su comportamiento. La arquitectura MVC da lugar a todo un enfoque de desarrollo muy arraigado en los entornos gráficos de usuario realizados con técnicas orientadas a objetos.
    - Modelo: Se refiere al modelo de datos que utiliza el objeto. Es la información que se manipula mediante el objeto Swing. Ejemplo: un array de cadenas que contenga los meses del año.
    - Vista: Es cómo se muestra el objeto en la pantalla. Ejemplo: mostrar el array en un cuadro combinado de Windows (JComboBox).
    - Controlador: Es lo que define el comportamiento del objeto. Ejemplo: capa software que permite captura el clic del mouse a fin de mostrar los elementos contenidos en el cuadro combinado de Windows.
- ✓ Potentes manipuladores de texto: Además de campos y áreas de texto, se presentan campos de sintaxis oculta JPassword, y texto con múltiples fuentes JTextPane.
- ✓ Componentes para tablas y árboles de datos: Mediante las clases JTable y JTree.

- ✓ Escritorios virtuales: Se pueden crear escritorios virtuales o "interfaz de múltiples documentos" mediante las clases JDesktopPane y JInternalFrame.
- ✓ Gestión mejorada de la entrada del usuario: Se pueden gestionar combinaciones de teclas en un objeto KeyStroke y registrarlo como componente. El evento se activará cuando se pulse dicha combinación si está siendo utilizado el componente, la ventana en que se encuentra o algún hijo del componente.
- ✓ Aspecto modificable (look and feel): Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Window Win32).

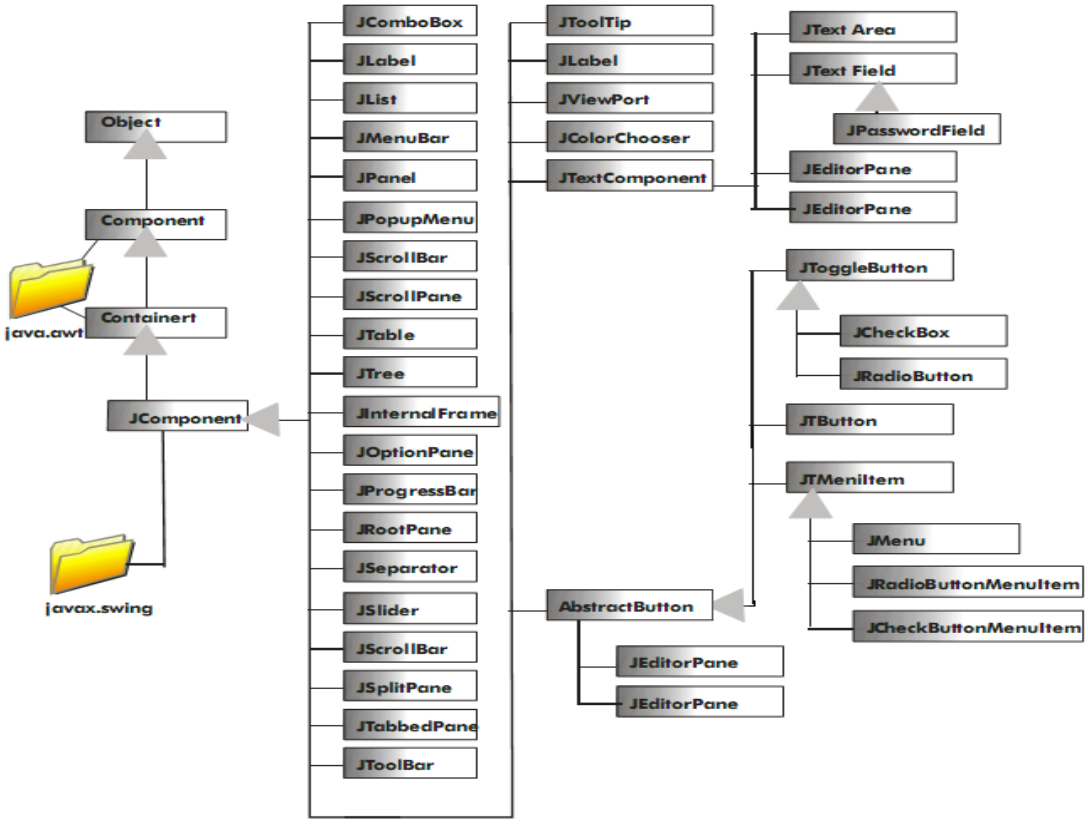


Figura 1: Estructura de Carpetas del Framework Swing

A continuación se detallan algunos de los paquetes que Swing presenta dentro de sus librerías:

- ✓ javax.swing: Contiene el núcleo de los componentes de swing, incluyendo la mayoría de las interfaces de modelos y clases de apoyo.
- ✓ javax.swing.border: Contiene la definición para la clase de borde abstracta para manejar los bordes de los componentes.
- ✓ javax.swing.event: Define varios nuevas interfaces y eventos que los componentes Swing utilizan para comunicar información asincrónica entre clases, permitiendo el control de los eventos sobre los componentes.
- ✓ javax.swing.pending: Contiene un conjunto de componentes que no están listos para usarse por primera vez, pero que lo pueden estar en un futuro.
- ✓ javax.swing.plaf: Define los elementos únicos que componen el aspecto visual de cada componente Swing.
- ✓ javax.swing.table: Brinda modelos y vistas para crear tablas, estilos de tablas y sus formas.
- ✓ javax.swing.text: Brinda clases basadas en textos e interfaces que soporten un diseño común conocido como *document/view.*
- ✓ javax.swing.tree: Define modelos y vistas para el componente de jerarquía de árbol.
- ✓ javax.swing.undo: Contiene la funcionalidad necesaria para implementar funciones para revertir operaciones.
- ✓ javax.swing.preview: Contiene las clases necesarias para el tratamiento de ficheros en los distintos tipos de sistemas operativos.

Capa de Negocio. Framework Spring:

Es un framework contenedor liviano basado en la técnica Inversión de Control (IoC de sus siglas en inglés Inversion of Control) y una implementación de desarrollo según el paradigma de Aspect Oriented Programming (AOP).

El objetivo central de Spring es permitir que objetos de negocio y de acceso a datos sean reusables y no atados a servicios J2EE específicos. Estos objetos pueden ser reutilizados tanto en entornos J2EE (web o EJB), aplicaciones Standalone y entornos de pruebas.

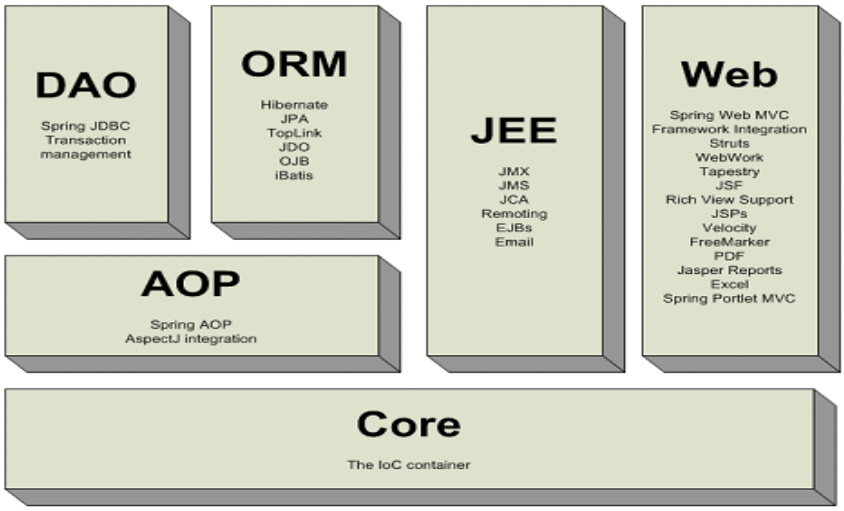


Figura 2: Componetes de la Arquitectura del Framework Spring

Explicación de los componentes de la Arquitectura de Spring:

- ✓ DAO: Accesos JDBC con manejo de transacciones (desde el módulo AOP).
- ✓ ORM: Integración con framework de persistencia como Hibernate, JDO, etc.
- ✓ JEE: Acceso e interacción con servicios Enterprise.
- ✓ Web: Provee un contexto apropiado para el desarrollo de aplicaciones web e integración con otros framework (Struts, JSF, Tapestry, etc.)

Características:

- ✓ Inversión de Control (Inversion of Control IoC): Promueve el bajo acoplamiento a partir de la inyección de dependencias (Dependency Injection DI) entre los objetos (relaciones). Permite una sofisticada administración de configuración para POJOs y trabaja con otras partes de Spring para proveer servicios como la administración de la configuración.
- ✓ Programación Orientada a Aspectos (AOP): Permite comportamientos que deberían ser esparcidos de otra manera a través de diferentes métodos para ser modularizados en un simple lugar. Spring usa la AOP para implementar importantes servicios tales como: administración de transacciones

declarativas y además es usado para implementar códigos estándar que debería de otra manera ser esparcido entre las clases de la aplicación.

- ✓ Administración de transacciones: Presenta una abstracción de transacciones que puede mover transacciones globales JTA (manipuladas por un servidor de aplicaciones) o transacciones locales usando JDBC, Hibernate, JDO u otro API de acceso a datos. Estas abstracciones presentan un consistente modelo de programación en una variedad de ambientes, y es la base de la administración de transacciones programáticas y declarativas usadas por Spring.
- ✓ Facilita buenas prácticas de programació: La programación a interfaces y el uso de un contenedor de Inversión de Control reducen grandemente la complejidad del código a interfaces, más que a clases. El uso de los objetos a través de estas interfaces protege los requerimientos, los cuales pudieran cambiar en el desarrollo de la aplicación.
- ✓ POJO: (Plain Old Java Object) revalora la simplicidad de las clases Java aportando manejo de transacciones de forma no intrusiva.
- ✓ XML: Configuración basada en archivos XML.
- ✓ Seguridad: Maneja la seguridad como un requerimiento no funcional implementado como un aspecto (AOP) a través del framework Acegi.
- ✓ Testing: Provee un paquete de prueba específico para componentes del framework e integrado con JUnit.

Capa de Acceso a Datos. Framework Hibernate:

Hibernate es un entorno de trabajo que tiene como objetivo facilitar la persistencia de objetos Java en bases de datos relacionales, y al mismo tiempo la consulta de estas bases de datos para obtener objetos.

Es posible utilizar Hibernate en entornos manejados (servidores de aplicaciones J2EE), o en entornos no manejados (aplicaciones de escritorio, de consola, contenedores de servlets).

Sus principales características son:

- ✓ Permite expresar consultas utilizando SQL nativo o consultas basadas en criterios.

- ✓ Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los más populares servidores de aplicaciones J2EE y contenedores web, y por supuesto también puede utilizarse en aplicaciones de escritorio.
- ✓ Puede operar proporcionando persistencia de una manera transparente para el desarrollador.
- ✓ Soporta el paradigma de orientación a objetos de una manera natural: herencia, polimorfismo, composición y el framework de colecciones de Java.
- ✓ Provee un sistema de caché de dos niveles y puede ser usado en un clúster. Permite inicialización perezosa de objetos y colecciones.
- ✓ Proporciona el lenguaje Hibernate Query Languaje (HQL) el cual provee una independencia del SQL de cada base de datos, tanto para el almacenamiento de objetos como para su recuperación.
- ✓ Presenta un potente mecanismo de transacciones de aplicación, llegando incluso a permitir la interacciones largas (aquellas que requieren la interacción con el usuario durante su ejecución).
- ✓ Soporta los diversos tipos de generación de identificadores que proporcionan los sistemas gestores de bases de datos así como, generación independiente de la base de datos, incluyendo identificadores asignados por la aplicación o claves compuestas.

El API de Hibernate es una arquitectura de dos capas (Capa de persistencia y Capa de Negocio). La capa de negocio está situada sobre la capa de persistencia, debido a que actúa como un cliente de la capa de persistencia.

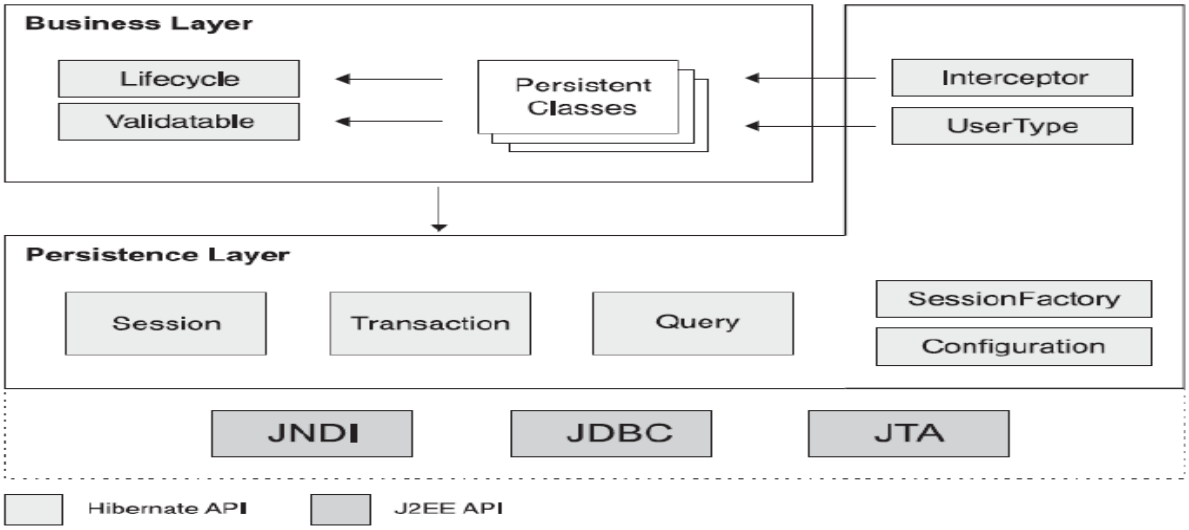


Figura 3: Arquitectura del Framework Hibernate

Las interfaces mostradas se clasifican de la siguiente forma:

- ✓ Interfaces llamadas por la aplicación para realizar operaciones básicas:
  - Session: Interfaz primaria utilizada en cualquier aplicación Hibernate (SessionFactory).
  - Transaction: Abstracción de una transacción concreta (JDBC, JTA, CORBA). Se utiliza para el control de transacciones.
  - Query: Permite realizar peticiones a la base de datos y controla cómo se ejecuta dicha petición (query). Las peticiones se escriben en HQL o en el dialecto SQL nativo de la base de datos que se está utilizando. Una instancia Query se utiliza para enlazar los parámetros de la petición, limitar el número de resultados devueltos por la petición y para ejecutar dicha petición.
- ✓ Interfaces llamadas por el código de la infraestructura de la aplicación para configurar Hibernate. La más importante es la clase Configuration: Se utiliza para configurar y "arrancar" Hibernate. La aplicación utiliza una instancia de Configuration para especificar la ubicación de los documentos que indican el mapeado de los objetos y propiedades específicas de Hibernate, y a continuación crea la SessionFactory.

- ✓ Interfaces callback que permiten a la aplicación reaccionar ante determinados eventos que ocurren dentro de la aplicación, tales como Interceptor, Lifecycle, y Validatable.
- ✓ Interfaces que permiten extender las funcionalidades de mapeado de Hibernate, como por ejemplo: UserType, CompositeUserType, e IdentifierGenerator.

Además, Hibernate hace uso de APIs de Java, tales como JDBC, JTA (Java Transaction Api) y JNDI (Java Naming Directory Interface).

La configuración de Hibernate se realiza por medio del fichero hibernate.cfg.xml o fichero hibérnate.properties (ejemplo siguiente). En él se especifican propiedades (como el dialecto, el driver de conexión) y los ficheros de mapeo.

```
hibernate.dialect net.sf.hibernate.dialect.HSQLDialect
hibernate.connection.driver_class org.hsqldb.jdbcDriver
hibernate.connection.username sa
hibernate.connection.password
hibernate.connection.url jdbc:hsqldb:hsql://localhost
```

Figura 4: Estructura de fichero de configuración .properties

En Hibernate, cada clase persistente necesita un fichero XML de mapeo. A partir de él, obtiene toda la información necesaria para realizar las operaciones CRUD. Este fichero tiene extensión *.hbm.xml en el que se describe cómo se relacionan clases, tablas, propiedades y columnas. Ejemplo:

```
<?xml version="1.0"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-2.0.dtd">

<hibernate-mapping>
  <class name="escuela.beans.Categoria" table="categorias">
    <id name="id" type="integer" column="ID" unsaved-value="-1">
      <generator class="identity"/>
    </id>

    <property name="categoria" column="CATEGORIA"
              type="string"
              unique="true"
              not-null="true"/>
  </class>
</hibernate-mapping>
```

Figura 5: Fichero de Mapeo

Otros Framework

JUnit

JUnit permite realizar la ejecución de clases Java de manera controlada y de esta manera poder evaluar si el funcionamiento de cada uno de los métodos de la las mismas se comporta como se espera. En función de algún valor de entrada se evalúa el valor de retorno esperado.

Este framework es también un medio de controlar las pruebas de regresión necesarias cuando una parte del código ha sido modificado, y se desea ver que el nuevo código cumple con los requerimientos anteriores además de que no se ha alterado su funcionalidad después de la nueva modificación.

El propio framework incluye formas de ver los resultados (runners) que pueden ser en modo texto, gráfico (AWT o Swing) o como tarea en Ant.

Acegi Security System para Spring

Acegi Security proporciona servicios de seguridad dentro de Spring Framework. Aunque no forma parte directa de Spring está íntimamente ligado con éste.

Proporciona una serie de características clave:

Integración: Utiliza los mecanismos de configuración de Spring y se integra completamente con él.

- ✓ Seguridad: Maneja la seguridad a nivel de instancia de objetos del dominio. Proporciona un completo paquete Lista de Control de Acceso (Access Control List ACL) con características que incluyen máscaras de bits, herencia de permisos, un repositorio utilizando JDBC, caché y un diseño *pluggable* utilizando interfaces.
- ✓ Mantiene los objetos libres de código de seguridad: Muchas aplicaciones necesitan proteger datos a nivel de objeto basándose en cualquier combinación de parámetros (usuario, hora del día, autoridad del usuario, método que es llamado, parámetros del método invocado,...). Acegi brinda esta flexibilidad sin necesidad de añadir código a los objetos de negocio.

- ✓ Métodos de almacenamiento de la información de autenticación: Acegi incluye la posibilidad de obtener los usuarios y permisos utilizando ficheros XML, fuentes de datos JDBC o implementando un interfaz DAO para obtener la información de cualquier otro lugar.
- ✓ Caché: Utilizando EHCache o otra implementación propia se puede hacer caché de la información de autenticación, evitando que la base de datos o cualquier otro tipo de fuente de información no sea consultado repetidamente.

EHCache

EhCache es uno de los frameworks más utilizados para el trabajo con objetos en Java. Consta de "regiones de caché" en donde se ubican objetos. Estas regiones se configuran en un archivo XML para determinar el tiempo de expiración, cantidad máxima de elementos, etc.

2.10 Gestor de Base de Datos Seleccionado

Debido a las políticas de migración a software libre del país y teniendo en cuenta que se trabajará en el área del software para CUPET, y que la información que se manipula es sensible y relevante, por ser información relativa a la economía del país, conviene entonces, escoger un SGBD que cumpla con las funciones de respaldo, y PostgreSQL deviene candidato ideal, además de que no se ha querido encarecer el sistema final usando un gestor como Oracle o SQLServer.

PostgreSQL es un gestor de código abierto, es decir, no está controlado por una sola compañía, sino que cuenta con comunidad global de desarrolladores y compañías para su evolución.

Las principales mejoras en PostgreSQL v 8.2 influyen grandemente en su elección, estas incluyen:

- ✓ Los bloqueos de tabla han sido sustituidos por el control de concurrencia multi-versión, el cual permite a los accesos de sólo lectura continuar leyendo datos consistentes durante la actualización de registros, y permite copias de seguridad en caliente desde pg_dump mientras la base de datos permanece disponible para consultas. Esta técnica elimina la necesidad de hacer bloqueos de lectura y asegura los principios ACID (atomicidad, consistencia, aislamiento y durabilidad) de la base de datos de una manera eficiente.

- ✓ Se han implementado importantes características del motor de datos, incluyendo subconsultas, valores por defecto, restricciones a valores en los campos (constraints) y disparadores (triggers).
- ✓ Se han añadido funcionalidades en línea con el estándar SQL92, incluyendo claves primarias, identificadores entrecomillados, forzado de tipos cadena literal, conversión de tipos y entrada de enteros binarios y hexadecimales.
- ✓ Los tipos internos han sido mejorados, incluyendo nuevos tipos de fecha/hora de rango amplio y soporte para tipos geométricos adicionales.
- ✓ Establecer condiciones o CHECKs para validar las entradas de datos
- ✓ Permitir transacciones, es decir, múltiples operaciones de tabla o registros de manera segura.
- ✓ Administración de Grupos de usuarios, y soporte nativo SSL.
- ✓ Múltiples lenguajes para procedimientos almacenados (incluyendo el nativo PL/PgSQL, PL/PHP, PL/Perl y PL/Python)
- ✓ La velocidad del código del motor de datos ha sido incrementada aproximadamente en un 20-40%, y su tiempo de arranque ha bajado el 80% desde que la versión 6.0 fue lanzada.
- ✓ Incluye Slony-I: Slony-I es un sistema de replicación maestro a múltiples esclavos, que soporta actualizaciones en cascada y promociones de esclavos. La gran razón para incluirla como herramienta de replicación es que de esta forma se cuenta con un sistema que posee las capacidades necesarias para replicar grandes bases de datos a un número razonable de sistemas esclavos. Es un sistema previsto para centros de datos y sitios de respaldo, donde la situación normal es que todos los nodos se encuentres disponibles, y puedan ser asegurados.

2.11 Otras Herramientas

2.11.1 Hibernate Tools

Hibernate Tools es un conjunto de herramientas enteramente para trabajar con el framework Hibernate, implementado como una suite integrada de plug-in para NetBeans y Eclipse.

Este plug-in tiene las siguientes características disponibles:

- ✓ Editor de mapeos: Es un editor para los archivos de mapeos XML de Hibernate, soportando auto completamiento y sintaxis resaltada. Soporta incluso auto completamiento semántico para nombres de clases, propiedades, tablas y columnas.
- ✓ Consola: La perspectiva de consola de Hibernate permite configurar conexiones a base de datos, provee visualización de clases y sus relaciones. Admite además, ejecutar consultas en formato del lenguaje de consultas de Hibernate (HQL) interactivamente contra la base de datos y mostrar los resultados.
- ✓ Ingeniería inversa: Es su característica más importante, permitiendo generar las clases del modelo de dominio y los archivos de mapeos de Hibernate a partir de una base de datos.
- ✓ Presenta asistentes como: generador de archivos de configuración rápidamente, configuración de la consola, entre otros.
- ✓ Tareas de Ant: Permite ejecutar la generación de esquemas, mapeos o código Java como parte de su construcción.

2.11.2 JasperReport

JasperReports es la mejor herramienta de código libre en Java para generar reportes. Puede entregar ricas presentaciones o diseños en la pantalla en formato PDF, HTML, RTF, XLS, CSV y XML, los cuales se pueden imprimir. Otra ventaja de utilizar JasperReport es que se integra perfectamente con el JFreeChart que es una librería libre para la generación de todo tipo de gráficas.

Este generador de reportes se puede utilizar en una gran variedad de aplicaciones de Java, incluyendo J2EE o aplicaciones Web para generar contenido dinámico.

2.11.3 iReport

iReport es un diseñador visual de código libre para JasperReports escrito en Java. Es un programa que ayuda a los usuarios y desarrolladores que usan la librería JasperReports para diseñar reportes visualmente. A través de una interfaz rica y simple de usar, iReport provee las funciones más importantes para crear reportes amenos en poco tiempo.

iReport provee una interfaz visual para construir reportes, generar archivos “jasper” y “print” de prueba. iReport nació como una herramienta de desarrollo, pero puede utilizarse como una herramienta de oficina para adquirir datos almacenados en una base de datos, sin pasar a través de alguna otra aplicación.

2.11.4 *JFreeChart*

Es una librería que permite crear gráficas para utilizar en aplicaciones Java. Estas gráficas no solo se muestran como imagen dentro de la aplicación sino que también se pueden exportar en formato jpg o pdf.

Además esta librería es software libre bajo licencia LGPL lo que permite la creación de aplicaciones comerciales.

2.11.5 Subversion

Subversion es un sistema de control de versiones libre y de código fuente abierto. Es decir, Subversion maneja ficheros y directorios a través del tiempo. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos.

Este control de versiones puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Brinda la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomentando de esta forma la colaboración. Se puede progresar más rápidamente sin un único conducto por el cual deban pasar todas las modificaciones. Y puesto que el trabajo se encuentra bajo el control de versiones, no hay razón para temer porque la calidad del mismo vaya a verse afectada por la pérdida de ese conducto único si se ha hecho un cambio incorrecto a los datos, simplemente deshaga ese cambio [30].

2.11.6 TortoiseSVN

TortoiseSVN es un cliente gratuito de código abierto para el sistema de control de versiones Subversion. El mismo 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 [31].

2.11.7 NetBean 6.5

Es un IDE gratuito de código abierto para desarrolladores de software. Es fácil de instalar y de uso instantáneo y se ejecuta en varias plataformas incluyendo Windows, Linux y Mac OSX y Solaris. Soporta la programación en lenguajes como: C/C++, Java e incluso Ruby. Ostenta soporte para framework como Swing, Spring y Hibernate.

Esta versión provee un mejor rendimiento, especialmente un arranque mucho más rápido que las versiones anteriores, un menor consumo de memoria y mejor respuesta cuando se trabaja con proyectos grandes. Además, posee un asistente para la generación de clases entidades de una Base de Datos. Soporta todo tipo de llave primaria y todo tipo de relación entre entidades.

NetBeans, cuentan con un plug-in para el framework JUnit que permite la generación de plantillas necesarias para la creación de las pruebas de una clase Java. Esto ofrece al programador enfocarse en la prueba y el resultado esperado, dejando a la herramienta la creación de las clases que coordinan las pruebas.

2.11.8 DEW

El software de réplica de datos DEW se fundamenta en la copia de datos de una localización a otra. Pretende cubrir las principales necesidades relacionadas con la distribución de datos entre los gestores más populares como la protección, recuperación, sincronización de datos, transferencia de datos entre diversas localizaciones o la centralización de la información en una única localización.

Permite además la representación de complejos escenarios de replicación con herramientas para la definición de localizaciones o nodos y la selección de los datos de replicación. Cuenta además con una herramienta Web para la administración y monitoreo de los datos de replicación.

Características

- ✓ Tipos de replicación: Soporta la replicación de transacciones a través de Hibernate, JDBC y la replicación de acciones a través de triggers.
- ✓ Gestores soportados: Se integra actualmente con el gestor de bases de datos Oracle actualmente está en desarrollo el soporte para PostgreSql, MySql, Microsoft SQL Server.
- ✓ Replicación programada: Permite programar el sistema de réplica para enviar los datos en horarios definidos por el usuario y además poder realizar el intercambio de datos en tiempo real.
- ✓ Selección de datos: Provee la facilidad de seleccionar el subconjunto de tablas y datos a ser replicados de la base de datos mediante la definición de filtros aplicables a las tablas y la selección de usuarios propios del gestor.
- ✓ Protocolos de envío de datos: La transferencia de datos de replicación puede realizarse a soporte para replicación sobre TCP/IP, FTP y HTTP. Los archivos de gran tamaño son enviados por FTP permitiendo resumir la trasmisión caso de interrupciones en la red.
- ✓ Configuración entre nodos: El mecanismo de registro entre nodos de replicación se basa únicamente en un identificador (id) del nodo lo que permite la abstracción de los datos físicos de cada nodo como son el IP y el protocolo de comunicación. Este mecanismo permite que los nodos puedan moverse por distintas subredes y mantener sus datos sincronizados. Por ejemplo, mantener sincronizada la base de datos de una laptop con la base de datos central a través de Internet.
- ✓ Seguridad: Los nodos de replicación manejan credenciales entre ellos para verificar la autenticidad de los datos transferidos. El envío de datos se realiza utilizando protocolos de comunicación seguros como son HTTPS y SSL.
- ✓ Monitoreo: Permite el conocer el estado de los datos transmitidos, puede realizarse en tiempo real a través de la Web así como dar un seguimiento al funcionamiento interno del mecanismo.
- ✓ Resolución de conflictos: Los conflictos se detectan y pueden ser solucionados interactivamente o automáticamente mediante la definición de reglas para la resolución de conflictos (este módulo de resolución de conflictos está actualmente en desarrollo).
- ✓ Interfaz visual Web: La administración y configuración de la réplica se realiza a través de una interfaz Web, por lo que es administrable vía remota usando solamente un navegador.

- ✓ Independiente de la plataforma: El software de réplica puede ser instalado en cualquier sistema operativo y no necesita de un ambiente gráfico en el mismo para su funcionamiento.
- ✓ Tolerancia a fallos: Detecta errores de conexión. Mantiene los datos de réplica en un estado estable en caso de desconexión. Al restablecerse la conexión, automáticamente sincroniza los datos entre las bases de datos.

Principales componentes del software de réplica

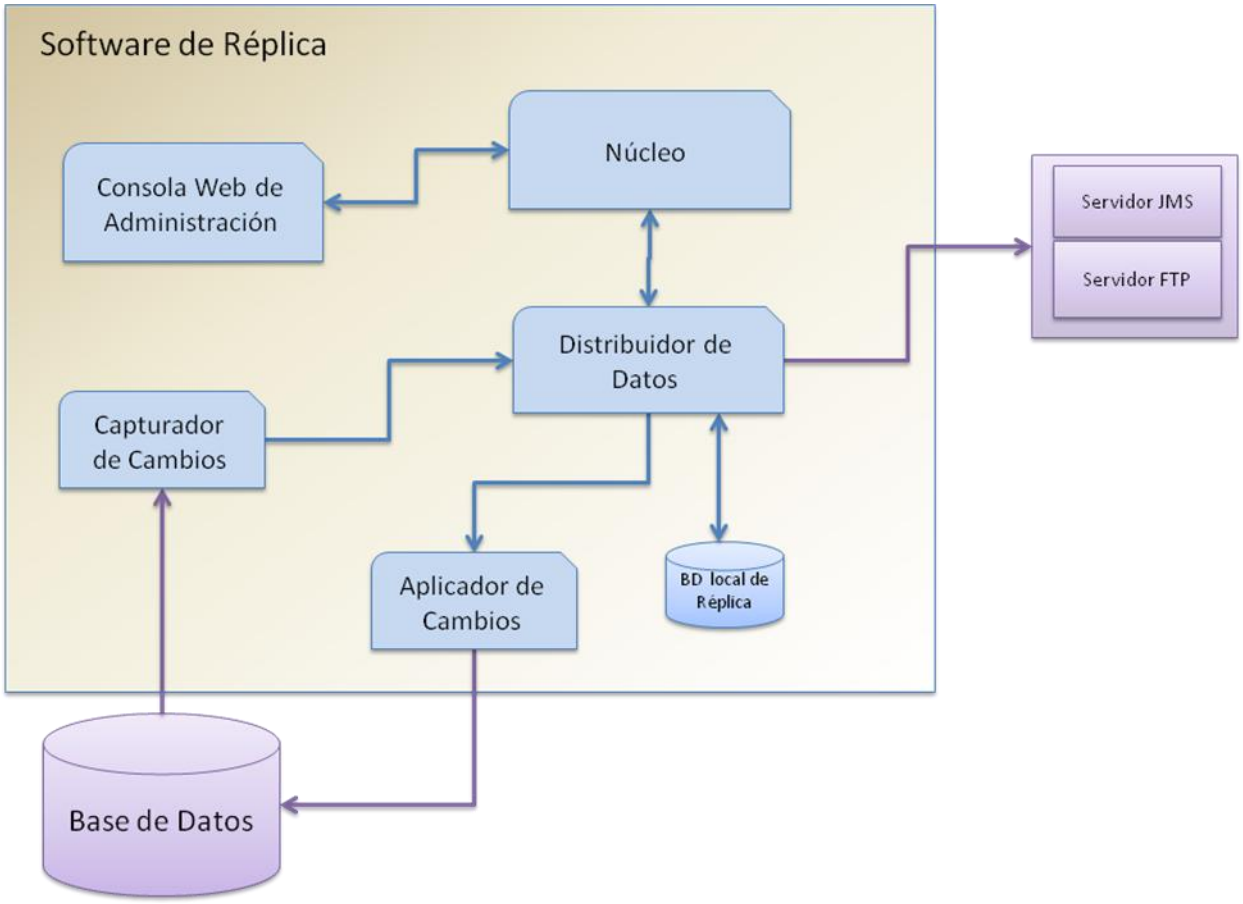


Figura 6: Principales componentes del Software de Réplica.

- ✓ Núcleo: Maneja las configuraciones fundamentales del software y agrupa las principales funcionalidades de procesamiento información.

- ✓ Capturador de Cambios: Captura los cambios que se realizan sobre la BD y se los entrega al Distribuidor de Cambios.
- ✓ Aplicador de Cambios: Ejecuta sobre la BD los cambios que sean replicados hacia la BD.
- ✓ Distribuidor de Cambios: Determina para dónde debe ser enviado cada cambio realizado en la BD, los envía y se responsabiliza de que cada cambio llegue a su destino.
- ✓ BD Local de Réplica: Es utilizada para guardar las configuraciones propias de la réplica, las acciones sobre la BD que han dado conflicto al aplicarse y las acciones o transacciones que no han podido llegar a su destino.
- ✓ Consola Web de Administración: Representa la interfaz del software. Permite realizar las configuraciones principales del software como el registro de nodos, configuración de las tablas a replicar, datos a replicar y el monitoreo del funcionamiento del software.
- ✓ Servidor JMS: Servidor de Mensajería bajo la especificación Java Message Service, es utilizado como punto intermedio en la distribución de la información enviada bajo JMS.
- ✓ Servidor FTP: El servidor FTP es utilizado de forma opcional en el funcionamiento del software. La función principal de utilizar un servidor de FTP es para enviar grandes archivos de réplica y que se pueda resumir la transmisión de los mismos en caso de problemas en la conexión.
- ✓ Base de datos: Es la base de datos que se está replicando, el software de réplica envía los cambios que se realizan sobre ella y aplica los cambios que provienen de otros nodos de réplica.

2.11.9 Máquina Virtual de Java (JVM)

La JVM es un intérprete que convierte el ByteCode compilado de Java en el código de máquina nativo en que debería ejecutarse o sea es un emulador de la ejecución de un código nativo. La tecnología de máquinas virtuales se ha distinguido de la tecnología de interpretación básicamente por el nivel en que se realiza la interpretación. Los intérpretes trasladan un "código de bit" nativo directamente a llamadas del sistema o instrucciones de máquina, pero la emulación consiste en permitir una arquitectura de máquina intermedia.

La arquitectura de la JVM se basa en el concepto de una implementación que no es específica de una máquina. Esto se refiere a que, la arquitectura misma no asume nada acerca de la máquina o de las características físicas y de construcción sobre la cual es implementada. De esta manera, la JVM es una

entidad autónoma y única que ejecuta los archivos de clases. La JVM se separa en cinco unidades de funcionalidad distintas, las que se dedican a la tarea de ejecutar los archivos de clases.

2.12 Conclusiones

En este capítulo se realizó un análisis de las principales características de las tecnologías y herramientas seleccionadas para la construcción y desarrollo del sistema propuesto.

Por lo que se escogieron los patrones de diseño que estarán presente en la solución, así como algunas de las herramientas que servirán para el modelamiento e implementación, garantizando que la selección de estas viabilicen un desarrollo eficaz y efectivo del sistema. Argumentando en cada caso las ventajas que propician para agilizar la construcción del software deseado.

Capítulo 3: Descripción de la Arquitectura

3.1 Introducción

En este capítulo, se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación, tomando como base las descripciones de tecnologías y herramientas del Capítulo 2 de la presente investigación.

La solución se divide en dos sub-tópicos correspondiendo cada uno de ellos a los artefactos fundamentales que define y construye el arquitecto de software. A través de estos artefactos (Línea Base y Documento de Descripción de la Arquitectura) se describe la solución arquitectónica del sistema, incluyendo además otros artefactos definidos por la metodología RUP seleccionada.

3.2 Línea Base de la Arquitectura

La línea base contiene los elementos imprescindibles para lograr la mayor abstracción en el diseño arquitectónico de la aplicación. En ella se exponen los estilos arquitectónicos seleccionados para la aplicación, así como los componentes, conectores y sus configuraciones. Se enumeran los patrones, las restricciones de software y hardware, las tecnologías y herramientas utilizadas en el diseño de la arquitectura.

El propósito de la línea base de la arquitectura es brindar la información necesaria para estructurar el sistema desde el mayor nivel de abstracción.

Los usuarios que interactúan con la línea base de la arquitectura son:

- ✓ El equipo de arquitectos: La utiliza como guía para la toma de decisiones arquitectónicas, y son los encargados de su refinamiento.
- ✓ El equipo de desarrolladores: La utiliza como guía para la implementación.
- ✓ Clientes: Pueden encontrar en ella la garantía de la calidad y el conocimiento de las tecnologías y herramientas seleccionadas.

3.2.1 Concepciones Generales

El sistema se desarrollará utilizando RUP como metodología de desarrollo. Esta metodología define tres puntos fundamentales que guían el diseño de la arquitectura, estos son:

- ✓ Guiado por los casos de uso.
- ✓ Centrado en la arquitectura.
- ✓ Iterativo e incremental.

Estos puntos garantizan que la solución satisfaga las necesidades del cliente, ya que al centrar el desarrollo en los casos de uso permite concebir el sistema en subsistemas y estos a su vez en módulos, teniendo en cuenta para cada uno de ellos el conjunto de operaciones, que dada su prioridad y complejidad son escogidas para el desarrollo de la primera iteración del sistema. Es válido aclarar que con la primera iteración no culmina el desarrollo, en las iteraciones restantes y definidas en el plan de desarrollo del proyecto, se definen y estructuran las restantes necesidades del sistema y se refinan las desarrolladas en iteraciones anteriores.

3.2.2 Organigrama de la Arquitectura

La arquitectura propuesta es para el Sistema de Producción para Empresas Petroleras, el cual se ubica dentro de la categoría de software para la gestión. Este debe poseer una base de datos que le permita el almacenamiento de la información que luego se utilizará para una toma de decisiones. Este tipo de sistema debe poseer el grado máximo de confiabilidad y garantía, pues la principal beneficiada será la economía del país.

En el sistema en cuestión las principales operaciones están enfocadas a gestionar los indicadores de producción (Extracción, Recolección, Tratamiento, Transporte y Venta) de petróleo crudo y gas asociado, a partir de los datos primarios de los pozos, yacimientos e instalaciones de la EPEPO, estas operaciones se catalogan y organizan de acuerdo, principalmente, a las instalaciones identificadas.

Instalación Básica de Medición (IBM): Encargada del control y gestión de las mediciones de los pozos.

Centro Colector (CC): Encargado del control y gestión de las mediciones de los pozos. Contabiliza además, los productos químicos que tiene a su disposición y maneja los parámetros propios del centro.

Batería (Bat): Mantiene un estricto control de la producción que llega desde los centros colectores, además de gestionar las ventas a súper tanqueros y empresas clientes de la EPEPO. Controla también sus parámetros como centro.

Despacho de Producción (DP): Mantiene un estricto control de la producción de crudo y gas asociado de la empresa.

Despacho Central (DC): Encargado de controlar y verificar la información de la producción, así como el cierre diario de la misma. Envía documentos e informes de producción a los directivos de la EPEPO y CUPET que lo requieran, para la toma de decisiones.

Yacimiento (Yac): Encargado de elaborar los planes de producción tanto anual como operativa para la EPEPO.

Configuración (CFG): Gestiona todos los codificadores, los recursos humanos de la EPEPO, así como la seguridad del sistema.

Es extremadamente importante para este sistema que la información se gestione con una alta seguridad y calidad. Las interrupciones en la aplicación deben ser evitadas y la misma debe ser capaz de asimilar cambios en la lógica de los procesos, minimizando el impacto de estos cambios.

A partir de los elementos y necesidades mencionadas anteriormente la arquitectura del sistema se ha organizado a partir del estilo en capas.

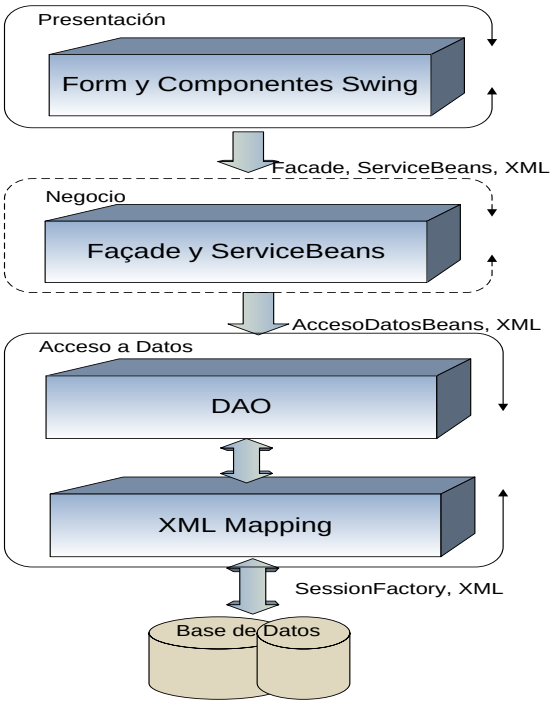


Figura 7: Estilo en Capas

Componentes o elementos

Los principales componentes que distinguen la arquitectura son:

Capa de Presentación

Esta capa contiene las interfaces precisas para que el usuario y el sistema intercambien toda la información necesaria en el proceso de gestión. Está compuesta por formularios y componentes desarrollados a partir del framework Swing. Interactúa con la capa inferior, mediante invocación de los métodos que conforman la lógica del negocio, produciendo un intercambio de objetos. Así, en esta capa se resuelven cuestiones como:

- ✓ Navegabilidad del sistema.
- ✓ Formateo de los datos de salida: resolución del formato más adecuado para la presentación de resultados.

- ✓ Validación de los datos de entrada, en cuanto a formatos, longitudes máximas, etc.
- ✓ Interfaz gráfica con el usuario.

Capa de Negocio

Esta capa almacena la lógica del negocio. En ella se deben implementar todas aquellas reglas obtenidas a partir del análisis funcional del proyecto. Así mismo, debe ser completamente independiente de cualquiera de los aspectos relacionados con la presentación de la aplicación. De esta forma, permite una reutilización de código al poder utilizar la misma capa de negocio para una aplicación web común, o una standalone.

Por otro lado, la capa de negocio ha de ser también completamente independiente de los mecanismos de persistencia empleados en la capa de acceso a datos. Cuando la capa de negocio requiera recuperar o persistir entidades o cualquier conjunto de información, lo hará siempre apoyándose en los servicios que ofrezca la capa de acceso a datos para ello. De esta forma, la sustitución del motor de persistencia no afecta en lo más mínimo a esta parte del sistema.

Capa de Acceso a Datos

La capa de acceso a datos es la responsable de la gestión y persistencia de la información manejada en las capas superiores. Esta capa contiene la funcionalidad de acceder al repositorio de los datos, se encarga de ejecutar la lógica de acceso a los datos y posee tres subcapas fundamentales:

1. XML Mapping (Ficheros de Mapeo): Contienen la información de la base de datos con la cual trabajará el sistema, así como, los campos de cada una de las tablas y sus relaciones.
2. Objetos de negocio: Se generan a partir del mapeo de las clases de las tablas de la base de datos son clases entidades de Java que contiene los atributos y métodos para acceder a ellos, estos constituyen los objetos con los que se trabaja en la aplicación.
3. DAO (Objetos de Acceso a Datos): Son las clases que contienen la funcionalidad necesaria para acceder a la base de datos y realizar un conjunto de operaciones definidas para poder realizar las transacciones.

Conectores / Configuraciones

Los conectores, son las formas de comunicación entre los componentes o elementos definidos; es la forma en que está representada la información que fluye entre estos.

Presentación – Negocio, ServiceBeans

Utilizando el esquema propuesto para los XML por el framework Spring, se crean los servicios en la capa de Negocio los cuales son referenciados por las vistas en la capa de Presentación.

La referencia a estos Beans puede hacerse mediante el constructor o mediante una propiedad. De esta forma las vistas pueden acceder a las funcionalidades que les brindan los servicios referenciados.

```
<bean id="applicationFacade" class="app.comun.facade.impl.ApplicationFacadeImpl">
    <property name="applicationService" ref="applicationService" />
</bean>
```

En este caso se hace referencia la bean applicationFacade, lo que quiere decir que, al invocar al constructor de la vista se está invocando al bean del servicio quedando conectadas las dos capas.

Negocio – Acceso a Datos, AccesoDatosBeans

Los DAO que se crean en la capa de Acceso a Datos son referenciados por los bean que se crean en la capa de Negocio.

```
<bean id="applicationService" class="app.comun.service.impl.ApplicationServiceImpl">
    <property name="applicationDao" ref="applicationDao" />
</bean>
```

En este caso se le referencia al bean applicationService, lo que quiere decir que al invocar al constructor de la clase del Servicio se está invocando al bean del DAO. De esta forma quedan conectadas las dos capas.

Acceso a Datos – Base Datos, SessionFactory

El SessionFactory es un bean del framework Hibernate que proporciona la conexión al repositorio de datos.

```
<hibernate-configuration>
  <session-factory>
    <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
    <property name="hibernate.dialect">org.hibernate.dialect.PostgreSQLDialect</property>
    <property name="hibernate.connection.driver_class">org.postgresql.Driver</property>
    <property name="transaction.factory_class">org.hibernate.transaction.JDBCTransactionFactory</property>
    <property name="hibernate.cache.provider_class">org.hibernate.cache.HashtableCacheProvider</property>
    <property name="hibernate.hbm2ddl.auto">update</property>
    <property name="hibernate.connection.url">jdbc:postgresql://localhost:5432/dbtest</property>
    <property name="hibernate.connection.username">user</property>
    <property name="hibernate.connection.password">pass</property>
  </session-factory>
```

El SessionFactory es la fábrica de las sesiones (conexiones) que se crean en la base de datos, contiene los datos necesarios para realizar la conexión los que se le pasan como propiedad al bean, por ejemplo: el driver que identifica a qué tipo de base de datos se va a conectar, el usuario y el password para conectarse y el nombre del servidor.

Restricciones

La principal restricción que se les impone a los componentes, es que los ubicados en las capas superiores, solo pueden hacer uso de los que están ubicados en capas inferiores o verticales, que se extiendan a su nivel.

3.2.3 Framework de desarrollo

Otro de los niveles de abstracción que se ha visto en el capítulo anterior fueron los Framework. Estos no son más que arquitecturas definidas para un determinado dominio de la aplicación que contiene un

conjunto de componentes implementados y sus interfaces bien definidas. Dichos componentes se pueden utilizar, redefinir y crear.

A partir de la organización de la arquitectura del estilo en Capas definido para el sistema, se propone la utilización de tres framework distribuidos en las tres capas del sistema:

Capa de Presentación: Framework Swing.

Capa de Negocio: Framework Spring.

Capa de Acceso a Datos: Framework Hibernate.

Sus características se explican detalladamente en el capítulo anterior.

3.2.4 Patrón de arquitectura

El patrón de arquitectura escogido para el desarrollo del sistema es Arquitectura en Capas. Sus características, ventajas, desventajas y fundamentación se encuentran explicados en el capítulo anterior.

3.2.5 Lenguaje, tecnologías y herramientas de apoyo al desarrollo

Para lograr un menor tiempo de desarrollo y minimizar los gastos en licencia, garantizando la escalabilidad y robustez del sistema, así como para conseguir un producto fiable que se adapte a las necesidades de la EPEPO, además de permitir a los clientes finales un entorno de trabajo amigable y flexible se proponen las siguientes herramientas y tecnologías a utilizar, usando como base lo expuesto en el capítulo anterior:

- ✓ Gestor de base datos: PostgreSQL v8.2.
- ✓ Lenguaje de Programación: Java con la utilización de los framework Swing, Spring v2.0 e Hibernate v3.2.
- ✓ Metodología de desarrollo de software: Rational Unified Proccess con notación UML.
- ✓ Herramienta CASE de modelado UML: Visual Paradigm v3.4.
- ✓ IDE´s de desarrollo: NetBean v6.5.
- ✓ Herramienta de control de versiones: SubVersion y como cliente Tortoise.

- ✓ Herramienta de generación de Capa de Accesos a Datos: Hibernate Tool.
- ✓ Herramienta de generación de reporte: JasporReport.

3.3 Descripción 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 realiza la descripción de la Arquitectura.

Para esto se proporciona una comprensión arquitectónica global del sistema, utilizando las vistas arquitectónicas definidas por la metodología RUP, que muestran las diferentes características del sistema. Esta descripción debe capacitar a los desarrolladores, directivos, clientes y otros usuarios en la compresión al detalle del sistema propuesto, facilitando su participación en el mismo.

La descripción de la arquitectura influye en la toma de decisiones arquitectónicas correspondientes al Sistema de Producción para Empresas Petroleras y abarca todos los subsistemas del software.

3.3.1 Representación Arquitectónica

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

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

Estas vistas serán representadas en UML utilizando Visual Paradigm como herramienta CASE de modelado.

3.3.2 Objetivos y restricciones de la arquitectura

Las metas y restricciones que definen características de importancia para la arquitectura serán descritas a continuación:

Requerimientos de Hardware

Estaciones de Trabajo

- ✓ Periféricos Mouse y Teclado PS 2 o USB.
- ✓ Tarjeta de red.
- ✓ 1Mb de caché L2, 256 Mb de memoria RAM.
- ✓ 5 GB de espacio libre en disco.
- ✓ CPU Pentium II a 800 MHz o superior.
- ✓ Impresora.

Servidor de Base de Datos

- ✓ Periféricos: Mouse y Teclado PS 2 o USB.
- ✓ Tarjeta de Red.
- ✓ 1GB MB de RAM.
- ✓ Capacidad de Almacenamiento 20 GB.
- ✓ Arquitectura hardware Intel.
- ✓ Sistema operativo Windows 2000 o superior, GNU-Linux (recomendado), Unix.

Requerimientos de Software

Estaciones de Trabajo

- ✓ Sistema Operativo: Windows 2000 o superior, GNU-Linux, Unix.
- ✓ Java Runtime Environment (Máquina Virtual de Java) versión 1.6.

Servidor de Base de Datos

- ✓ Sistema Operativo: Windows 98 o superior, GNU-Linux (recomendado), Unix.

- ✓ Gestor de Base de Datos: PostgreSQL. v8.2.

Redes

- ✓ La red existente en las instalaciones debe soportar la transacción de paquetes de información de al menos unas 10 máquinas a razón de 2 ó 3 Mb/s.
- ✓ Para hacer más fiable la aplicación la misma debe estar protegida contra fallos de corriente y conectividad, para lo que se deberá parametrizar los tiempos para realizar copias de seguridad. Las réplicas se realizará utilizando la herramienta DEW donde se especificará que: primero replicarán los Centros Colectores, luego las Baterías y por último el Despacho de Producción; el Despacho Central trabajará directamente con la Base de Datos central. Lo que se realizará teniendo en cuenta el tiempo de mejor tráfico en la red, o sea, a partir de las 12 am.

Seguridad

- ✓ La seguridad se tratará desde el modelamiento del sistema, pasando por las primeras fases de desarrollo del sistema y profundizando en las finales.
- ✓ Se utilizarán las reglas de la “programación segura”, se deberá hacer un fuerte tratamiento de excepciones desde la capa de Presentación hasta la capa de Acceso a Datos incluyendo el trabajo con la Base de Datos. Para reforzar este aspecto los desarrolladores se apoyarán en los framework seleccionados.
- ✓ Los usuarios de la BD solamente tendrán acceso a las tablas de la Base de Datos a las cuales se les designe según el rol que desempeña, no se utilizarán usuarios con privilegios administrativos para realizar las conexiones entre servidores.
- ✓ Se registrarán y auditarán las trazas dejadas por los usuarios al realizar las diferentes operaciones en el sistema. La programación de las auditorías será según corresponda.
- ✓ La asignación de usuarios y sus opciones sobre el sistema se garantizará desde el subsistema de Administración.

Portabilidad, escalabilidad y reusabilidad

- ✓ El sistema deberá ser utilizado desde cualquier plataforma de software (Sistema Operativo).

- ✓ El sistema permitirá hacer un uso racional de los recursos de hardware de la máquina, sobre todo en las PC clientes.
- ✓ La aplicación se construirá utilizando estándares internacionales y patrones, para facilitar su integración futura con componentes desarrollados por cualquier empresa y garantizar posibilidades de un mantenimiento ágil.
- ✓ De acuerdo a las condiciones económicas del país, el sistema deberá minimizar la cantidad de gastos de despliegue y contar con la capacidad de ajustarse a las necesidades de la EPEPO.
- ✓ Se desarrollará cada pieza del sistema en forma de componentes (subsistemas) con el fin de re-utilizarlos para futuras versiones.
- ✓ La documentación de la arquitectura deberá ser reutilizable para poder documentarla como una familia de productos.

Restricciones de acuerdo a la estrategia de diseño

- ✓ El diseño de las aplicaciones se hará utilizando la Programación Orientada a Objetos (POO).
- ✓ Se utilizarán las tecnologías que brindan los framework definidos para cada una de las capas de la aplicación:
    - o Para la capa de presentación: framework Swing.
    - o Para la capa de lógica del negocio: framework Spring.
    - o Para la capa de Acceso a Datos: framework Hibernate.

Herramientas de desarrollo

- ✓ Para modelado se utilizará Visual Paradigm, el cual demanda como mínimo 512 MB de RAM y recomendado 1 GB de RAM.
- ✓ Para implementación se utilizará como IDE de desarrollo NetBean el cual demandan como mínimo 768 MB de RAM y recomendado 1.0 GB de RAM.
- ✓ Para implementación los plug-in serán:
    - o Hibernate Tool el cual facilita la ingeniería inversa y reversa de los ficheros de mapeo entre clases Java y tablas de la base de datos.
    - o JUnit el cual facilita las pruebas de unidad.
    - o SVNeclipse el cual facilita la integración con Subversión.

- SVNNetBean el cual facilita la integración con Subversión.
- JasperReport el cual permite la generación automática de reportes.

- ✓ Como servidor de Base Datos PostgreSQL (256 MB de RAM).
- ✓ Como servidor de control de versiones Subversión.
- ✓ Como herramienta para la replicación se utilizará DEW.

3.3.3 Estructura del equipo de Desarrollo

El equipo de trabajo está distribuido de la siguiente forma: Ver anexo no: 2.

- ✓ Jefe de proyecto.
- ✓ Líder de desarrollo.
- ✓ Arquitecto Principal.
- ✓ Arquitecto de Información.
- ✓ Analista Principal.
- ✓ Implementador e Integrador.
- ✓ Planificador.
- ✓ Diseñador de Base de Datos.
- ✓ Diseñador de Interfaz de Usuario.
- ✓ Administrador de Configuración y Control de Cambios.
- ✓ Grupo de Calidad.
- ✓ Grupo de Investigación.

El software se compone por subsistemas, y estos a su vez por módulos. Los subsistemas no son totalmente independientes entre sí, y para el desarrollo de la aplicación se tienen concebidos los siguientes:

- ✓ Subsistema Centro Colector (CC).
- ✓ Subsistema Batería (Bat).
- ✓ Subsistema Despacho de Producción (DP).
- ✓ Subsistema Despacho Central (DC).

- ✓ Subsistema Yacimiento (Yac).
- ✓ Subsistema Configuración y Seguridad (CFG).

Configuración de los puestos de trabajos por roles

Analista

- ✓ PC, con mouse y teclado.
- ✓ Instalación de Visual Paradigm.
- ✓ Instalación del paquete Office/ Open Office.

Implementador

- ✓ PC, con mouse y teclado.
- ✓ Instalación del IDE NetBean y los plug-in necesarios (JUnit, Hibernate-Tool, JasperReport).
- ✓ Instalación de la máquina Virtual JDK v1.6.

Documentador

- ✓ PC, con mouse y teclado.
- ✓ Instalación de Visual Paradigm.
- ✓ Instalación del paquete Office/ Open Office.

Diseñador BD

- ✓ PC, con mouse y teclado.
- ✓ Instalación de Visual Paradigm.
- ✓ Instalación de PostgreSQL.

3.3.4 Vista de Casos de Uso

La vista de casos de uso presenta los actores y casos de uso (o escenarios de esos casos de uso) más importantes, es decir, representa los casos de uso arquitectónicamente significativos. Estos son aquellos que describen alguna funcionalidad importante y crítica, o que impliquen algún requisito importante que

deba desarrollarse pronto dentro del ciclo de vida del software. Esta vista se utiliza como entrada al hacer la planificación de lo que deba desarrollarse dentro de una iteración.

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.

Subsistema de Administración y Configuración.

Estos casos de uso son prioritarios en el sistema, ya que brindan funcionalidades necesarias para la puesta en marcha de la aplicación, además de que constituyen la base para la seguridad de la aplicación. A partir de ellos se puede gestionar la traza de las operaciones realizadas en el sistema, los codificadores que se utilizan en varios o todos los módulos del sistema, así como los datos del personal y la producción, de los cuales dependen varias operaciones fundamentales para el sistema.

Gestionar Usuario: Permite al administrador del sistema gestionar los usuarios que luego interactuarán con el sistema.

Autenticar Usuario: Este caso de uso permite la autenticación de los usuarios que acceden al sistema, controlando de esta manera la seguridad del mismo.

Gestionar Rol: Permite la gestión de roles para los diferentes usuarios.

Gestionar Permiso: Permite la gestión de los permisos de los usuarios, para controlar de esta manera el nivel de acceso a las diferentes partes del sistema.

Gestionar Codificadores: Permite gestionar los codificadores del sistema los cuales se utilizarán en varios o todos los módulos de la aplicación.

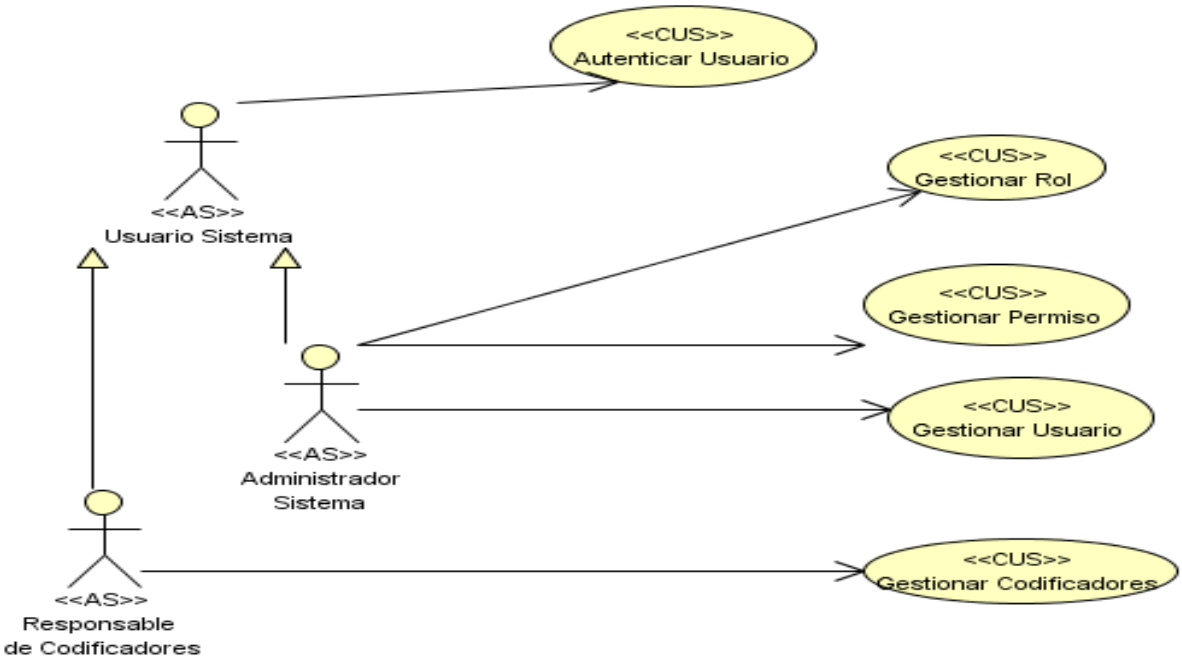


Figura 8: Subsistema Administración y Configuración

Subsistema Centro Colector

Gestionar Parámetros de Pozos de CC: Este caso de uso (CU) tiene como principal objetivo gestionar (insertar, modificar y eliminar) los parámetros de los pozos asociados al Centro Colector.

Gestionar Medición: este CU Se inicia cuando el responsable del CC quiere realizar una inserción, modificación o eliminación de una medición hecha a un Pozo asociado al Centro Colector.

Gestionar Trasiego: Permite al responsable del CC la inserción, modificación y eliminación de los Trasiegos realizados por las pailas o camiones cisternas hacia algunas de las instalaciones de la empresa para el procesamiento del crudo.

Generar Parte de Producción de CC: El objetivo fundamental de este CU es confeccionar el parte de la producción del CC teniendo en cuenta la producción de cada uno de los pozos en el día.

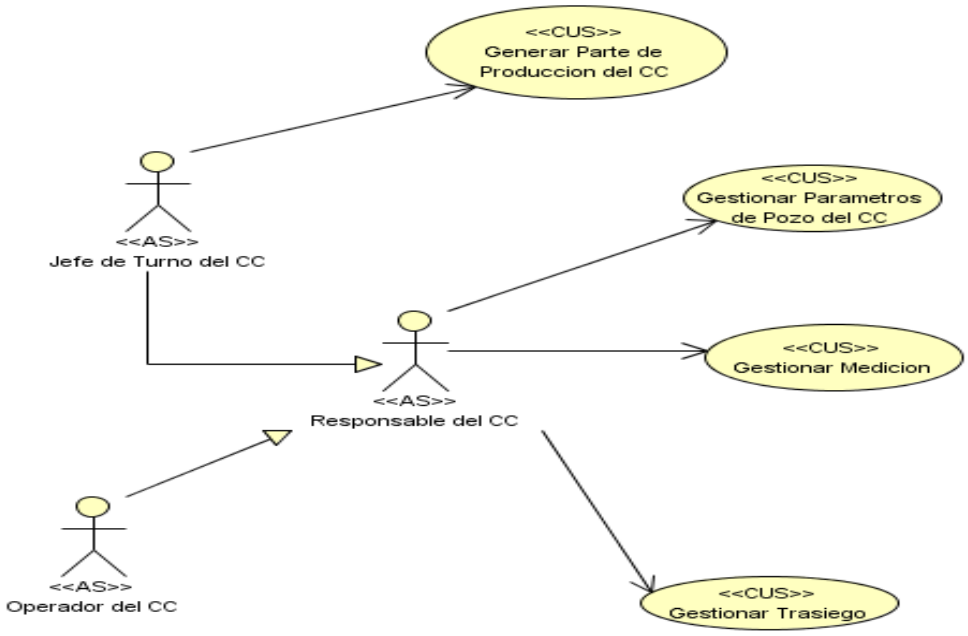


Figura 9: Subsistema Centro Colector

Subsistema de Batería

Gestionar Ventas: Permite insertar, eliminar y modificar las ventas realizadas a los diversos clientes de la EPEPO.

Gestionar Parámetros de Batería: Permite gestionar los parámetros propios de esta instalación.

Gestionar Control de Parámetro de Entrega de Batería: Gestiona todos los parámetros referentes al crudo recibido de los CC e Instalaciones Básicas de Medición (IBM).

Generar parte de Producción de Batería: Calcula y genera la producción diaria de la Batería teniendo en cuenta las ventas, las entregas realizadas, los consumos, etc.

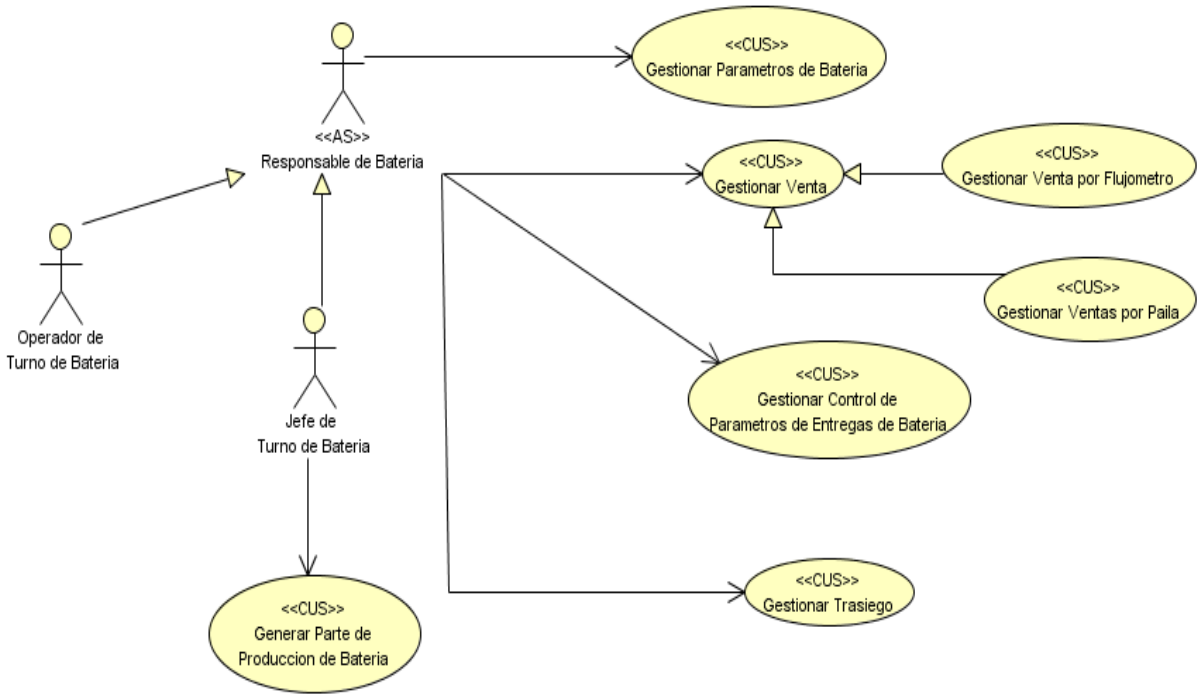


Figura 10: Subsistema Batería

Los casos de uso anteriormente explicados encierran las funcionalidades correspondientes a las operaciones básicas que debe realizar el sistema, a partir de aquí se pueden extender las demás funcionalidades.

3.3.5 Vista Lógica

La vista lógica apoya principalmente los requisitos funcionales, lo que el sistema debe brindar en términos de servicios a sus usuarios. Se describe y divide en paquetes del sistema de una forma abstracta, y las relaciones que existen entre ellos, ya sean de dependencia o uso. 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, 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 aplicación.

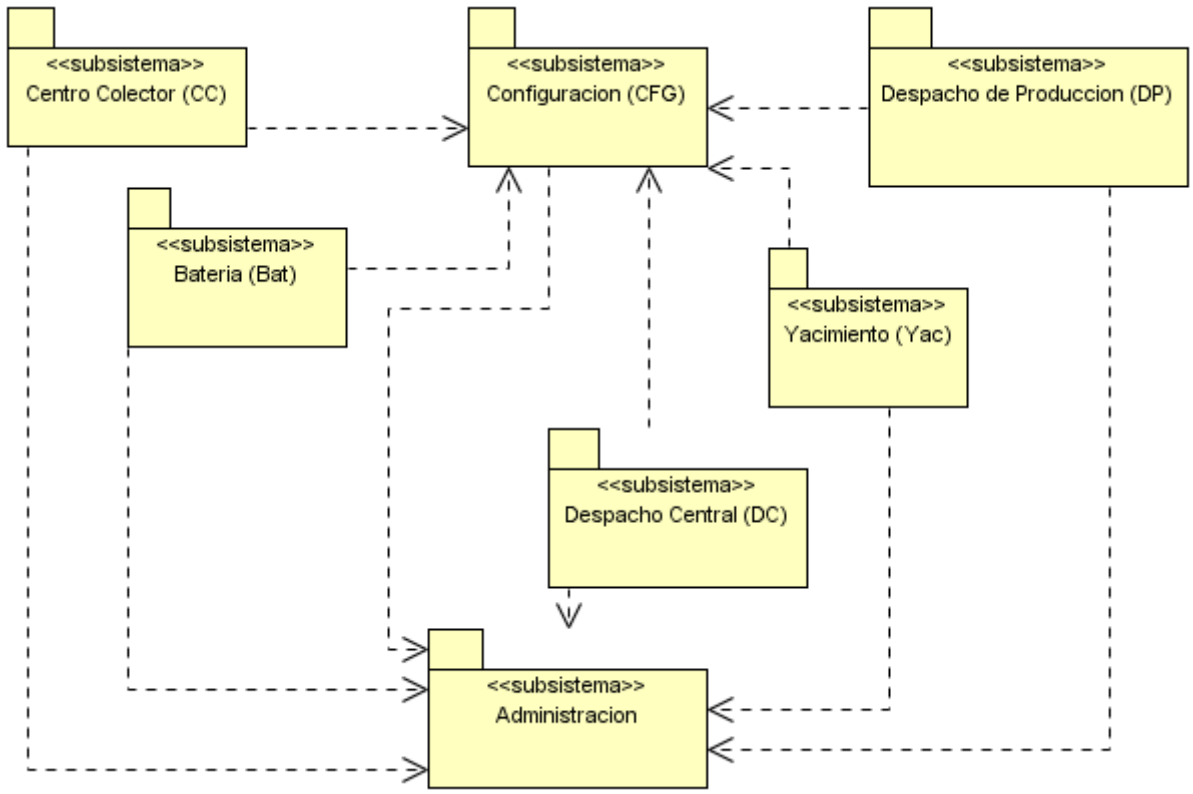


Figura 11: División del Sistema en Subsistemas

Los subsistemas principales identificados son:

- ✓ Administración: Contiene la realización de los Casos de Uso (CU) correspondiente con la gestión de usuarios, roles y permisos sobre el sistema.
- ✓ Configuración: Contiene la realización de los CU correspondientes con la gestión de codificadores y configuración del sistema.

- ✓ Centro Colector (CC): Contiene la realización de los CU correspondientes a la gestión de parámetros de pozos, sus mediciones, su producción y el cálculo de la producción en general del CC.
- ✓ Batería (Bat): Contiene la realización de los CU correspondientes al control de la producción proveniente de los CC asociados, así como, las ventas realizadas a los clientes de la EPEPO y los parámetros propios de la Batería.
- ✓ Despacho de Producción (DP): Contiene la realización de los CU correspondientes al control de la producción diaria de la EPEPO y la gestión de las afectaciones.
- ✓ Despacho Central (DC): Contiene la realización de los CU correspondientes al control de la producción diaria, cierre operacional y anual de la producción de crudo y gas de la EPEPO.
- ✓ Yacimiento (Yac): Contiene la realización de los CU correspondientes a la confección de los planes operativos y anuales de la empresa.

La distribución de la aplicación para cada uno de los subsistemas se realizará de la siguiente forma:

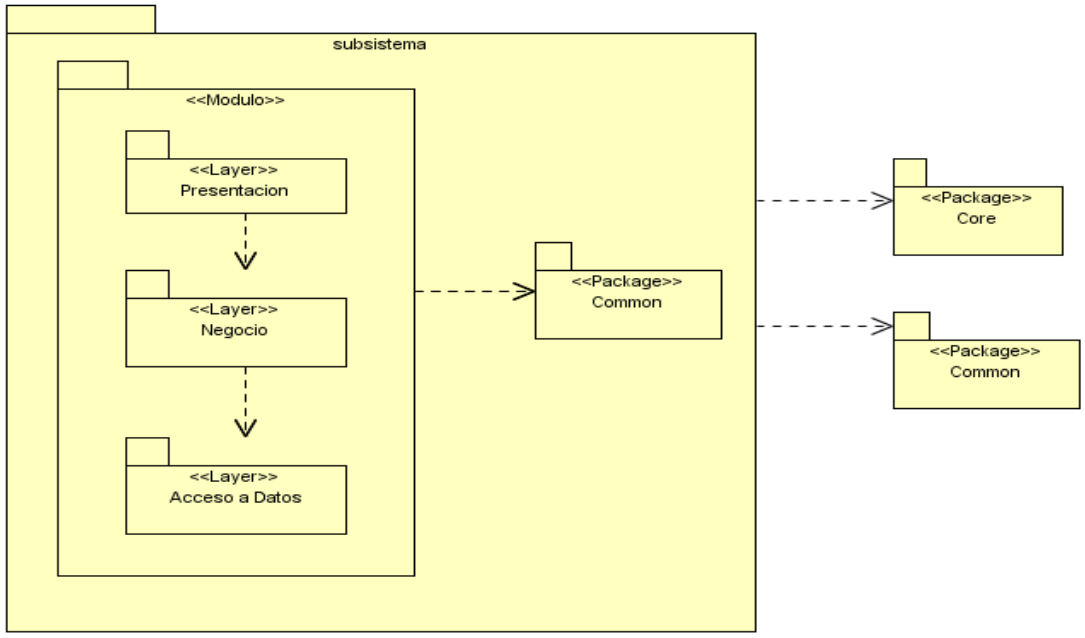


Figura 12: Visión General de la Arquitectura

- ✓ Presentación: Contiene las clases, formularios y componentes del framework Swing que componen el sistema.
- ✓ Negocio: Contiene las clases que controlan la Lógica del Negocio y componentes del framework Spring.
- ✓ Acceso a Datos: Clases que controlan la lógica transaccional y la interacción con la Base de Datos, componentes del framework Hibernate.
- ✓ Common: Paquete que encierra clases e interfaces comunes de la configuración de los módulos. Permite eliminar dependencias directas entre los módulos específicos de la aplicación y se extiende también para establecer este mecanismo de colaboración a nivel de subsistemas.
- ✓ Core: Contiene los paquetes y clases de configuración del sistema, presenta un mecanismo genérico de acceso a datos.

A continuación se detallan cada uno de los paquetes mencionados anteriormente:

El paquete Core:

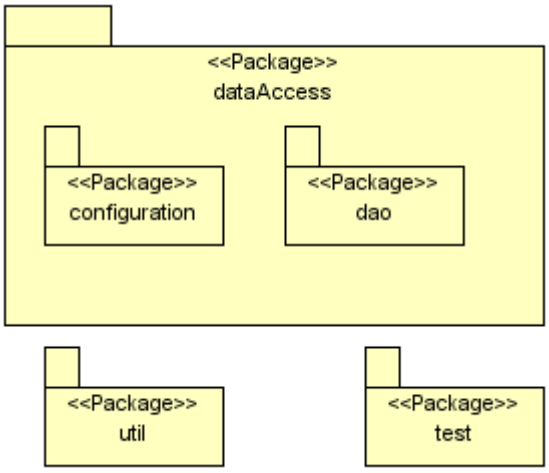


Figura 13: Paquete Core

- ✓ dataAccess: Es el paquete que agrupa los paquetes relacionados con el mecanismo genérico de acceso a datos, los cuales son:

- configuration: En este paquete están contenido los AppContext que instancian todos los objetos necesarios para proveer la conexión a la base de datos, así como el pool de conexiones y el administrador de las transacciones (Transaction Manager).
- dao: En la raíz de este paquete se encuentra la interfaz genérica de acceso a datos BaseDao. Esta interfaz define las operaciones básicas (Crear, Leer, Actualizar y Eliminar) de todos los objetos de acceso a datos. Estas operaciones serán compartidas por todos los objetos de acceso a datos de la aplicación.

✓ test: Contiene las clases responsables de las pruebas de unidad haciendo uso de JUnit.

✓ util: Contiene un conjunto de paquetes y clases de utilidades para el correcto funcionamiento del sistema.

El paquete Common contiene la misma estructura básica de los módulos, su función fundamental es eliminar dependencias directas entre los subsistemas y módulos específicos de la aplicación, así como moverlas hacia un paquete común para lograr una mayor independencia. Su estructura es la siguiente:

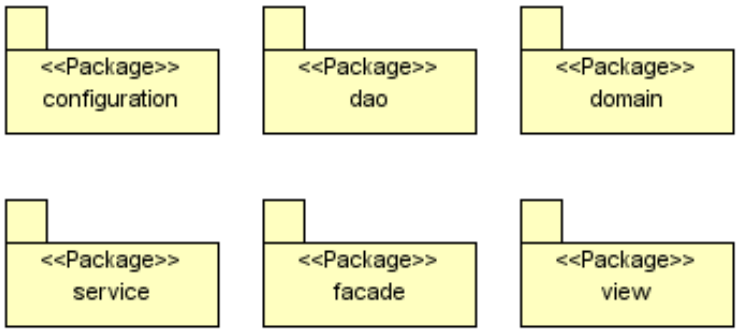


Figura 14: Paquete Common

✓ configuration: Se localizarán todos los archivos que tienen que ver con la configuración general del módulo o subsistema según sea el caso, los “Application-Context” de Spring Framework, los ficheros de propiedades “.properties” y cualquier otro destinado a estos fines.

✓ dao: Se encontrarán la declaración y la implementación de las interfaces DAO.

✓ domain: Se encontrarán todas las entidades generales persistentes, pertenecientes al subsistema o sistema en general.

- ✓ facade: Se encontrarán las declaraciones e implantaciones de las interfaces de las fachadas generales del negocio.
- ✓ service: Contendrá la declaración y la implementación de la interfaz BaseService.
- ✓ view: Se encontrarán las clases utilizadas para validar datos de la capa de presentación.

La estructura de la capa de Presentación es la siguiente:

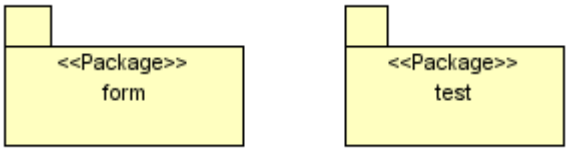


Figura 15: Capa de Presentación

- ✓ Paquete form: Contendrá las clases de interfaz de usuario que permitirán interactuar con la aplicación.
- ✓ Paquete test: Se encontrarán las clases para la realización de las pruebas de unidad.

La estructura de la capa de Lógica de Negocio es la siguiente:

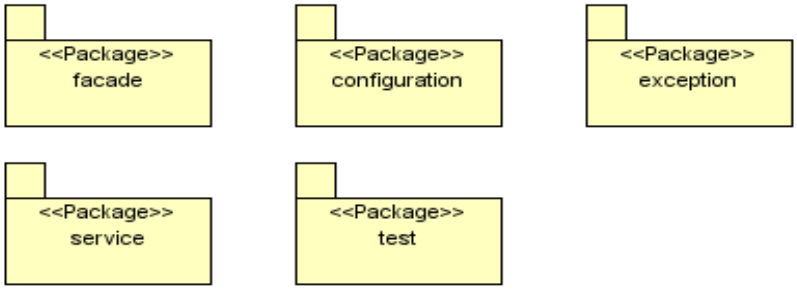


Figura 16: Capa Lógica de Negocio

- ✓ Paquete facade: Se localizarán las interfaces e implementaciones de los servicios de la fachada del negocio.
- ✓ Paquete configuration: Se encontrarán los ficheros XML para la configuración de los Beans de cada uno de los servicios referente a cada módulo en específico.

- ✓ Paquete test: Contendrán las pruebas de unidad (JUnit) para cada uno de los servicios implementados.
- ✓ Paquete exception: Se encontrarán las clases para el manejo de las excepciones.
- ✓ Paquete service: Se encontrarán las interfaces que representan las operaciones de negocio que se van a exponer o consumir como servicio dentro de la aplicación.

La estructura de las capas de Acceso a Datos es la siguiente:

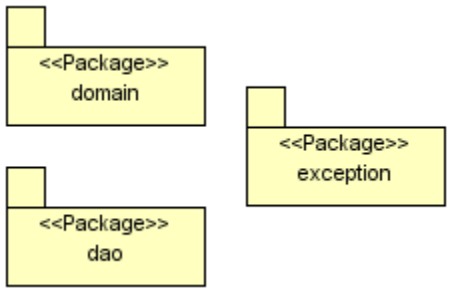


Figura 17: Capa de Acceso a Datos

- ✓ Paquete dao: Se localizarán las implementaciones de las interfaces DAOs, ficheros de mapeo utilizados por hibernate y las clases pruebas utilizadas para realizar pruebas de unidad a las implementaciones de los DAO.
- ✓ Paquete domain: Se encontrarán todas las entidades persistentes pertenecientes al módulo.
- ✓ Paquete exception: Se localizarán las clases para el manejo de las excepciones.

3.3.6 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 concurrentes o hilos.

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

3.3.7 Vista de Implementación

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.

Los paquetes definidos para cada uno de los subsistemas y módulos de la aplicación son:

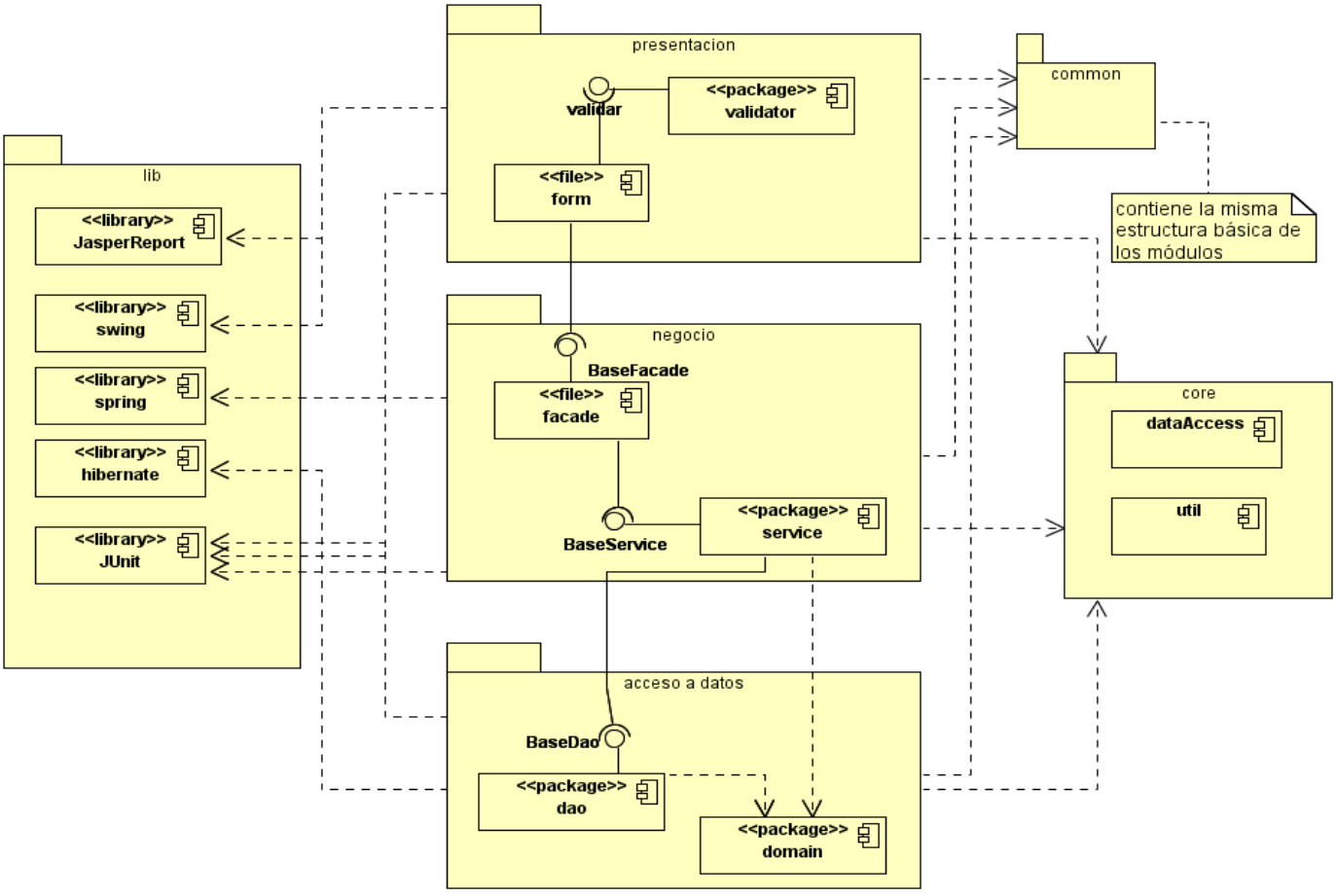


Figura 18: Vista de Implementación

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

- ✓ core: Contiene los componentes fundamentales para el trabajo en cada una de las capas, así como los componentes de acceso a datos.
- ✓ common: Contiene la misma estructura básica de los módulos y encierra clases e interfaces comunes.
- ✓ presentación: Contiene todos los formularios, los cuales están constituidos por los componentes que brinda el framework swing, así como las clases de validación y las que controlan los eventos lanzados en la vista. Permite visualizar los reportes mediante el generador de reporte JasperReport en cualquiera de los formatos que este soporta.
- ✓ negocio: Contiene todas las interfaces y servicios requeridos para la densa lógica de negocio identificada.
- ✓ acceso a datos: Contiene las clases u objetos de acceso a datos de la aplicación, ficheros de mapeo generados por hibernate y conjunto de xml con las consultas a la BD en HQL y clases persistentes.

3.3.8 Vista de Despliegue

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

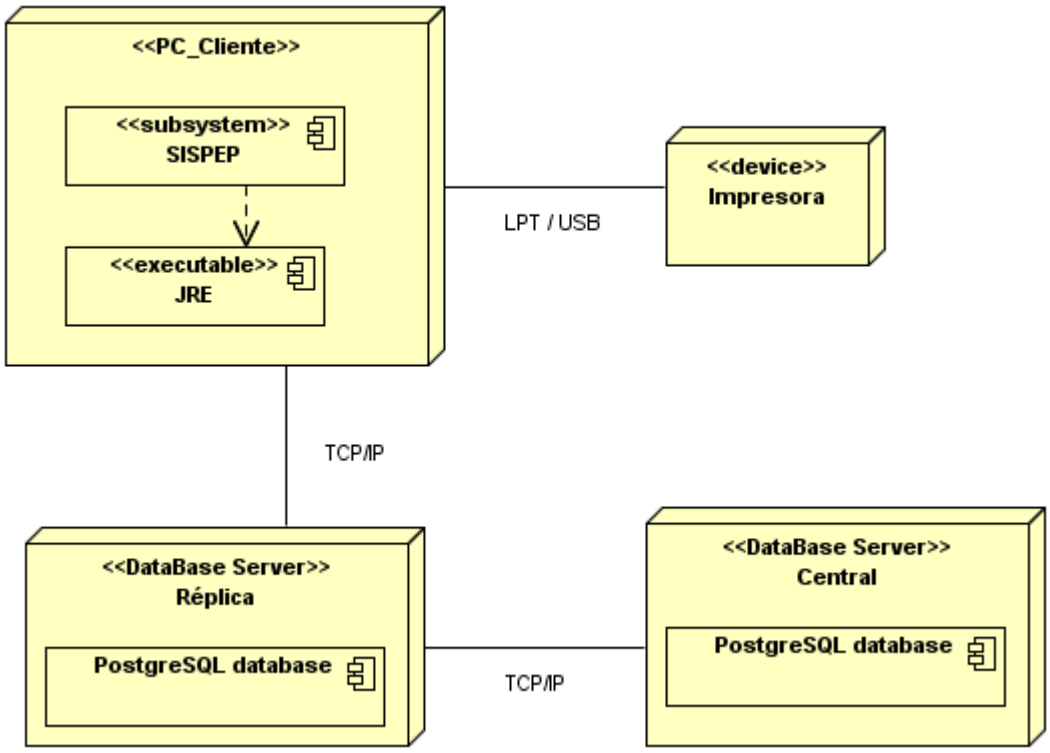


Figura 19: Vista de Despliegue

Nodo DataBase Server Réplica: Constituye un nodo servidor de base de datos el cual contendrá el Gestor de base de datos PostgreSQL v 8.2. Es importante tener presente que este servidor de BD será una réplica, ya que dada la distribución de la aplicación y la estructura de redes con que cuenta la empresa, se necesita realizar réplicas de la BD central para un mejor funcionamiento del software.

Nodo DataBase Server Central: Constituye un nodo servidor de base de datos central en el cual se encuentra la información centralizada referente al control de la producción. También con el SGBD PostgreSQL v 8.2.

Nodo PC_Cliente: Contiene el ejecutable de la aplicación (.jar) y la Java Runtime Environment (JRE o Máquina Virtual de Java). La utilización de los recursos de la PC Cliente viene dado por los subsistemas de la aplicación que el cliente necesite trabajar, así se evita sobrecargar las máquinas con elementos de la aplicación que no serán usados.

Dispositivo Impresora: Satisfacen los requisitos de los clientes de impresión de reportes generados por la aplicación. La forma de conectarse al dispositivo puede ser 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.

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.

3.4 Conclusiones

En este capítulo, se presentan dos documentos de obligada consulta y constante refinamiento por parte del arquitecto de software, que conforman la descripción arquitectónica del Sistema de Producción para Empresas Petroleras (SisPEP). Se exponen y fundamentan los patrones utilizados, así como las tecnologías y herramientas. Se describe la arquitectura mediante las vistas propuestas por la metodología RUP. Se justifica un desarrollo ágil y de calidad mediante el uso de los framework propuestos.

Conclusiones Generales

El valor fundamental de este sistema se expresa en la contribución a simplificar el trabajo y la demora que produce el procesamiento manual de la información y mejorar la gestión de las actividades que se realizan en la Empresa de Perforación y Extracción de Petróleo de Occidente (EPEPO).

Se realizó un diagnóstico de los procesos de gestión de información en la EPEPO, del cual se pudo concluir que los procesos que se llevan a cabo en la misma son lentos y poco eficientes, debido al trabajo manual que debe realizar el personal involucrado con el elevado cúmulo de información que engloba el proceso.

Además se arribaron a las siguientes conclusiones:

- ✓ Con el estudio detallado de los principales aspectos de la descripción arquitectónica se logró seleccionar el estilo arquitectónico adecuado para la solución, así como sus componentes, configuraciones y restricciones.

- ✓ Con el estudio detallado de las principales categorías de patrones (arquitectónicos, diseño e idioma) se logró fundamentar la elección de algunos para su aplicación, añadiéndole al sistema calidad, claridad y sencillez en la estructura y diseño de la solución, además de facilidades de lectura y mantenimiento de código.

- ✓ Con las restricciones impuestas por algunos requisitos no funcionales del cliente y en función de los principales atributos de calidad de la arquitectura de software se logró definir las principales tecnologías y herramientas a utilizar en el desarrollo del sistema y se configuró cada uno de los puestos de trabajo por roles en el equipo de desarrollo.

Por lo que se considera que la propuesta cumple con los requisitos que se necesitan como solución al problema planteado, que es completamente desarrollable por parte de los implementadores y puede ser aprobada y administrada por los arquitectos y jefe de proyecto.

Recomendaciones

Se recomienda a líderes y arquitectos del polo Petrosoft:

- ✓ Efectuar un continuo refinamiento de la arquitectura durante cada ciclo de desarrollo.
- ✓ Realizar evaluación de costo de la tecnología y requerimientos de hardware e incentivar la búsqueda de la reducción de estos hacia las necesidades de los clientes.
- ✓ El estudio de cualquier método de evaluación de la arquitectura existente, con el objetivo de identificar las principales debilidades de la misma, tanto en la descripción arquitectónica como en el propio diseño arquitectónico.
- ✓ Valorar la posibilidad de hacer extensiva esta propuesta a otras áreas temáticas relacionadas con la industria petrolera.

Bibliografía

1. Mastemegazine. [citado 2009 enero 17]; Disponible en: http://www.mastermagazine.info/termino/5366.php.

2. Fernández, I. and M. Suárez, *Estudio sobre la unificación de conceptos en Ciencias de la información, Bibliotecología y Archivología: Una propuesta fundamentada*. 1996, Habana. Disponible en: www.inforosocial.net/ponencias/eje02/53.pdf

3. Woodman, L., *Information management in large organizations: Information management from strategies to action*. 1985, London: ASLIB. 95-114. Disponible en: http://bvs.sld.cu/revistas/aci/vol10_5_02/aci04502.htm.

4. Victoria, in *DefiniciónABC*. 2008.

5. Pressman, R.S. *Ingeniría de Software "Un enfoque práctico"*.[citado 7 de enero de 2009].

6. Definición.de. *Definición de ingeniería de software*. [citado 2009 10 enero].

7. Bass, Clements, and Kazman. *Published Software Architecture Definitions, Modern Definitions.* 2003 [citado 2009 13 enero]; Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Modern.

8. Billy Reynoso, C. *Introducción a la Arquitectura de Software.* . 2004 [citado 2008 19 de diciembre]; Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.

9. Shaw, M. and P. Clements, *A field guide to Boxology: Preliminary classification.* abril 1996.[citado 16 de febrero de 2009].

10. Reynoso, C. and N. Kicillof, *Lenguajes de descripción arquitectónica de Sofware (ADL)*. 2004 [citado 15 de enero de 2009]. Disponible en: http://www.willydev.net/descargas/prev/ADL.pdf.

11. Tramullas, J. (1997, 2000) *Los sistemas de gestión de bases de datos.*, [citado 5 de febrero de 2009] Disponible en: http://tramullas.com/documatica/2-4.html.

12. NELSON, A. and J. ROTTMAN, *Before and after CASE adoption. Information & Management*. 1996 [citado 24 de febrero del 2009].

13. ORLIKOWSKI, W. (1993) *CASE tools as organizational change: investigating incremental and radical changes in systems developmen*. 309 - 340 [citado 22 de febrero de 2009]

14. Celis, I. *WebTaller "El ataque de los framework"*. [citado 2009 25 de enero]; Disponible en: http://www.webtaller.com/maletin/articulos/el-ataque-de-los-frameworks.php.

15. Mateos", U.P.A.L., *Instituto Politécnico Nacional*: Mexico.

16. S.A., G. *Presentación de Metodología MSF*. 7 [citado 20 de diciembre del 2008.] Disponible en: http://www.e-gattaca.com/eContent/library/documents/DocNewsNo50DocumentNo6.PDF.

17. Mendoza Sánchez, M.A. *Metodologías De Desarrollo De Software*. [citado 15 de diciembre del 2008.]Disponible en: www.willydev.net/descargas/cualmetodologia.pdf.

18. GALLEGO, J.P.G. and I.J. GALVES (2007) *FUNDAMENTOS DE LA METODOLOGIA RUP RATIONAL UNIFIED PROCESS* [citado 13 de enero de 2009] Disponible en: http://www.scribd.com/doc/297224/RUP.

19. Letelier, P. and M.C. Penadés *Metodologías ágiles para el desarrollo de software*. [citado 7 de enero del 2009.] Disponible en: www.willydev.net/descargas/masyxp.pdf.

20. Reynoso, C. and K. Nicolás (2004) *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. [citado 5 de febrero de 2009] Disponible en: http://www.willydev.net/.

21. Garlan, D. and M. Shaw, *An Introduction to Software Architecture*, CMU-CS-94-166, Editor. 1994 [citado 10 de enero de 2009].

22. International, V.P. *Build Quality Applications Faster, Better and Cheaper*. [citado 2009 27 de febrero]; Disponible en: http://www.visual-paradigm.com/product/.

23. ORACLE *Packs de Gestión de Base de Datos de Oracle*. [citado 10 de febrero de 2009] Disponible en: http://www.oracle.com/global/es/products/database/db_manageability.html.

24. PostgreSQL, E.d.d.d. *Manual de Usuario de PostgreSQL*. [citado 2009 5 de enero]; Disponible en: http://es.tldp.org/Postgresql-es/web/navegable/user/user.html

25. Bustamante, P., I. Aguinaga, and M. Aybar, *Aprenda C++ Básico*. 2004 [citado 22 de febrero de 2009]. Disponible en: www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/Cpp/basico/cppbasico.pdf.

26. IberCom. Disponible en: https://www.ibercom.com/soporte/index.php?_m=knowledgebase&_a=viewarticle&kbarticleid=935.

27. Ayala, R.M., *XML Iniciación y refernecia*, [citado 18 de febrero de 2009]. Disponible en: http://bibliodoc.uci.cu/pdf/reg01501.pdf.

28. Clements, P., Northrop, L., *Software Product Lines. Practices and Patterns*. 2001, Boston.

29. Garcés, K. (2007) *Administración de variabilidad en una línea de productos basada en modelos.*, [citado 24 de marzo de 2009]

30. Collins-Sussman, B., B. W. Fitzpatrick, and C. Michael Pilato, *Control de versiones con Subversion*. 2004 [citado 20 de febrero de 2009]. Disponible en: http://svnbook.red-bean.com/nightly/es/svn-book.pdf.

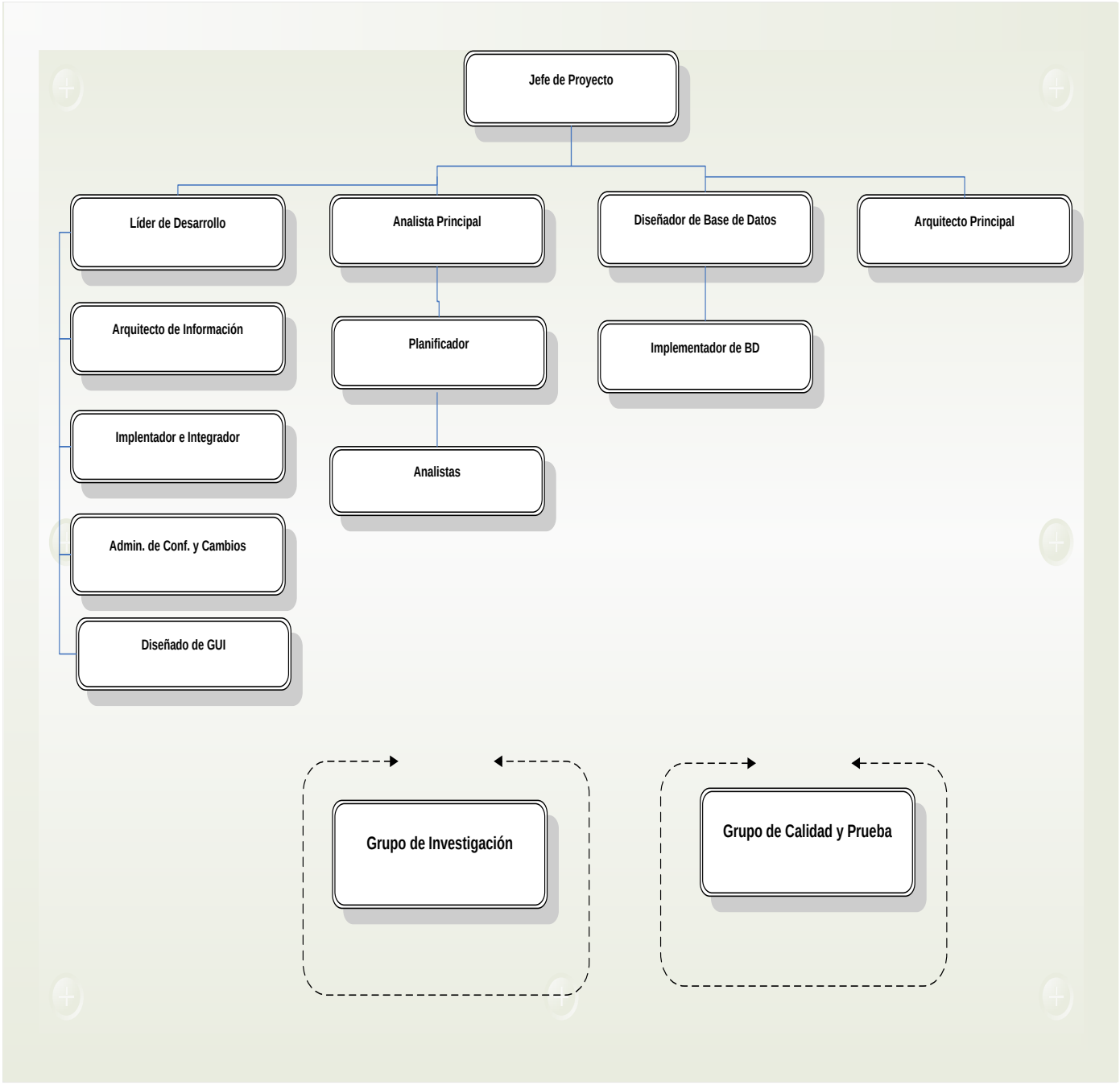
31. Küng, S., L. Onken, and S. Large, *TortoiseSVN: Un cliente de Subversion para Windows:* 2006, 22 de noviembre [citado 22de febrero de 2009].

Anexos

Anexo 1

Tipo de Patrón	Comentario	Problemas	Soluciones	Fase de Desarrollo
Patrones de Arquitectura	Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos	Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento	Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad	Diseño inicial
Patrones de Diseño	Conceptos de ciencia de computación en general, independiente de aplicación	Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes, etc.	Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC)	Diseño detallado
Patrones de Análisis	Usualmente específicos de aplicación o industria	Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes	Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. logging & reinicio)	Análisis
Patrones de Proceso o de Organización	Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización	Productividad, comunicación efectiva y eficiente	Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación	Planeamiento
Idiomas	Estándares de codificación y proyecto	Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predictibilidad.	Sumamente específicos de un lenguaje, plataforma o ambiente	Implementación, Mantenimiento, Despliegue

Anexo 2


Jefe de Proyecto
Líder de Desarrollo
Analista Principal
Diseñador de Base de Datos
Arquitecto Principal
Arquitecto de Información
Planificador
Implementador de BD
Implentador e Integrador
Analistas
Admin. de Conf. y Cambios
Diseñado de GUI
Grupo de Investigación
Grupo de Calidad y Prueba

Glosario de Términos

4GL-Los lenguajes 4GL o lenguajes de cuarta generación fueron proyectados para estar más cerca del lenguaje natural. Los lenguajes para acceder a las bases de datos son generalmente descriptos como 4GL.

API- Una interfaz de programación de aplicaciones o API (del inglés Application Programming Interface) es el conjunto de funciones y procedimientos (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Applet - Programa en lenguaje java que se utiliza en las páginas de Internet para conseguir efectos especiales que el lenguaje html no puede realizar.

CORBA-En computación, CORBA (Common Object Request Broker Architecture arquitectura común de intermediarios en peticiones a objetos), es un estándar que establece una plataforma de desarrollo de sistemas distribuidos facilitando la invocación de métodos remotos bajo un paradigma orientado a objetos.

European Computer Manufacturers Association (ECMA)- Ecma International es una organización internacional basada en membresías de estándares para la comunicación y la información.

JDBC- Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java independientemente del sistema de operación donde se ejecute o de la base de datos a la cual se accede utilizando el dialecto SQL del modelo de base de datos que se utilice.

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

JTA (del inglés Java Transaction API - API para transacciones en Java)- es parte de Java EE APIs, JTA establece una serie de Interfaces java entre el manejador de transacciones y las partes involucradas en el sistema de transacciones distribuidas: el servidor de aplicaciones, el manejador de recursos y las aplicaciones transaccionales

L2- Es un tipo de memoria intermedia entre el procesador y la memoria RAM y que almacena temporalmente los datos que presumiblemente vaya a usar el procesador de una forma inmediata.

POJO- Es el acrónimo de Plain Old Java Object. Clase del lenguaje de programación Java. Este nombre se les da a las clases que no son de algún tipo especial (EJBs, Java Beans, etcétera) y no cumplen ningún otro rol ni implementan alguna interfaz especial.

Servlet - Un programa Java del lado del servidor que ofrece funciones suplementarias al servidor.

SGML- Significa Standard Generalized Mark-up Language, Lenguaje Generalizado Estándar para el Formato de Documentos ("mark-up" es un término de imprenta que significa el conjunto de instrucciones estilísticas detalladas escritas en un manuscrito que debe ser tipografiado). Es un estándar internacional que permite definir lenguajes para dar formato a documentos (mark-up languages). Por ejemplo, el HTML, es un lenguaje de formato de documentos definido de acuerdo con SGML (o en otras palabras una aplicación de SGML) para dar formato a documentos de hipertexto.