Universidad de las Ciencias Informáticas
Facultad 9

Propuesta de Arquitectura de Software para el Sistema de Análisis Petrofísico

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

Autor: Raudel Chávez Arroyo

Tutor: Ing. Danis Rego Castillo

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

Agradecimientos

A mis padres por su amor, apoyo y por guiarme siempre por un buen camino y confiar en mi.

A mi abuela por su amor y preocupación.

A mi hermano por apoyarme en todo.

A toda mi familia por su apoyo y generosidad durante todos estos años.

A mis amigos y compañeros de estudio.

A mis amigos del barrio.

A mi tutor.

A Yusleydy.

A Olga Castro y sus a compañeras de departamento por su paciencia y dedicación.

Agradecer especialmente a Pimienta, ya que sin su colaboración hubiera sido imposible la realización de este trabajo.

Y agradecer a todos los que de una forma u otra me ayudaron en la elaboración de este trabajo.

Dedicatoria

A mis padres Zulima y Jesús.

A mi hermano Alois.

A mi sobrinito Alois Alejandro.

A mis tías Blanca, Martica y Olga.

A mis primos Lester, María de los Angeles, Indira, Evelín y Ernesto.

Dedicado especialmente a mi abuela Martha.

Resumen

El objetivo del presente trabajo de diploma es proporcionar una Arquitectura de Software que permita el desarrollo de un sistema para el análisis petrofísico de los registros de pozos. Para esto se realiza el estudio de los principales elementos que constituyen la Arquitectura de Software. Durante su desarrollo se realiza un análisis de dichos elementos, con el objetivo de lograr una organización estructural del Sistema de Análisis Petrofísico, expresada en la relación entre sus componentes, conectores, y restricciones.

La arquitectura del Sistema de Análisis Petrofísico debe de ser modular, flexible, que satisfaga con los requisitos (Analistas), que pueda ser construible (Diseñadores y Programadores), ser probado (Calidad) y que cumpla con los parámetros de funcionalidad, eficiencia y confiabilidad, además de que presente un alto grado de reutilización.

Palabras Claves: Arquitectura de Software.

Índice de Contenido

Agradecimientos ..... *I*

Dedicatoria ..... *II*

Resumen ..... *III*

Introducción *1*

Capítulo 1. Estado del Arte y Fundamentación Teórica *5*

Introducción. 5

1.1. Conceptos Asociados a la Petrofísica. 6

1.1.1. Petrofísica. 6

1.1.2. Núcleos de Pozos. 6

1.1.3. Registros de Pozos. 6

1.1.4. Necesidad de la Interpretación de los Registros de Pozos. 8

1.2. Arquitectura de Software. 8

1.2.1. Estilos Arquitectónicos. *9*

1.2.2. Marco de Trabajo. 11

1.2.3. Importancia de la Arquitectura de Software. 11

1.3. Patrones 11

1.4. Lenguajes de Descripción Arquitectónica. 12

1.5. Lenguaje Unificado de Modelado. 12

1.6. Ingeniería de Software Asistida por Computadoras. 13

1.7. Lenguajes de Programación. 13

1.8. Sistema Gestor de Bases de Datos. 14

1.8.1. Base de Datos Embebidas. 14

1.9. Empresas Internacionales Productoras de Software para el Análisis Petrofísico. 14

1.9.1. Funcionalidades. 15

1.10. Tendencias, Tecnologías, Herramientas y Metodologías actuales. 17

1.10.1. Arquitecturas o Estilos Arquitectónicos. 17

1.10.2. Metodologías de desarrollo de software. 19
1.10.3. Herramientas CASE de modelado con UML. 22
1.10.4. Lenguajes de Programación. 23
1.10.5. Lenguaje Extensible de Marcas (XML). 24
1.10.6. SGBD Embebidos. 25
1.10.7. Familia o Línea de Productos de Software (LPS). 27
Conclusiones Parciales. 28
Capítulo 2. Tecnologías y Herramientas a Utilizar *29*
Introducción. 29
2.1. Arquitectura seleccionada. 29
2.2. Marcos de Trabajo de desarrollo a seleccionados. 30
2.3. Lenguaje para Representar la Arquitectura de Software. 36
2.4. Metodología de Desarrollo de Software seleccionada. 36
2.5. Herramienta CASE para el modelado con UML seleccionada. 37
2.6. Lenguaje de Programación seleccionado. 38
2.7. SGBDR Embebido seleccionado. 39
2.8. JFreeChart 1.0.13. 40
2.9. Patrones de Diseños a utilizar. 40
2.9.1. Patrón Creador. 41
2.9.2. Patrón Experto. 41
2.9.3. Patrón Bajo Acoplamiento. 42
2.9.4. Patrón Alta Cohesión. 42
2.9.5. Patrón Fachada, (Façade). 43
2.9.6. Singleton. 44
2.9.7. Patrón Data Access Object, (DAO). 44
2.9.8. Patrones de Idioma. 45
2.10. Otras Herramientas a utilizar. 47
2.10.1. Entorno de Desarrollo Integrado (IDE). 47

2.10.2. JDK v 1.6. 47
2.10.3. Subversion (SVN). 48
2.10.4. TortoiseSVN. 48
Conclusiones Parciales. 49
Capítulo 3. Línea Base de la Arquitectura *50*
Introducción. 50
3.1. Organigrama de la Arquitectura. 50
3.2. Frameworks de Desarrollo. 55
3.3. Sistema Gestor de Base de Datos H2 Database Engine. 55
3.4. Objetivos y Restricciones Arquitectónicas. 57
3.5. Estructura del Equipo de Desarrollo. 59
3.6. Representación de la Arquitectura. 61
3.6.1. Vista de Casos de Uso. 61
3.6.2. Vista Lógica. 63
3.6.3. Vista de Implementación. 68
3.6.4. Vista de Despliegue. 69
Conclusiones Parciales. 70
Conclusiones Generales *71*
Recomendaciones *72*
Referencias Bibliográficas *73*
Bibliografía *77*
Anexos *79*
Glosario de Términos *80*

Índice de Figuras

Figura 1: Núcleos de Pozo. 6
Figura 2: Registro de Pozo (Formato ASCII). 7
Figura 3: Arquitectura en Capas. 17
Figura 4: Arquitectura Modelo-Vista-Controlador. 18
Figura 5: Arquitectura del framework Swing a nivel de componente. 30
Figura 6: Referencias de las clases Swing. 31
Figura 7: Arquitectura del framework Spring. 32
Figura 8: Diagrama de clases del patrón Fachada. 43
Figura 9: Diagrama de clases del patrón Singleton. 44
Figura 10: Diagrama de clases del patrón Data Access Object (DAO). 45
Figura 11: Arquitectura en Capas. 51
Figura 12: Estructura del equipo de desarrollo (incluye los módulos que se proponen para la segunda iteración del sistema). 60
Figura 13: Vista de Casos de Uso. 62
Figura 14: Distribución del Sistema en Capas. 64
Figura 15: División del sistema en Módulos. 64
Figura 16: Distribución para cada módulo. 66
Figura 17: Vista de Implementación. 69
Figura 18: Vista de Despliegue. 70

Introducción

El Centro de Investigaciones del Petróleo (CEINPET) es un centro cubano avalado por una alta calificación y que acumula una gran experiencia de trabajo en el sector. La misma se encarga de dar respuesta de forma integral a toda la actividad petrolera, desde la Exploración hasta la Refinación, en el proceso de investigación.

Una de las áreas de investigación del centro es la Petrofísica. Esta se dedicada al estudio de las propiedades físicas de las rocas, mediante el análisis petrofísico de los núcleos de pozos y registros de pozos extraídos de los yacimientos.

Para llevar a cabo los trabajos en dicha área, el centro cuenta con conjunto de sistemas propietarios que se encargan de interpretar y graficar las mediciones contenidas en los registros de pozos. Los mismos restringen su uso por medio de licencias, lo que provoca, unido a la falta de acceso a las actualizaciones por ser costosas para el centro, que estos sistemas puedan solo ser usados en una computadora a la vez. Esto trae consigo que se acumule y se torne engorroso el trabajo, además de que parte de sus investigadores se vean excluidos de la utilidad que brindan dichos sistemas.

Por estas razones, el centro plantea la necesidad de contar con un sistema para la interpretación de los registros, evitando grandes gastos económicos y que permita la sustitución de importaciones.

De aquí surge como propuesta de proyecto en el Polo PetroSoft de la Facultad 9 de la Universidad de las Ciencias Informáticas (UCI), el desarrollo de un sistema para el análisis petrofísico de los registros de pozo. Este debe ser capaz de equipararse con los sistemas ya existentes, ser desarrollado sobre herramientas libres que eliminen las dependencias tecnológicas y adaptarse a las exigencias y necesidades de los usuarios.

Para que el sistema cumpla con estas particularidades es necesario tener en cuenta una serie de aspectos que intervienen de forma directa en el cumplimiento de estas exigencias; estos son tratados en lo que se conoce como la Arquitectura de Software.

La Arquitectura de Software comprende la estructura de un sistema, de acuerdo a los elementos que la conforman, los protocolos por lo que se comunican y las relaciones que existen entre ellos. De su robustez y flexibilidad depende su capacidad de adaptarse a los cambios, tanto de requisitos como de tecnología.

La gran mayoría de los sistemas realizados en la Facultad 9 entran en la clasificación de sistema de gestión. Generalmente estos fueron desarrollados como aplicaciones web a partir del lenguaje de programación PHP, ASP y el uso de otras tecnologías propias de este tipo de aplicaciones. Esto trae consigo que no se cuente dentro del polo con una arquitectura de referencia que pueda ser utilizada total o parcialmente al sistema a desarrollar.

Debido a la situación problemática anteriormente expuesta surge el siguiente **problema a resolver**: *Inexistencia de una adecuada Arquitectura de Software que guíe el desarrollo de un producto para el análisis petrofísico.*

La investigación se enmarca en el **objeto de estudio:** *Proceso de desarrollo de un software para el análisis petrofísico de pozos petroleros y reservorios cubanos*, y el **campo de acción** determinado se inscribe en el *Sistema de Análisis Petrofísico.*

Para dar solución al problema se ha trazado como **objetivo general** de nuestro trabajo *diseñar una Arquitectura de Software que soporte el desarrollo de un Sistema de Análisis Petrofísico para los pozos petroleros y reservorios cubanos,* que incluye como **objetivos específicos:**

- Identificar las principales funcionalidades que brindan los productos de software existentes en el mercado.
- Identificar posible mercado.
- Definir la línea base de la arquitectura.

Para dar cumplimiento al objetivo general y específicos se han trazado las siguientes **tareas de investigación:**

- Identificar y caracterizar a grandes rasgos las empresas que dedicadas al desarrollo de software para análisis petrofísico, así como sus principales productos.
- Identificar la metodología, herramientas y tecnologías para la arquitectura del sistema.
- Identificar las necesidades y restricciones que el sistema debe tener.
- Fundamentar la utilización de la metodología, las herramientas y las tecnologías más factibles para la arquitectura del sistema.
- Fundamentar la utilización y aplicación de los estilos arquitectónicos.
- Diseñar una propuesta arquitectónica que cumpla con los requerimientos de la aplicación y que

garantice el desarrollo paralelo y la reutilización de componentes de la misma.

Para la realización de las tareas de investigación se han empleado los **métodos de investigación** que se describen a continuación.

Métodos Teóricos:

- ✓ **Analítico-Sintético**, con el objetivo de analizar y aumentar los conocimientos entorno al objeto de estudio a partir de consultar la bibliografía científica correspondiente, para después, haciendo uso de la síntesis, lograr resumir y exponer los resultados obtenidos del análisis.
- ✓ **Histórico-Lógico**, se empleó p ara estudiar la trayectoria histórica y evolución de los productos de software que existen para la interpretación de registros y su posterior análisis petrofísicos.
- ✓ **Inductivo-Deductivo**, con el objetivo de definir los procesos de análisis petrofísicos de manera general hasta llegar a las especificidades de cada sistema utilizado por CEINPET.
- ✓ **Modelación**, se empleó para modelar la Arquitectura de Software propuesta.

Métodos Empíricos:

- ✓ **Observación**, con el objetivo de obtener un registro visual de toda la información referente a los procesos de análisis petrofísicos y el funcionamiento del centro en una situación real.

- ✓ **Entrevista**, al personal que se encargan del análisis petrofísico para recopilar información confiable y real de las necesidades del centro, y que se tornan indispensables para la solución del problema de investigación.

El presente trabajo se encuentra estructurado de la siguiente forma:

Capítulo 1. Estado del Arte y Fundamentación Teórica.

En esta sección se define el estado del arte de las empresas y software asociados a la petrofísica, así como, las tecnologías actuales y conceptos relacionados a la Arquitectura de Software.

Capítulo 2. Tecnologías y Herramientas a Utilizar.

En esta sección se definen y fundamentan la utilización de la metodología herramientas y tecnologías a utilizar en la propuesta de la arquitectura del sistema.

Capítulo 3. Línea Base de la Arquitectura.

En esta sección se describe la propuesta de la arquitectura como solución al problema propuesto.

Capítulo 1. *Estado del Arte y Fundamentación Teórica*

Introducción.

La Petrofísica, como área de investigación asociada al estudio de las propiedades físicas de las rocas, interviene de forma activa en la evaluación de pozos y reservorios petroleros. Esta evaluación es indispensable en la industria petrolera en la fase de exploración. La evaluación de los pozos y reservorios tiene que ver con la interpretación de los registros de pozos y con el estudio y análisis en laboratorio de los núcleos extraídos de los pozos.

Esta área de investigación se encuentra respaldada por un conjunto de soluciones informáticas que apoyan el proceso de evaluación de los pozos y reservorios. Específicamente el proceso de interpretación de los registros de pozos, cuenta con sistemas que permiten la visualización gráfica de los datos del registro, además del cálculo de las propiedades físicas.

Una **solución informática** o **software** es un conjunto o disposición de elementos que están organizados para realizar un objetivo predefinido procesando información (1). La misma puede tener como objetivo soportar alguna función de negocio o desarrollar un producto que pueda venderse para generar beneficios.

El desarrollo de estos programas de computadoras se realiza mediante un tipo de ingeniería, la **Ingeniería de Software (ISW)**. La ISW es la aplicación de un enfoque sistemático, disciplinado y cuantificable hacia el desarrollo, operación y mantenimiento del software (2). Esta abarca un grupo de métodos, técnicas y herramientas que se utilizan en la producción del software, más allá de la actividad principal de programación.

En el desarrollo de este capítulo se enunciarán una serie de conceptos y aspectos relacionados con el negocio que sustentará al Sistema de Análisis Petrofísico, así como las empresas desarrolladoras de soluciones informáticas para la industria petrolera y los sistemas que tienen en el mercado. Se realizará además, un estudio detallado de los principales conceptos que giran en torno al objeto de estudio, así como las tendencias, herramientas, tecnologías y metodologías actuales con el objetivo de seleccionar las más adecuadas a la solución del problema.

1.1. Conceptos Asociados a la Petrofísica.

1.1.1. Petrofísica.

La Petrofísica es el estudio de las propiedades físicas y químicas que describen la incidencia y el comportamiento de las rocas, los sólidos y los fluidos (3). Esta se apoya en la utilización de herramientas que permiten el sondeo de los pozos en busca de información de las propiedades físicas de las rocas a través de imágenes acústicas y resistivas de alta resolución emitidas por dispositivos desarrollados, y el análisis en laboratorio de los núcleos y la interpretación de registros de pozos.

Para el estudio de estas propiedades físicas y químicas se necesita como materia prima los núcleos de pozos y los registros de pozos o registros eléctricos, como también se les conoce.

1.1.2. Núcleos de Pozos.

Los núcleos de pozos son muestras de las formaciones rocosas que se extraen puntualmente en los pozos. El estudio y análisis de estos se realizan en laboratorios (4).

Figura 1: Núcleos de Pozo.

1.1.3. Registros de Pozos.

Los registros de pozo son mediciones que se realizan con sensores remotos constituidos por buzo, cable y registrador de superficie. Estos registros se relacionan con las propiedades físicas, mecánicas, eléctricas y radiactivas de las secuencias de rocas que descubre un pozo (4).

```
~Version Information
VERS.  2.0 : CWLS Log ASCII Standard-VERSION 2.0
WRAP.  NO  : One line per depth step
~Well Information Block
#MNEM.UNIT      Data       Description of Mnemonic
#------------------------------------------------
STRT.M       620            : Start Depth
STOP.M      4612            : Stop Depth
STEP.M       0.25           : Step
NULL.        -9999.000      : Null Value
COMP.        EPEP OCC.      : Company
WELL.   Boca de Jaruco 501  : Well
FLD.    Boca de Jaruco      : Field
LOC.    Santa Cruz          : Location
PROV.    Habana             : Province
SRVC.    EPEP OCC.          : Service Company
DATE.                       : Log Date
UWI.         BJ 001         : Unique Well ID
~Curve Information Block
#MNEM.UNIT    API CODE      Curve Description
#------------------------------------------
DEPTH.M                     : Depth Curve
SP.mV                       : Potencial Espontáneo
SGR.gamma/cm                : Gamma Natural
CAL.mm                      : Caliper
RS.ohm-m                    : Resistividad Somera(A0.5M6.0N)
RD.ohm-m                    : Resistividad Profunda(A2.0M0.1M)
Near.u.r.c                  : Ing (Intensidad neutrón gamma)
~Parameter Information
#MNEM.UNIT    Value      Curve Description
#-----------------------------------------
~A  DEPTH        SP          SGR       CAL        RS         RD         NEAR
   662.750  -9999.000       1.804     490.361     0.347      1.553      3.564
   663.000  -9999.000       1.770     490.361     0.507      1.682      3.542
   663.250  -9999.000       1.715     485.401     0.719      1.784      3.528
   663.500  -9999.000       1.652     477.490     0.931      1.886      3.491
   663.750  -9999.000       1.587     469.579     1.144      1.987      3.428
   664.000  -9999.000       1.530     465.615     1.356      2.089      3.347
   664.250  -9999.000       1.476     457.669     1.454      2.190      3.359
   664.500  -9999.000       1.432     449.918     1.694      2.291      3.389
   664.750  -9999.000       2.160     444.018     1.737      2.415      3.418
   665.000  -9999.000       2.313     445.426     1.788      2.564      3.444
   665.250  -9999.000       2.427     447.243     1.832      2.712      3.464
```

Figura 2: Registro de Pozo (Formato ASCII).

1.1.3.1. Formato de los Registros de Pozos.

Los datos digitales representados en estos registros se presentan en dos formatos principales: Código Estándar Americano para el Intercambio de Información (ASCII, por sus siglas en inglés) y Binario (5).

El formato Log ASCII Standard (LAS), comprende archivo de datos individuales escritos en ASCII. Representa el encabezado del registro de pozo y las curvas ópticas en forma digital. Este formato es ampliamente aceptado por parte de las compañías de Exploración y Producción para el envió rápido de registros de pozos en el mundo entero. Debido a su estructura de archivo ASCII puede ser escrito y leído por cualquier sistema operativo de computadora. Además puede ser abierto con cualquier editor de texto y extraer información sobre el pozo en forma visual.

El formato de datos binarios Log Information Standard (LIS) fue producido a partir de los sistemas de

adquisición de Schlumberger[1]. Era el formato convencional de datos dentro de la industria del perfilaje hasta que fue superado por el Digital Log Interchange Standard (DLIS).

El formato DLIS es un estándar sintáctico para sísmica, perforación y perfilaje de pozos. El formato asegura la rastreabilidad requerida por la industria de Exploración y Producción, al especificar el equipamiento, las herramientas, los procesos y los datos.

La información que contienen los registros de pozos, en cualquiera de los formatos anteriores, es interpretada mediante productos de software especializados que permiten la visualización de las curvas que representan las mediciones contenidas en los registros y el cálculo de las propiedades físicas.

1.1.4. Necesidad de la Interpretación de los Registros de Pozos.

La interpretación de los registros se realiza con el fin de establecer la relación que existe entre las mediciones de los campos y las propiedades físicas de las rocas. Las mediciones tales como: resistividad, neutrones y densidad, permiten cuantificar las propiedades físicas: permeabilidad, saturaciones y la porosidad efectiva; y así, poder caracterizar un depósito de petróleo o gas. La interpretación se realiza a través de relaciones matemáticas que involucran las mediciones de los registros y los análisis en laboratorio de los núcleos.

1.2. Arquitectura de Software.

La Arquitectura de Software (AS) surgió como un campo de estudio por la década de 1960, con las tempranas observaciones de Edsger Dijkstra, de David Parnas y de Fred Brooks. Durante algún tiempo la AS quedó en estado de hibernación, hasta 1992 cuando Dewayne Perry y Alexander Wolf publicaron un artículo bajo el nombre *"Foundations for the study of software architecture"*, donde exponen lo que sería la década de 1990 para la arquitectura de software.

> *La década de 1990, creemos, será la década de la arquitectura de software. Usamos el término "arquitectura" en contraste con "diseño", para evocar nociones de codificación, de abstracción, de estándares, de entrenamiento formal (de los arquitectos de software) y de estilo* (6).

[1] Es una de las empresas de servicios para la industria petrolera más grande del mundo, y líder en este sector.

Estas ideas fueron tomadas en primera instancia por los miembros de lo que podría llamarse la universidad estructuralista de Carnegie Mellon (CMU): David Garlan, Mary Shaw, Paul Clements y Robert Allen. Junto a Rick Kazman, Mark Klein, Len Bass y otros metodólogos del Instituto de Ingeniería de Software (SEI) perteneciente a la misma universidad, comienzan a trabajar de inmediato en lo que es considerada actualmente la tendencia predominante en la AS.

Se puede decir que la AS es una práctica joven, de poco más de una década, por lo que no existe una definición universal y que sea respaldada por la totalidad de los arquitectos de sistemas. Dentro de las más reconocidas se encuentran las siguientes definiciones:

- La **Arquitectura de Software** es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (7).
- La **Arquitectura de Software** de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente, y las relaciones entre ellos (8).

Se pueden apreciar en ambas definiciones puntos de coincidencia, en cuanto a la estructura a grandes rasgos de un sistema, estructura basada en componentes o elementos como también se les conoce y las relaciones entre ellos.

En la práctica del diseño y la implementación en la AS, se siguen ciertas regularidades de configuración en respuesta a similares demandas. Estas regularidades están asociadas a que cualquier arquitectura se enmarca en un estilo arquitectónico en específico o combinación de ellos.

1.2.1. Estilos Arquitectónicos.

Un **estilo arquitectónico** es una familia de sistemas en términos de un patrón de organización estructural. Determina el vocabulario de los componentes y conectores que pueden ser utilizados, junto con una serie de restricciones sobre la forma en que pueden ser combinados (9).

Perry y Wolf lo definen como una codificación particular de elementos de diseño que limita su tipo y su acuerdo formal. Además de restringir las relaciones formales entre ellos (6).

Se puede concluir que un estilo arquitectónico brinda una estructura en cómo deben estar organizada

la aplicación en cuanto a los *componentes*, los *conectores*, las *configuraciones* y las *restricciones* de estos elementos y las relaciones que pueden existir entre ellos.

Reynoso (10) proporciona un compendio de los principales estilos arquitectónicos más representativos y vigentes, en forma de estilos y sub-estilos, destacando que esta agrupación es susceptible a realizarse de múltiples formas.

- **Estilos de Flujos de Datos.**
    - *Tubería y filtros.*
- **Estilos de Código Móvil.**
    - *Arquitectura de Máquinas Virtuales.*
- **Estilos de Llamada y Retorno.**
    - *Modelo-Vista-Controlador (MVC).*
    - *Arquitecturas en Capas.*
    - *Arquitecturas Orientadas a Objetos.*
    - *Arquitecturas Basadas en Componentes.*
- **Estilos Peer – to – Peer.**
    - *Arquitecturas Basadas en Eventos.*
    - *Arquitecturas Orientadas a Servicios (SOA).*
    - *Arquitectura Basada en Recursos (REST).*
- **Estilos de Datos Centralizados.**
    - *Arquitectura de Pizarra o Repositorio.*

Un estilo arquitectónico indicaría cuáles son los patrones y restricciones de interconexión o composición entre ellos, lo que denomina como invariantes del estilo. Por último, asociadas a cada estilo hay una serie de propiedades que lo caracterizan, determinando sus ventajas e inconvenientes, y que condicionan la elección de uno u otro estilo para resolver un determinado problema (11).

1.2.2. Marco de Trabajo.

Según Canal, un **marco de trabajo** o **framework** define una arquitectura adaptada determinado dominio de aplicación, definiendo de forma abstracta una serie de componentes y sus interfaces, y estableciendo las reglas y mecanismos de interacción entre ellos (11).

Jacobson simplifica la definición de marco de trabajo como, una arquitectura genérica que proporciona una plantilla ampliable para su aplicación dentro de un dominio (12).

Se puede concluir que un marco de trabajo no es más que una arquitectura genérica definida para un determinado dominio de aplicaciones y que contiene una serie de componentes implementados que son accesibles a través de sus interfaces que les permite ser reutilizados y redefinidos.

Son diseñados con el intento de facilitar el desarrollo de los sistemas, permitiendo a los desarrolladores pasar más tiempo identificando requerimientos del sistema, que tratando con los tediosos detalles de bajo nivel para proveer un sistema funcional.

1.2.3. Importancia de la Arquitectura de Software.

Existen varias razones que validan la importancia de la AS tanto, a una edad temprana, como a lo largo de todo el ciclo de vida del desarrollo de un sistema. A continuación mencionaremos tres razones claves que refleja Pressman (1).

- Las representaciones de la arquitectura de software facilitan la comunicación entre todas las partes interesadas en el desarrollo de un sistema basado en computadora.
- La arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional.
- La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajan juntos sus componentes.

1.3. Patrones

Según Craig Larman (13), un **patrón** es una pareja de problema/solución con un nombre y que es aplicable a otros contextos, con una sugerencia sobre la manera de usarlo en situaciones nuevas.

Canal (11) lo define solución probada que se puede aplicar con éxito a un determinado tipo de problemas que aparecen repetidamente en algún campo.

El establecimiento de los patrones posibilita el aprovechamiento de la experiencia acumulada en el diseño de aplicaciones. Pueden dividirse y clasificarse de acuerdo al nivel de abstracción de la vistas de un sistema, en: *patrones de arquitectura*, *patrones de diseño*, patrones de análisis, *patrones de proceso o de organización* y *patrones de idiomas* (ver anexo I).

1.4. Lenguajes de Descripción Arquitectónica.

Los **lenguajes de descripción arquitectónica (Architecture Description Language, ADLs)**, son leguajes (gráfico, textual, o ambos) para describir una aplicación informática en términos de sus elementos arquitectónicos y las relaciones entre ellos (14). Resultan de un enfoque lingüístico para el problema de la representación formal de una arquitectura (15).

Estos lenguajes pueden describir la arquitectura de manera formal o semiformal y sus características vienen dadas por los requerimientos que implica. Son creados para el modelado, descripción y la posterior prueba para determinar la consistencia arquitectónica. Además permiten la representación de los componentes, conectores, configuraciones y restricciones y proporcionan tanto una sintaxis específica como un marco conceptual para modelar la misma de un sistema.

Canal expone además que estos lenguajes se centran en la estructura de alto nivel de un sistema, más que en los detalles de implementación de los módulos de código fuente que lo constituyen (11).

1.5. Lenguaje Unificado de Modelado.

El **Lenguaje Unificado de Modelado (Unified Modeling Language, UML)** se define como un lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software. Es un sistema notacional destinado a los sistemas de modelado que utilizan conceptos orientados a objetos (13).

Los conceptos en UML son divididos en varias vistas que facilitan su comprensión, estas son: vista estática, vista de casos de uso, vista de máquina de estado, vista de actividades, vista de interacción y las vistas físicas. Las mismas son un subconjunto que modelan elementos que representan varios aspectos de un sistema. Los conceptos de cada vista son representados mediante una o dos clases de

diagramas que proporcionan una notación visual.

La finalidad de los diagramas es presentar diversas perspectivas de un sistema, a las cuales se les conoce como modelo. Es importante destacar que un modelo UML describe lo que supuestamente hará un sistema, pero no dice cómo implementar dicho sistema.

1.6. Ingeniería de Software Asistida por Computadoras.

La creciente demanda y complejidad de nuevos productos de software, impone a los ingenieros de sistemas informáticos un alto nivel de exigencia en cuanto a la calidad y el cumplimiento de los plazos de entrega del producto final. Por lo tanto, se hace necesario dotar al ingeniero en sistemas de un ambiente que contenga un conjunto de herramientas que controlen y organicen todo la información generada en desarrollo de un proyecto informático, sobre todo en el caso de que proyecto sea de una complejidad considerable. A la acción de usar estas herramientas para el modelado de un sistema se denomina **Ingeniería de Software Asistida por Computadora (Computer Aided Software Engineering, CASE)**, lo que permite distinguir a estas como *herramientas CASE*.

Podría definirse la Ingeniería de Software Asistida por Computadora, como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación (16).

Estos métodos proporcionan al ingeniero la posibilidad de automatizar actividades manuales y de mejorar su visión general de la ingeniería. Al igual que las herramientas de ingeniería y de diseño asistidos por computadora que utilizan los ingenieros de otras disciplinas, las herramientas CASE ayudan a garantizar que la calidad se diseñe antes de llegar a construir el producto (1).

1.7. Lenguajes de Programación.

Un **lenguaje de programación** es un sistema notacional para describir computaciones en una forma legible tanto para la computadora como para los usuarios (17).

Rafael Berlanga y José M. Iñesta (18) lo definen como un lenguaje que provee una notación sistemática para describir tanto los datos como las sentencias de un programa.

Actualmente existe una gran variedad de lenguajes de programación. Estos son agrupados en distintas

categorías de acuerdo al nivel de abstracción que poseen con respecto a la arquitectura de la máquina. Las categorías en la que son agrupados los distintos lenguajes de programación son: *los lenguajes de máquina o bajo nivel* y *los lenguajes de alto nivel*.

1.8. Sistema Gestor de Bases de Datos.

Un **Sistema Gestión de Bases de Datos (SGBD)**, consiste en una colección de datos interrelacionados y un conjunto de programas para acceder a esos datos. El principal objetivo de un SGBD es proporcionar un entorno que sea a la vez conveniente y eficiente para ser utilizado al extraer y almacenar información de la base de datos (19).

1.8.1. Base de Datos Embebidas.

Las **bases de datos embebidas** operan simbólicamente dentro de la aplicación. Lo que significa que su código esta entrelazado o incrustado como parte del programa que lo alberga. Estas reducen los gastos relacionados con la red en cuestión de llamadas, simplifica la administración y el despliegue de la aplicación. Por lo que se hace innecesaria la configuración de una red o su administración.

1.9. Empresas Internacionales Productoras de Software para el Análisis Petrofísico.

Existen a nivel internacional empresas que se dedican al desarrollo de productos de software para el cálculo y representación de los parámetros petrofísicos a partir de los registros de pozos. Estos productos son propietarios, requieren licencias para poder ser operados, a un alto costo al igual que las sus actualizaciones. A continuación se describen algunas de las empresas y los productos que han desarrollado para el área de la Petrofísica.

Hydrocarbon Data Systems Inc, es conocida como HDS. Su misión se enmarca en ser un fuerte proveedor de sistemas para el Análisis Petrofísico de Registros, para los geólogos, ingenieros y petrofísicos. Sus productos están desarrollados sobre la plataforma Windows (20).

Su principal exponente es **HDS 2000**. La función primaria del sistema es interpretar las perforaciones y en algunos casos los registros eléctricos de las perforaciones usando cualquiera de las unidades de registro Métrica o Inglesa. La aplicación es capaz de recibir y procesar los datos tanto de registros de pozo como de terreno, permitiendo al usuario identificar áreas de saturación de hidrocarburos y el cálculo de parámetros petrofísicos.

Senergy, es un grupo dedicado a la creación de soluciones informáticas para el sector de la energía a nivel global. Este grupo posee una fuerte herencia en el sector del Petróleo y Gas, y sus soluciones cubren la mayoría de los procesos de negocio que existen dentro del sector energético, incluyendo las nuevas alternativas, como es la energía geotérmica y eólica (21).

Dentro de sus principales exponentes se encuentra **Interactive Petrophysics™ (IP)**, que es vendido y comercializado exclusivamente por Schlumberger. IP es un programa avanzado para la interpretación gráfica, fue diseñado y desarrollado por petrofísicos. Es usado para la comprensión de modelos deterministas y probabilísticos para el cálculo de las propiedades físicas de las rocas a través de los registros de pozo.

Crocker Data Processing Pty Ltd, es una empresa privada fundada por Hugh Crocker y Robert Charlebois en 1983, con el objetivo de desarrollar soluciones informáticas para la exploración y producción de petróleo, específicamente para la interpretación de los registros de pozos. Ambos fundadores son expertos reconocidos internacionalmente en el campo de la petrofísica (22).

El producto del cual han sacado la mayor parte de sus ganancias es **Petrolog**. Este es un sistema especializado en la Gestión de Datos de Registros y Análisis Petrofísico que optimiza muchas de las tareas asociadas a la gestión y evaluación de los registros de pozo. Consta de versiones para diferentes plataformas como Linux, Unix, Windows 95/98/2000/XP. En el 2006 fue portado a Microsoft Windows .Net Framework, usando el lenguaje de programación C#. Todos los métodos de procesamiento y características originales de Petrolog han sido portados para esta plataforma.

1.9.1. Funcionalidades.

Los productos de software vistos anteriormente poseen un conjunto de funcionalidades comunes presentes en cada uno de ellos, y a la vez poseen otras que son únicas o propias de cada sistema.

Funcionalidades Comunes.

Dentro de las funcionalidades que presentan los productos de software en común podemos mencionar (4):

- Digitalización o importación de archivos .LAS.
- Correcciones ambientales a los registros.

- Interpretación
- Representación de las curvas de los registros originales y los resultados de la interpretación.
- Introducción de fórmulas y ecuaciones propuestas por el usuario.
- Gráficos de propiedades cruzadas.
- Histogramas de frecuencia de los registros y las propiedades.
- Correlaciones entre pozos.

Funcionalidades Únicas o Propias.

Dentro de las funcionalidades únicas o propias que presentan cada producto de software podemos mencionar (4):

- Predicción de registros utilizando redes neuronales.
- Generación de registros sintéticos para calibrar la sísmica.
- Representación de mapas de propiedades en 2D y 3D.
- Tratamiento de imágenes: Registros FMI.
- Gráficos de propiedades cruzadas.
- Interpretación de registros de tecnología más compleja: NMR, TDT y DSI.
- Análisis mineralógico.
- Integración de la información para un grupo de pozos.

Las funcionalidades que estas aplicaciones ofrecen, tanto el fundamento matemático que las respaldan como el proceso de visualización mediante gráficos 2D y 3D de la información, propician que estas realicen un fuerte uso de los recursos de la computadora. Los resultados que proporcionan son mostrados al usuario en el menor tiempo posible, permitiéndole a este interactuar con estos de forma constante y dinámica en busca de nuevos resultados. La casi totalidad sus funcionalidades obtienen los datos que necesitan localmente del ordenador, lo que las hace independientes de una conexión a red.

Estos productos de software son implementados como aplicaciones de escritorio lo que les permite ofrecer una experiencia de usuario más fluída entre una acción y otra, y una mayor rapidez en el procesamiento de la información, funcionamiento y cumplimiento de las peticiones.

1.10. Tendencias, Tecnologías, Herramientas y Metodologías actuales.

El creciente uso de soluciones informáticas en los procesos productivos y sociales, ha ocasionado el crecimiento en la confiabilidad y la calidad que estos deben tener. Para ello en la actualidad se cuentan con un conjunto de herramientas, tecnologías y metodologías que aseguran la calidad, así como el empleo de buenas prácticas para el desarrollo las mismas.

1.10.1. Arquitecturas o Estilos Arquitectónicos.

Los miles de sistemas basados en computadoras desarrollados casi desde el surgimiento de esta, utilizan uno o la combinación de los estilos arquitectónicos vistos anteriormente. Por tanto, la organización del Sistema Análisis Petrofísico deberá ser regida por algunos de estos estilos. A continuación se especifican los principales estilos a analizar, para su utilización en el sistema.

Arquitectura en Capas.

Es 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 (23).

Figura 3: Arquitectura en Capas.

Resulta útil cuando se pueden identificar distintas clases de servicios que pueden ser articulados jerárquicamente. Los componentes de cada capa consisten en conjuntos de clases. Las interacciones entre las capas ocurren generalmente por invocación de métodos, por definición los niveles de abajo no deben poder utilizar funcionalidad ofrecida por los de niveles superiores.

Ventajas

- Soporta un diseño basado en niveles de abstracción crecientes.
- Proporciona una alta reutilización.
- Modularidad del sistema.
- Facilita la localización de errores.
- Mejora el soporte del sistema.
- Ayuda a controlar y encapsular aplicaciones complejas.

Desventajas

- Puede ser difícil definir que componentes ubicar en cada una de las capas.

Arquitectura Modelo-Vista-Controlador.

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 (10):

- Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista. Maneja la visualización de la información.
- Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

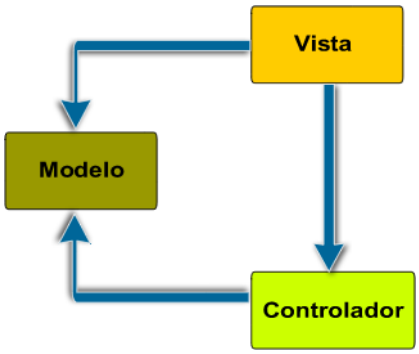

Figura 4: Arquitectura Modelo-Vista-Controlador.

Ventajas

- Soporte de vistas múltiples.
- Adaptación al cambio.

Desventajas

- Complejidad.
- Costo de actualizaciones frecuentes.

Arquitectura Orientada a Objetos.

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos son ejemplos que representan una clase de componentes (*manager)*, porque es responsable de la conservación de la integridad de un recurso (23).

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación (10).

Ventajas

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Es posible descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventajas

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.
- Problemas de efectos colaterales en cascada.
- Granularidad muy fina para sistemas grandes.

1.10.2. Metodologías de desarrollo de software.

Las **metodologías de desarrollo de software** son un conjunto de prácticas, procedimientos y

herramientas que guían la construcción de un sistema. Logrando hacer un control de todo el proceso de desarrollo y que al final del ciclo de vida del producto, este cumpla con las necesidades del cliente.

No existe una metodología de software universal. Por tanto su elección para la utilización en el desarrollo de un producto de software, debe basarse en las características específicas del proyecto. Se hace necesario además, determinar previamente el alcance del proyecto, así como su complejidad.

A continuación se describen las principales metodologías a analizar, para su guiar el desarrollo del Sistema de Análisis Petrofísico, a partir de las características esenciales de estas.

Proceso Unificado de Desarrollo de Software. (Rational Unified Process, RUP).

RUP es un proceso de desarrollo de software, que no es más que un conjunto de actividades necesarias para convertir los requisitos de un usuario en un sistema de software (24).

Utiliza el Lenguaje Unificado de Modelado (UML), para la confección de todos los esquemas de los sistemas, lo que lo hace esencial a esta metodología.

Es una metodología dirigida por casos de uso, centrada en la arquitectura, iterativo e incremental.

Consta de 4 *fases* de desarrollo de software:

- Inicio.
- Elaboración
- Construcción.
- Transición.

Estas fases se desarrollan mediante iteraciones en el ciclo de vida del proyecto. Dentro de estas fases se realizan mediante nueve *flujos de trabajos* divididos en dos grupos: los de ingeniería y los de apoyo.

Flujos de trabajo de Ingeniería:

- Modelo del negocio.
- Requerimientos.
- Análisis y Diseño.
- Implementación.

- ❖ Pruebas.
- ❖ Despliegue.

Flujos de trabajo de Apoyo.

- ❖ Gestión de cambios y configuración.
- ❖ Gestión de proyectos.
- ❖ Entorno.

Los *elementos* que componen a RUP son:

- Las **actividades**, que son los procesos que se llegan a determinar en cada iteración dentro del ciclo de vida del producto.
- Los **trabajadores**, que son todas las personas involucradas en desarrollo del proyecto.
- Los **artefactos**, son documentos, modelos y elementos de modelos que se generan a lo largo de todo el desarrollo del proyecto.

Esta metodología es usada comúnmente para proyectos de larga duración y alta complejidad. Presenta como particularidad que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo de software.

Microsoft Solutions Framework (MSF).

MSF es una flexible e interrelacionada 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.

Los modelos de proceso de MSF son:

- **Modelo de Cascada.**

  Este modelo utiliza tramos como puntos de transición y de carga. Al usar el modelo de cascada, se necesitaría completar un conjunto de tareas en forma de fase para después continuar con la fase próxima.

- **Modelo en Espiral.**

  Este modelo se basa en la necesidad continua de refinar los requerimientos para un determinado proyecto. El modelo espiral es eficaz cuando se utiliza para el rápido desarrollo de proyectos muy pequeños.

- **Modelo de Proceso MSF.**

  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.

  El modelo de proceso MSF, propone una secuencia generalizada de actividades para la construcción de soluciones empresariales. Este proceso es flexible y se puede adaptar al diseño y desarrollo de una amplia gama de proyectos de una empresa.

  Consta de cinco *fases* distintas:

  - Previsión.
  - Planeamiento.
  - Desarrollo.
  - Estabilización.
  - Implementación.

Puede ser adaptada a proyectos de cualquier dimensión y complejidad, y usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.10.3. Herramientas CASE de modelado con UML.

Las herramientas CASE de modelado con UML permite aplicar la metodología de análisis y diseño orientados a objetos y abstraernos del código fuente, en un nivel donde la arquitectura y el diseño se tornan más obvios y más fáciles de entender y modificar (25).

Dentro de las herramientas más populares basadas en UML, como lenguaje de modelado para la construcción de los distintos tipos de diagramas, están: Rational Rose y Visual Paradigm for UML.

Rational Rose Enterprise.

Rational Rose es una herramienta de modelado visual para apoyar el análisis y diseño de sistemas orientados a objetos (26). Permitiendo además el modelado de un sistema antes de haber escrito una sola línea de código, asegurando un sistema arquitectónicamente sano. El modelo Rose es una imagen de un sistema desde varias perspectivas. Estos incluyen todos los diagramas de UML, actores, casos de uso, objetos, clases, componentes y nodos de despliegues en un sistema.

Rose permite la aplicación de la metodología RUP. Soporta el desarrollo iterativo incremental del proceso de desarrollo de software. Cubre todo el ciclo de vida de un proyecto con la concepción y formalización del modelo, la construcción de los componentes, la transición del producto a los usuarios y la certificación de cada fase.

Visual Paradigm para UML.

Visual Paradigm for UML es una herramienta CASE que cubre un amplio rango de usuarios, tales como: Ingenieros de Sistemas, Analistas de Sistema, Analistas Comerciales, Arquitectos de Sistema (27). Consta de versiones que soportan todas las versiones de UML, incluyendo la 2.0 y cuenta con versiones para diversas plataformas como Windows, MacOSX y Linux.

Visual Paradigm permite la generación de código a partir de los modelos a lenguajes como: Java y C#. Además de la generación y exportación de la documentación en distintos formatos, como son, PDF, HTML, Word, entre otros.

Tiene la capacidad de importar y exportar archivos XML, proyectos generados en Rational Rose, los modelos de datos realizados en el ERwin. Consta de interfaces de colaboración con herramientas de gestión de repositorios como el Subversion y CVS. Además se integra con IDEs (Eclipse y NetBeans) para soportar la fase de implementación del proceso de desarrollo de sistemas.

1.10.4. Lenguajes de Programación.

Java.

Fue creado por la Sun Microsystem en 1991. Se rige estrictamente por la teoría de la Programación Orientada a Objetos (POO). El polimorfismo y el encapsulamiento son características intrínsecas de

Java. Posee como característica adicional que es multi-hilos, lo que le permite atender eficientemente varias tareas a la vez (28).

Las nuevas aplicaciones creadas con este lenguaje tienen su basamento en un conjunto de clases ya pre-existentes que forman parte del propio lenguaje (el API o Application Programming Interface). Además son multiplataforma, debido a la Máquina Virtual de Java (JVM). Esta se compone de una serie de especificaciones independientes de cualquier característica particular de un tipo de arquitectura, que definen a su vez el ambiente de programación y permiten la ejecución de programas escritos en Java sin tener que cambiar su código.

El desarrollo, compilación y ejecución de programas en Java se logra mediante conjunto de programas y librerías embebidas en el Java Development Kit (JDK). Además tiene incorporado un Debbuger para la detección y corrección de errores. El Java Runtime Environment (JRE), es una versión reducida de JDK destinada únicamente a ejecutar código Java.

C Sharp (C#).

C# es una versión avanzada de C y C++ y se ha diseñado especialmente para el entorno .NET. C# es un nuevo lenguaje orientado a objetos empleado por los programadores para desarrollar aplicaciones que se ejecuten en la plataforma .NET. Mezcla la potencia de C, las capacidades de orientación a objetos de C++ y la interfaz gráfica de Visual Basic (29).

La plataforma .NET es un entorno de desarrollo integrado, mientras que la C# es un lenguaje de programación creado para el desarrollo de aplicaciones en dicha plataforma. Aunque ya existe un compilador implementado que provee el Framework de DotGNU - Mono para una plataforma diferente como Win32 o UNIX / Linux (30).

Con C# pueden ser creadas aplicaciones cliente/servidor y soporta todas las características propias del paradigma de programación orientada a objetos: encapsulación, herencia y polimorfismo.

1.10.5. Lenguaje Extensible de Marcas (XML).

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 (31). XML contiene, formas,

etiquetas, estructuras, y protege la información. Todo ello realizado con símbolos incrustados en el texto, llamados *marcas* o *etiquetas*. Las marcas aumentan el sentido de la información en ciertas formas de la identificación de las partes y cómo se relacionan entre sí (32).

Se propone como un estándar para el intercambio de información estructurada entre diferentes plataformas. Se puede usar en bases de datos, editores de texto, hojas de cálculo y casi cualquier cosa imaginable. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

1.10.6. SGBD Embebidos.

Los principales SGBD son agrupados en dos grupos: SGBD Relacional (SGBDR) y SGBD Orientadas a Objetos (SGBDOO).

Los SGBDR son ideales para aplicaciones tradicionales que soportan tareas administrativas. Pero la creciente creación de aplicaciones sobre el paradigma de Programación Orientada a Objetos, están caracterizadas por estar compuesta de objetos complejos relacionados entre sí, también de forma compleja.

La representación de tales objetos y relaciones en un modelo relacional implica que los mismos deben ser descompuestos en una gran cantidad de tuplas. Luego, como las tablas están anidadas profundamente, se requieren de juntas (*joins*) para recuperar un objeto lo cual disminuye drásticamente el desempeño de la aplicación.

Los SGBDOO son ideales para almacenar y recuperar objetos complejos, permitiendo a la aplicación la utilización de estos objetos de forma directa y sin ningún otro proceso intermedio.

El SGBD ideal a utilizar en el Sistema de Análisis Petrofísico es del tipo OO. Pero la falta de disponibilidad de estos en el mercado de forma libre y sin ninguna restricción comercial, hace optar por la utilización de un SGBDR.

Dentro de los SGBDR embebidos más utilizados se encuentran:

SQLite.

SQLite es un SGDBR compatible con ACID, y que está contenida en una pequeña (~500KB) biblioteca en C. SQLite es un proyecto de dominio público creado por D. Richard Hipp (33).

Esta biblioteca implementa la mayor parte del estándar SQL-92, incluyendo las transacciones de base de datos atómicas, consistencia de base de datos, aislamiento y durabilidad (ACID), disparadores (triggers) y la mayor parte de las consultas complejas.

En su versión 3.0, permite bases de datos de hasta 2 Terabytes de tamaño y la inclusión de campos tipo BLOB[2].

H2 Database Engine.

H2 Database Engine es un SGBDR programado en Java, y está contenida en una biblioteca de 1 MB. Puede ser incorporado en aplicaciones Java o ejecutarse de modo cliente-servidor. Soporta los estándares SQL, y posee una API para JDBC. Está disponible como una aplicación de código libre bajo la Licencia Pública de Mozilla (MPL) o la Eclipse Public License (EPL).

Esta base de datos soporta diferentes tipos de conexión:

- En modo embebido (conexión local usando JDBC).
- En modo servidor (conexión remota usando JDBC u ODBC sobre TCP/IP).
- En modo mixto (conexión local y remota al mismo tiempo).

Apache Derby.

Apache Derby es un subproyecto de Apache DB. Es una SGBDR implementada completamente en Java y disponible bajo la licencia Apache v 2.0. Puede ser embebida en aplicaciones Java y utilizada para procesos de transacciones online. La biblioteca tiene un peso de 2.0 MB, que contiene el motor y el driver para JDBC. Actualmente se distribuye como Sun Java DB (34).

Posee tres productos asociados:

- Derby Embedded Database Engine: El motor propiamente dicho.

[2] Los **BLOB** (**B**inary **L**arge **OB**jects, *Grandes Objetos Binarios*) son elementos utilizados en las bases de datos para almacenar datos de gran tamaño que cambian de forma dinámica. Generalmente estos datos son imágenes, archivos de sonido y de multimedia.

- Derby Network Server: Permite convertir Derby en una base de datos que sigue el modelo cliente-servidor tradicional.
- Database Utilities: Un paquete de utilidades.

1.10.7. Familia o Línea de Productos de Software (LPS).

Una **Familia** o **Línea de Productos (LPS)** es un grupo de soluciones informáticas que comparten un conjunto común y gestionado de aspectos que satisfacen las necesidades específicas de un segmento de mercado y que son desarrollados a partir de un conjunto común de activos de software que ya se encuentran desarrollados y probados (35).

Un activo de software reutilizable es un producto de software diseñado expresamente para ser utilizado múltiples veces en el desarrollo de diferentes sistemas o aplicaciones, estos pueden ser: un componente de software, una arquitectura de dominio, un modelo de negocio, un patrón de diseño, la documentación de un sistema, entre otros.

Las ventajas de establecer estas líneas de productos tienen que ver fundamentalmente con la reutilización, ya que implican una disminución de costes y un aumento de la calidad y fiabilidad del producto. Estas ventajas se materializan en (11):

- **Reutilización de la arquitectura.** Todas las aplicaciones de la línea comparten básicamente la misma arquitectura, lo que permite reutilizar su diseño, su documentación y asegurar el cumplimiento de determinadas propiedades.
- **Reutilización de los componentes.** 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 permite corregir defectos y aumentar su fiabilidad y calidad. Por otro lado, el uso de dichos componentes en aplicaciones que comparten la misma arquitectura permite garantizar que no hay pérdida de fiabilidad en la reutilización.
- **Planificación**. La experiencia repetida en el desarrollo de productos de la línea permite hacer una planificación más fiable del proyecto de desarrollo, y con mayores posibilidades de ser cumplida.

Conclusiones Parciales.

En este capítulo se han expuesto un conjunto de conceptos asociados al dominio del área de la Petrofísica, así como las empresas dedicas al desarrollo de soluciones informáticas para la industria petrolera y sus soluciones. Además de las metodologías, herramientas y tecnologías para lograr una arquitectura robusta y un posterior desarrollo del Sistema de Análisis Petrofísico.

Capítulo 2. *Tecnologías y Herramientas a Utilizar*

Introducción.

En presente capitulo se definirá la metodología de desarrollo de software que será aplicada para la confección del Sistema de Análisis Petrofísico. También serán definidas las herramientas CASE más factibles para la confección del ambiente de trabajo donde será desarrollada la aplicación. Así como las tecnologías, estilos arquitectónicos y patrones de diseño que regirán la propuesta arquitectónica y el lenguaje por medio del cual será representada.

2.1. Arquitectura seleccionada.

Se ha seleccionado utilizar la arquitectura en Capas para el desarrollo del Sistema de Análisis Petrofísico, siguiendo como objetivo la separación de incumbencias en cada una de las capas y lograr una mayor reutilización, escalabilidad y facilidad para el desarrollo del sistema.

Esta arquitectura soporta un diseño basado en niveles de abstracción crecientes, lo que permite la partición de un problema complejo en una secuencia de pasos incrementales. El estilo admite optimizaciones, refinamientos; proporciona además una amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes.

El sistema será divido en tres capas verticales:

- Capa de presentación: será la encargada de la presentación de la información y del manejo de los aspectos relacionados con la lógica de presentación de la aplicación. Además de la validación de los datos de entrada y de los formatos de los datos de salida.
- Capa de lógica de negocio: será la encargada de contener la implementación de las tareas y reglas del negocio.
- Capa de acceso a datos: será la encargada de todo lo relacionado con la persistencia de los objetos que se manejan en el negocio y necesiten ser almacenados; además de la actualización y recuperación de estos.

2.2. Marcos de Trabajo de desarrollo a seleccionados.

Los marcos de trabajo son arquitecturas definidas para un determinado dominio de la aplicación y contienen un número determinado de componentes implementados, con sus interfaces bien definidas que pueden ser reutilizados o redefinidos. En la mayoría de los casos un marco de trabajo implementa uno o más patrones de diseño que aseguran la escalabilidad del producto.

A partir de la organización de arquitectura a través del estilo arquitectónico definido, se propone la utilización de tres frameworks distribuidos en las tres capas del sistema con funcionalidades bien definidas y componentes implementados para cada una de estas.

- **Framework de Aplicaciones Swing (JSR 296).**

  El framework de Aplicaciones Swing (JSR 296) es una especificación de Java para proporcionar un framework de aplicación simple para aplicaciones de Swing. Este definirá la infraestructura común a la mayor parte de aplicaciones de escritorio, haciendo las aplicaciones de Swing más fáciles a crear (36). Permite la creación de interfaces graficas de usuarios (GUI) utilizando el lenguaje de programación y plataforma Java. Los sistemas de ventanas y componentes gráficos permiten que las GUI construidas a partir de estos, sean independiente del sistema operativo, ya que vienen incluidos dentro del JDK.

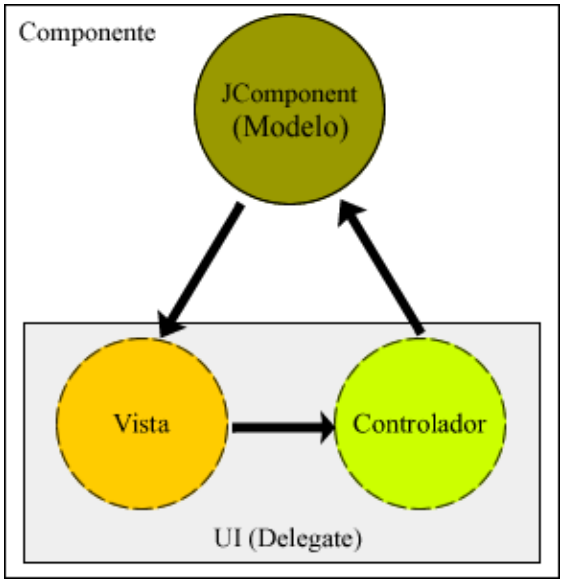

Figura 5: Arquitectura del framework Swing a nivel de componente.

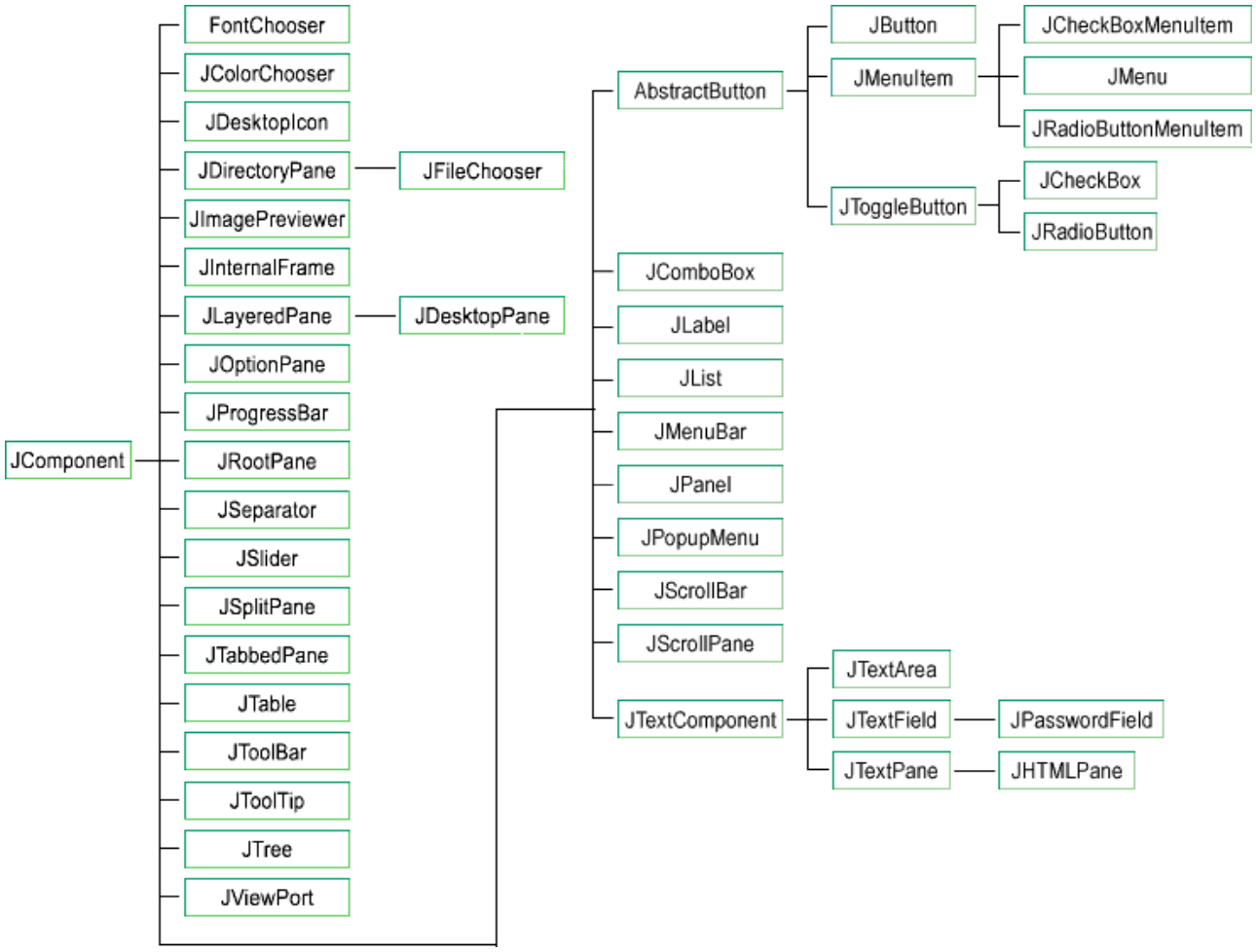

Figura 6: Referencias de las clases Swing.

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.
    - Vista: Es cómo se muestra el objeto en la pantalla.

- Controlador: Es lo que define el comportamiento del objeto.

- 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: Se puede personalizar el aspecto de las interfaces o utilizar varios aspectos que existen por defecto (Metal Max, Basic Motif, Window Win32).
- Diálogos personalizados: Se pueden crear multitud de formas de mensajes y opciones de diálogo con el usuario, mediante la clase *JOptionPane*.

- **Framework Spring 2.5.**

Spring es un framework contenedor liviano de objetos cuyas características principales son la Inversión de Control (Inversion of Control, IoC) y la Programación Orientada a Aspecto (Aspect-Oriented-Programming, AOP). Además es un framework de código abierto, que define la forma de desarrollar aplicaciones J2EE, dando soporte y simplificando la complejidad propia de los sistemas corporativos.

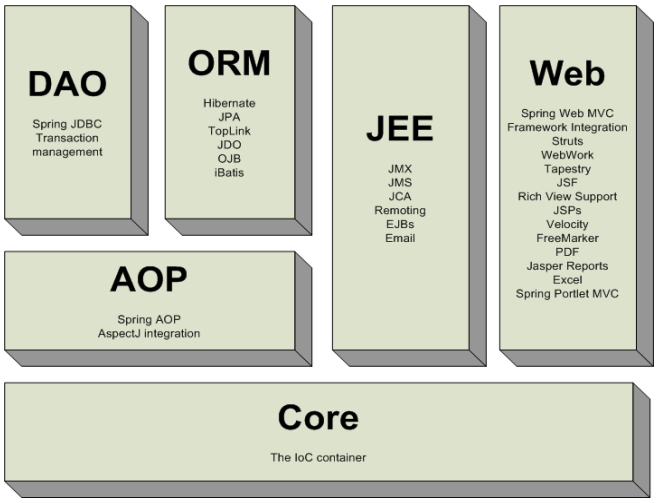

Figura 7: Arquitectura del framework Spring.

El módulo **Core** es la parte fundamental del framework y provee los rasgos de Inversión de Control y la Inyección de Dependencia (Dependency Injection, DI). El concepto básico de este módulo es el BeanFactory, que proporciona una implementación sofisticada del patrón Factory, que elimina la proliferación de Singletons y permite desacoplar la configuración y la especificación de dependencias de la lógica de negocio.

El módulo **AOP** presenta una estructura simplificada para el desarrollo y utilización de aspectos. Este permitirá el control aquellas partes del sistema que no tienen nada que ver su la lógica de negocio pero que serán necesarias. Además proporcionará modularidad y posibilitará una mejor separación de responsabilidades.

El módulo **DAO** proporciona una capa de abstracción de JDBC que elimina aspectos tediosos del código y análisis sintácticos específicos de la base de datos; accesos mediante JDBC con manejo de transacciones (desde el módulo AOP).

El módulo **ORM** proporciona las capas de integración para APIs de mapeo de objeto-relacional (Object-Realacional Mapping,ORM) como JDO, Hibernate e iBatis.

El módulo **J2EE** proporciona acceso e integración con servicios Enterprise.

El módulo **Web** provee un contexto apropiado para el desarrollo de aplicaciones web e integración con otros frameworks.

Características:

- Mediante los POJOs[3] (Plain Old Java Object), Spring revalora la simplicidad de las clases Java aportando manejo de transacciones de forma no intrusiva.
- La configuración del framework se basa en archivos XML.
- Provee un paquete de prueba específico para componentes del framework e integrado con JUnit.
- Utiliza la AOP para ofrecer manejo de transacciones declarativo sin utilizar un contenedor EJB; de esta forma, el control de transacciones se puede aplicar a cualquier POJO.
- Implementa patrones de diseño como Factory, Abstract Factory, Builder, Decorator,

[3] Acrónimo utilizado por programadores Java para enfatizar el uso de clases simples y que no dependen de un framework en especial.

Service Locator. Estos patrones son ampliamente usados en la industria del software.

- Facilita buenas prácticas de programación orientada a objetos mediante el uso de interfaces.
- Se centra en el manejo de objetos dentro de una arquitectura en Capas.
- Reduce la proliferación de Singletons.

Estas son algunas de las características claves que hacen de Spring un framework ideal para la implementación de lógica de negocio. De ahí su selección para la implementación de la lógica de negocio del Sistema de Análisis Petrofísico.

- **Framework Hibernate 3.2.**

Hibernate es un framework 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 (37). Es decir, Hibernate es un mapeador objeto-relacional (Object-Relational Mapping, ORM), encargado de automatizar y hacer transparente la persistencia de objetos en una aplicación Java en las tablas de una BD relacional, usando metadatos que describen el mapeo entre los objetos y la base de datos (38).

Características:

- Es un ORM no intrusivo, es decir, las clases persistentes no necesitan implementar ninguna interfaz.
- Cada clase persistente necesita un fichero XML de mapeo. A partir de él, obtiene toda la información necesaria para realizar las operaciones CRUD.
- Utiliza HQL, el cual es un dialecto orientado a objeto de SQL. Este permite:
    - Llamar a funciones SQL definidas por el usuario.
    - Consultas anidadas.
    - Consultas polimórficas, es decir, consultas por instancias de una clase y todas las instancias de sus subclases.
    - Outer Joins, que no es más que la recuperación de múltiples objetos por columnas.
- Posee una Caché, la cual puede aumentar significativamente el rendimiento de una

aplicación, sobre todo en aquellas en las que leen muchos datos. Esta mantiene una representación de la BD cerca de la aplicación, en memoria o en disco.

La utilización de un ORM como Hibernate en el Sistema de Análisis Petrofísico, permitirá reducir el impacto negativo sobre la aplicación, en torno a la necesidad de utilizar una Base de Datos Relacional para persistir los objetos manipulados por el sistema.

Otros frameworks de apoyo al desarrollo:

- **JUnit 4.5.**

JUnit es un marco de trabajo que permite realizar la ejecución de clases Java de manera controlada, y poder evaluar si cada método de la clase se comporta como se espera (39).

El programador puede utilizar este marco de trabajo para construir sus casos de prueba y ejecutarlos automáticamente. Los casos de prueba son realmente programas Java. Quedan archivados y pueden ser re-ejecutados tantas veces como sea necesario.

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

Características propias de la versión 4.5:

- Incluye anotaciones (*Java 5 annotations*) en lugar de utilizar herencia:
  - @Test sustituye a la herencia de TestCase.
  - @Before y @After como sustitutos de *setUp* y *tearDown*.
  - Se añade @Ignore para deshabilitar tests.
- Permite timeouts en los tests.
- Configurar excepciones esperadas.
- Ordenación, priorización, categorización y filtrado de tests.
- *Forward* y *backward* compatibilidad.
- Logging.
- Más tipos de aserciones (ej: *assertEquals(Object[], Object[])*).

- Se elimina la distinción entre errores (*errors*) y fallos (*failures*).

2.3. Lenguaje para Representar la Arquitectura de Software.

Como se ha visto en el acápite 1.2, la AS proporciona una visión global del sistema a construir. Describe la estructura y organización de los componentes de un sistema, sus propiedades y las relaciones entre ellos, sirviendo de comunicación entre las personas involucradas en el desarrollo del sistema. Ayuda además a la toma de decisiones de diseño tempranas y proporciona un mecanismo para evaluar los beneficios de las estructuras del sistema alternativas.

Para que la AS se torne una herramienta útil para el ingeniero de software en el desarrollo de un sistema basado en computadora, se hace necesario que se cuente con una manera eficiente para su representación.

Para la representación de AS se ha seleccionado UML (acápite 1.5), conocido en la industria y soportado por herramientas y metodologías de desarrollo, siendo adoptado como notación estándar por empresas productoras de aplicaciones informáticas a nivel mundial.

La representación de la AS mediante UML se hace de manera semi-formal, con la ventaja de que UML pueda ser usado en todas las etapas de desarrollo de un sistema y su representación gráfica pueda ser usada para la comunicación con los usuarios.

2.4. Metodología de Desarrollo de Software seleccionada.

Para guiar el proceso desarrollo del Sistema de Análisis Petrofísico, se ha seleccionado como metodología el Proceso Unificado de Desarrollo de Software (RUP). RUP junto con UML, constituye una potente metodología usada para el análisis, diseño, implementación y documentación de complejos sistemas informáticos. Además existen herramientas que soportan todas las fases y flujos de trabajos de esta metodología lo que facilita en gran medida su aplicación en el desarrollo del sistema.

Esta metodología es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas informáticos, para diferentes áreas de aplicación, diversas organizaciones, distintos niveles de aptitud y proyectos de diversos tamaños y complejidad (24).

RUP se caracteriza por ser un proceso formal, ya que proporciona un acercamiento disciplinado a la asignación de tareas y responsabilidades, es decir, cada miembro del equipo de desarrollo desempeña un rol, que tiene asignado un conjunto de responsabilidades y tareas específicas.

Esta metodología presenta tres características fundamentales que la hacen excepcional. RUP es un proceso *guiado por casos de uso*, *centrado en la arquitectura* e *iterativo e incremental*.

Los casos de uso son fragmentos de funcionalidad de un sistema y mediante ellos se representan los requisitos funcionales. El conjunto de todos los casos de uso forman el modelo de casos de uso, el cual describe la funcionalidad total del sistema. Los casos de uso no solo permiten la especificación de los requisitos del sistema, sino que además, guían su diseño, implementación y prueba. Por lo que se puede decir que es un proceso de desarrollo **guiado por casos de uso.**

Al ser RUP un proceso de desarrollo **centrado en la arquitectura** permite al arquitecto de software enfocarse en los objetivos adecuados, como la capacidad de adaptación al cambio y la reutilización. La arquitectura de software debe permitir el desarrollo de todos los casos de uso, así como, todos los casos de uso deben encajar en la arquitectura cuando se llevan a cabo.

RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Además divide el trabajo en partes más pequeñas o miniproyectos. Cada miniproyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son miniproyectos. De aquí la característica de que sea un proceso **iterativo e incremental.**

Con la eliminación de algunas de estas características el valor del Proceso Unificado de Desarrollo de Software caería drásticamente. Esto se debe a que se encuentran estrechamente relacionadas. La arquitectura proporciona la estructura sobre la cual guiar las iteraciones, mientras que los casos de uso definen los objetivos y dirigen el trabajo de cada iteración (24).

2.5. Herramienta CASE para el modelado con UML seleccionada.

Visual Paradigm for UML Enterprise Edition ha sido la herramienta CASE para el modelado con UML seleccionada para el desarrollo del proyecto. Esta herramienta da soporte completo al ciclo de vida del proyecto, permitiendo análisis y diseño orientados a objetos, construcción, pruebas y despliegue.

Además soporta la aplicación de RUP, así como la generación de la documentación y el grupo de artefactos característicos de esta metodología.

Además brinda distintas funcionalidades, como:

- Modelamiento de los Proceso de Negocio.
- Modelamiento Visual.
- Diseño de Interfaz de Usuario
- Integración a Entornos de Desarrollo Integrados.
- Ingeniería Directa e Inversa.
- Generación de código, entre el que se incluye código Java.
- Integración con Concurrent Version System (CVS) y Subversion (SVN).

2.6. Lenguaje de Programación seleccionado.

Para la implementación del sistema se ha seleccionado como lenguaje de programación Java, teniendo en cuenta un grupo de características que lo hacen sobresalir con respecto a otros.

Estas características son:

- Simple: Java incorpora nuevas tareas como un recolector de elementos no utilizados automático y elimina aspectos confusos y poco utilizados presentes en otros lenguajes. Otro aspecto es que Java no presenta nada realmente nuevo, todo procede de algún otro lugar, permitiendo que las funcionalidades procedan de prácticas ya usadas y probadas.
- Orientado a Objetos: Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Java trabaja con sus datos como objetos y con interfaces a esos objetos.
- Robusto: La robustez es la medida de la fiabilidad de un programa. Java contiene varias funciones integradas que mejoran la fiabilidad de un programa:
    - No tiene punteros: En Java hay referencias en lugar de punteros, evitando así la corrupción accidental de la memoria y el desbordamiento de una pila.
    - Realiza automáticamente la recolección de elementos no utilizados: Los programadores no tiene que preocuparse por la liberación de memoria, ni por la

parte de la memoria que deben liberar.

- Fomenta el uso de interfaces en lugar de clases: En lugar de pasar clases, se pasan interfaces, ocultando así las implementaciones. Si la implementación cambia, mientras la nueva clase implemente la antigua interfaz, todo lo demás funcionará perfectamente.

- Seguridad: Muchas funciones que hacen a Java robusto garantizan la seguridad del programa, como es la ausencia de punteros, evitando así la corrupción de la memoria.
- Arquitectura Neutral: En lugar de compilarlos en binarios específicos de cada plataforma, los programas de Java se realizan para que se ejecuten en cualquier equipo sin recompilarlos o revincularlos.
- Portabilidad: Los programas pueden ser ejecutados en cualquier arquitectura, es decir, tienen total independencia de la plataforma, hardware o software donde se ejecuten.

2.7. SGBDR Embebido seleccionado.

Proporcionar al Sistema de Análisis Petrofísico de un SGBD auto contenido, le permitirá ser desplegado en cualquier computadora que cumpla con los requisitos mínimos de hardware y software, sin la necesidad de un punto de conexión de red, ni la utilización de recursos adicionales para proporcionar un servidor de BD al sistema.

Se ha seleccionado utilizar dentro del Sistema de Análisis Petrofísico el SGBDR embebido H2. Este se caracteriza por tener un motor de base de datos muy rápido, logrando que las operaciones sobre las base de datos H2 se realicen con una gran velocidad. Está completamente escrita en Java y es de código abierto.

Este SGBD puede correr en diferentes modos, como son:

Modo Embebido: el SGBD se encontrará alojado dentro del sistema, dependiendo completamente su inicialización, cuando el sistema lo haya hecho. De igual forma se detendrá cuando el sistema lo haga. En este modo la base de datos puede ser creada de dos formas:

1- En **disco**: la BD se creará en el disco duro, y podrá ser accedida por el sistema desde su ubicación en el disco.

2- En **memoria**: la BD se creará en memoria. Las operaciones sobre estas se realizarán de forma más rápida que en disco, pero al terminar el tiempo de ejecución del sistema, esta se perderá.

Modo servidor: el SGBD correrá como un servidor fuera del sistema, no dependiendo del tiempo de ejecución de este. Además podrá ser accedido por otras aplicaciones a través de la red.

Soporta una gran variedad de estándares SQL, múltiples esquemas; es capaz de ejecutar funciones Java, disparadores (triggers) y permite procedimientos almacenados. Además modos de compatibilidad con otras BD como: IBM DB2, Apache Derby, MS SQL Server, MySQL Oracle y PostgreSQL.

En cuanto a seguridad incluye una solución para las inyecciones SQL. Para las conexiones en modo servidor, las contraseñas de los usuarios nunca son transmitidas en texto plano a través de la red. Todos los ficheros de base de datos pueden ser encriptados usando los algoritmos AES-256 y XTEA.

2.8. JFreeChart 1.0.13.

JFreeChart es una librería grafica libre escrita 100% en Java, que facilita a los desarrolladores mostrar gráficos de calidad profesional en sus aplicaciones.

Dentro de su amplio conjunto de características se encuentran:

- Posee un API consistente y bien documentada.

  La venta de esta documentación es la vía por la que su creador David Gilbert y su compañía Object Refinery Limited, obtienen los recursos necesarios para la continuar el desarrollo de esta librería.

- Soporta muchos tipos de salida, incluyendo compontes de Swing, archivos de imágenes (incluyendo PNG y JPEG) y ficheros con formato de gráficos vectoriales (incluyendo PDF, EPS y SVG).
- Es una librería de código abierto, liberada bajo la licencia LGPL, que permite su uso en la creación de aplicaciones propietarias.

2.9. Patrones de Diseños a utilizar.

Los patrones de diseño son soluciones a problemas comunes de diseño en un determinado contexto, y que están presentes en el desarrollo de software. Además de brindar una solución ya probada.

Permiten formalizar un vocabulario común entre los diseñadores del sistema y estandarizar el modo en se realiza el diseño. La utilización de los patrones de diseño permitirá entender de manera fácil, de mantener y ampliar el sistema, así como contribuir a la reutilización y diseño de componentes de software, a la organización del código, a la flexibilidad y extensibilidad, y la facilidad de realizar cambios en el sistema.

A continuación se describirá los patrones que serán utilizados en el diseño del sistema.

2.9.1. Patrón Creador.

Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos.

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.

2.9.2. Patrón Experto.

Este patrón es utilizado más que cualquier otro para asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos. Permite la asignación de responsabilidades a la clase que cuenta con la información necesaria para realizarla.

Dentro de sus ventajas están:

- Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.

- El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases "sencillas" y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

2.9.3. Patrón Bajo Acoplamiento.

Este patrón estimula asignar una responsabilidad de forma tal que su empleo no incremente el acoplamiento, al punto que produzca los resultados negativos propias de un alto acoplamiento. Además soporta el diseño de clases más independientes, reduciendo el impacto de los cambios, y más reutilizables, que incrementan la oportunidad de una mayor productividad.

Este patrón no puede considerarse en forma independiente a patrones como Experto y Alta Cohesión.

La utilización de este patrón incrementa su importancia cuando se busca la reutilización de los componentes al hacerlo más independiente.

Dentro de sus ventajas están:

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

2.9.4. Patrón Alta Cohesión.

Este patrón indica que una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas.

Dentro de sus ventajas están:

- Mejoran la claridad y la facilidad con que se entiende el diseño.
- Se simplifican el mantenimiento y las mejoras en funcionalidad.
- A menudo se genera un bajo acoplamiento.
- La ventaja de una gran funcionalidad soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

2.9.5. Patrón Fachada, (Façade).

Consiste en crear una única clase de manejo más fácil que nos permita acceder a un conjunto más o menos numeroso y complicado de clases que necesitan una inicialización compleja. Sirve a menudo para suministrar una interfaz pública a un paquete de servicios.

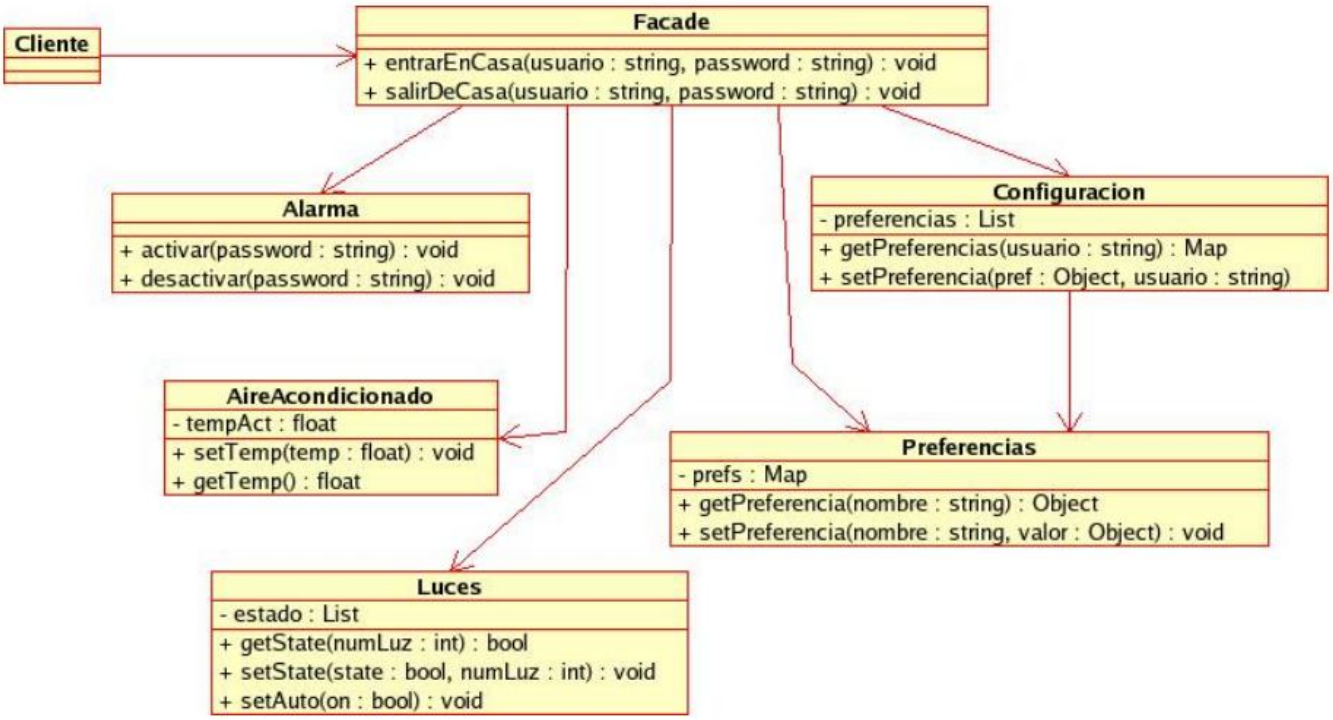

Figura 8: Diagrama de clases del patrón Fachada.

Este patrón es recomendable usarlo en presencia de las siguientes situaciones:

- 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”).

Dentro de sus ventajas están:

- Separa al cliente de los componentes del subsistema, y de esta forma se reduce el número de

objetos con los que el cliente trata, facilitando así el uso del subsistema.

- Su uso repercute en un bajo acoplamiento.
- 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.

2.9.6. Singleton.

Su misión es garantizar que una clase solo tenga una única instancia y proporcionar un punto de acceso global a ella.

Figura 9: Diagrama de clases del patrón Singleton.

Este patrón es recomendable usarlo cuando:

- Deba haber exactamente una instancia de una clase y esta deba 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 utilizar una instancia extendida sin modificar su código.

Dentro de sus ventajas están:

- Acceso controlado a la única instancia.
- Permite el refinamiento de operaciones y la representación. Se puede crear una subclase Singleton.
- Permite un número variable de instancias.

2.9.7. Patrón Data Access Object, (DAO).

Es un patrón de diseño directamente basado en la separación de responsabilidades, en el que se separa la persistencia de datos del resto de funcionalidades del sistema.

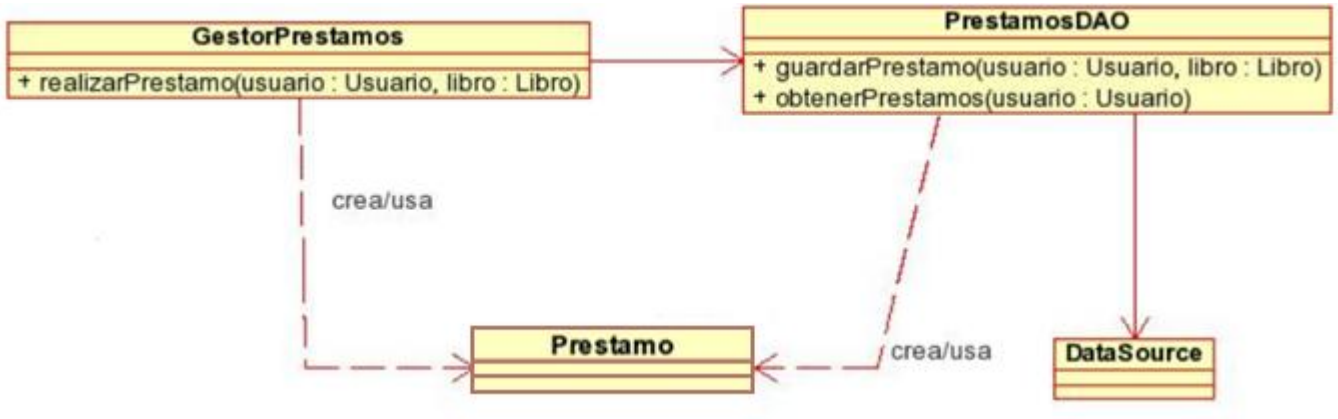

Figura 10: Diagrama de clases del patrón Data Access Object (DAO).

Este patrón 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.

Dentro de sus ventajas están:

- ✓ Proporciona la independencia del almacén de datos, debido a que el cambio de motor de base de datos o del mecanismo de almacenamiento 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 mecanismos de almacenamiento).

2.9.8. Patrones de Idioma.

Estos patrones se utilizan en los flujos de implementación, mantenimiento y despliegue, generalmente 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 predictibilidad.

A continuación, se describen instrucciones a seguir en el desarrollo y codificación del Sistema de Análisis Petrofísico.

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. Los subsecuentes componentes del nombre del paquete variarán.	com.sun.eng com.apple.quicktime.v2
Clases	Los nombres de las clases deben ser sustantivos. Cuando los nombres sean compuestos tendrán la primera letra de cada palabra que lo forma en mayúsculas y seguidos uno del otro, es decir sin la utilización del guión bajo "_" ni cualquier otro caracter. 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 Registro; class RegistroPozo;

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 en mayúscula.	ejecutar();
		
		ejecutarGraficas();

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 guión bajo "_" 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	int i; char c; float miGrafica;

	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.	
Constantes	Los nombres de las variables declaradas como constantes deben ir totalmente en mayúsculas separando las palabras con un guión bajo "_". (Las constantes ANSI se deben evitar, para facilitar su depuración.)2	static final int ANCHURA_MINIMA = 4; ANCHURA_MAXIMA=999;

2.10. Otras Herramientas a utilizar.

2.10.1. Entorno de Desarrollo Integrado (IDE).

IDE NetBeans 6.5.

El IDE NetBeans fue desarrollado por Sun Microsystems. Es una herramienta para escribir, compilar, depurar y ejecutar programas. Tiene todo lo necesario para la creación de aplicaciones profesionales para el escritorio con el lenguaje Java, C/C++, y Ruby. Es ejecutable en la mayoría de las plataformas como Windows, Solaris, Linux y Mac OSX. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

Se ha seleccionado para la implementación del sistema, el IDE NetBeans en su versión 6.5.

NetBeans 6.5 es integrable con sistemas de control de versiones como Subversion y CVS. Consta con soporte para el uso de frameworks como Spring, Swing e Hibernate. Además permite modelado con UML, ingeniería hacia adelante e inversa y generación de código.

2.10.2. JDK v 1.6.

Standard Edition 6 Java Development Kit (JDK v 1.6) es un producto de software que proporciona Sun Microsystems como parte de su Plataforma Java™, Standard Edition (Java™ SE). JDK v 1.6 es un software que provee las herramientas de desarrollo tales como compiladores y depuradores que son

necesarias o útiles para el desarrollo de applets y aplicaciones Java. Trae implícito el Java SE Runtime Environment (JRE) que proporciona las bibliotecas, Máquina Virtual Java (JVM), y otros componentes que necesita para ejecutar applets y aplicaciones escritas en el lenguaje de programación Java (40).

2.10.3. Subversion (SVN).

El control de versiones se basa en una serie de acciones más o menos estándares de comunicación entre la copia de trabajo y el repositorio. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Estas acciones son precisamente las que permite el cliente Subversion (SVN). SVN es un controlador de versiones empleado en la administración de archivos utilizados en el desarrollo de sistemas.

Subversion puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. A cierto nivel, la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomenta la colaboración.(41)

Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.

2.10.4. TortoiseSVN.

TortoiseSVN es un cliente gratuito de código abierto para Subversion. Maneja ficheros y directorios a lo largo del tiempo. Los ficheros se almacenan en un repositorio central. Está liberado bajo la licencia GNU GPL (42). Entra las características que presenta TortoiseSVN que lo hace un buen cliente para Subversion, se encuentran:

- Está implementado como una extensión del shell de Windows, haciéndolo perfectamente integrable al Explorador de Windows.
- El estado de cada carpeta y fichero versionado se indica por pequeños iconos sobreimpresionados. De esta forma, puede ver fácilmente el estado en el que se encuentra su copia de trabajo.
- Todos los comandos de Subversion están disponibles desde el menú contextual del explorador. TortoiseSVN añade su propio submenú allí.

Conclusiones Parciales.

En este capítulo se han definido las tecnologías y herramientas que serán utilizadas en el desarrollo del Sistema de Análisis Petrofísico.

La selección de las tecnologías se ha basado en las características fundamentales que presentan cada una ellas, y que las hacen ser cada vez más utilizadas en la industria del software. Además de las principales ventajas que ofrece su utilización en el desarrollo de aplicaciones informáticas.

Para la selección de las herramientas se ha tenido en cuenta la capacidad de cada una de ellas para proveer a los desarrolladores un ambiente de trabajo bien equipado. Se ha seguido como objetivo cubrir la mayoría de los procesos que se llevan a cabo dentro de la ingeniería de software, mediante herramientas que automaticen estos procesos.

Capítulo 3. *Línea Base de la Arquitectura*

Introducción.

En este capítulo, se hace una propuesta de solución al problema planteado en el diseño teórico de la investigación abordada en el Capítulo 1 de la presente investigación, tomando como base las descripciones de las tecnologías y herramientas expuestas en el Capitulo 2.

3.1. Organigrama de la Arquitectura.

La arquitectura de software como se ha explicado, es la estructura del sistema vista desde distintos niveles de abstracción, así como los elementos que la forman y las relaciones entre ellos.

Los estilos arquitectónicos son de hecho el nivel más alto de abstracción de la estructura de un sistema, y su elección implica en gran medida el tipo de sistema que se va a desarrollar.

El Sistema de Análisis Petrofísico entra en la categoría de Software de Ingeniería y Científico, utiliza algoritmos de manejo de números, análisis y representación de las propiedades. De este tipo sistema depende la evaluación de pozos petroleros y las futuras decisiones entorno a ellos, de ahí que debe poseer y ofrecer un alto grado de confiabilidad en cuanto a los resultados que proporciona.

En este sistema las funcionalidades están orientadas a la correcta lectura de los datos contenidos en el registro de pozo y a la visualización gráfica de todos los parámetros físicos extraídos de él. Además del posterior cálculo y análisis mediante algoritmos matemáticos, que permitan una correcta interpretación del registro.

La lógica de negocio de este tipo de sistemas no es muy variable, pero es bastante compleja. De ahí que se debe hacer una correcta separación de incumbencias en cada una de las capas tanto vertical como horizontalmente, con el objetivo de lograr un mayor modularidad del sistema y de esta forma facilitar la ubicación de posibles errores y posibilitar su reutilización en futuras versiones o sistemas similares.

A partir de los elementos mencionados la arquitectura del sistema se ha organizado de acuerdo con el estilo de arquitectura “Capas”.

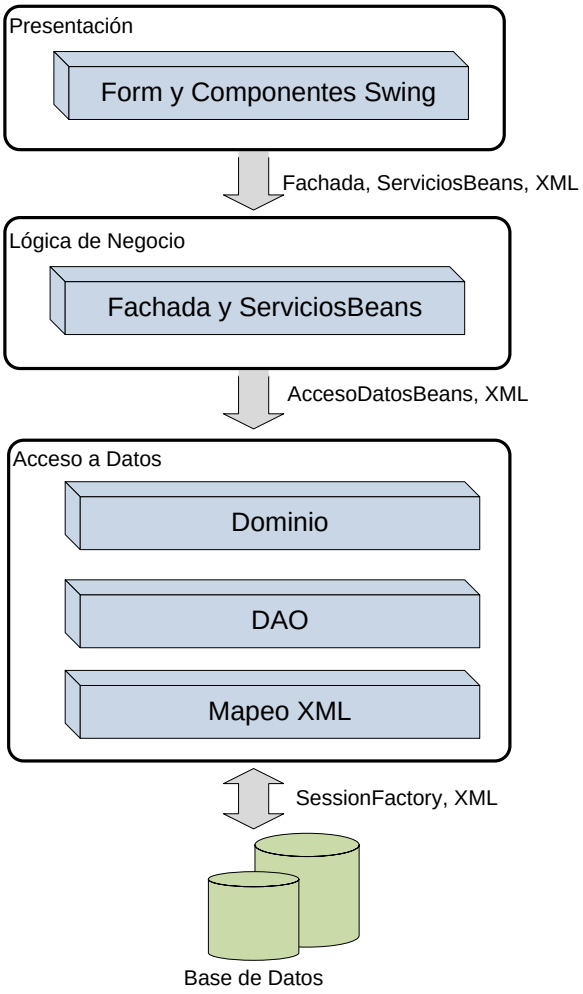

Figura 11: Arquitectura en Capas.

3.1.1. Componentes o Elementos.

Los principales elementos que distinguen esta arquitectura son:

- Presentación:

Esta capa contiene las interfaces necesarias para que el usuario y el sistema intercambien la información necesaria para el proceso de análisis de los registros, compuesta por formularios y componentes visuales desarrollados a partir del framework Swing.

- Lógica de Negocio:

Esta capa almacena todas las clases encargadas de representar la lógica de negocio. En la implementación de estas clases se manejan todas las transacciones de la aplicación y se aplican cada una de las reglas del negocio obtenidas a partir del análisis funcional del proyecto.

- Acceso a Datos:

Esta capa contiene la funcionalidad de acceder al repositorio de los datos y de ejecutar la lógica de acceso a datos. Es la encargada de persistir y acceder a los datos solicitados por la capa de negocio. Se ha dividido esta capa en tres subcapas, que contendrán los elementos que intervienen en el proceso de acceso a la base de datos.

1. Ficheros de Mapeo.

   Son fichero XML que contienen la información de la Base de Datos a la que hace referencia, contiene los campos de cada una de las tablas y sus relaciones.

2. Objetos de Negocio.

   Los Objetos de Negocio son clases entidades que contienen los atributos y funciones para acceder a ellos. Estos constituyen los objetos con los que funciona y trabaja el sistema.

3. Objetos de Acceso a Datos, (DAO).

   Los Objetos de Acceso a Datos son las clases que contienen las funcionalidades necesarias para acceder a la base de datos y efectuar un conjunto de procedimientos definidos para realizar las transacciones.

3.1.2. Conectores / Configuraciones.

Los conectores definen la forma de comunicación entre los componentes o elementos definidos en la arquitectura.

Presentación – Negocio, ServiceBeans:

Los Servicios que se crean en la capa de Lógica de Negocio son referenciados por las vistas en la capa de presentación mediante los Beans, que en el contexto de Spring son objetos. Las referencias a estos objetos pueden hacerse mediante el constructor o mediante una propiedad. De esta forma los Beans pueden acceder a las funcionalidades de los servicios referenciados.

Ejemplo:

```
<bean id="applicationFacade" class="CargarRegistro.ApplicationFacadeImp">
  <property name="applicationService" ref="applicationService" />
</bean>
```

En el ejemplo se hace referencia al bean *applicationFacade*, lo que quiere decir que, al invocar el constructor de la vista se está invocando al objeto del servicio. De esta forma quedan conectadas las dos capas.

El esquema utilizado para los archivos XML donde se referencian los Beans es el propuesto por el framework Spring.

Negocio – Acceso a Datos, ServiceBeans:

Los objetos de acceso a datos que se crean en la capa de Acceso a Datos son referenciados por los Beans que se crean en la capa de Negocio, específicamente las clases que prestan servicios. De igual forma estos pueden ser referenciados mediante el constructor o mediante una propiedad y así los ServiceBeans pueden acceder a las funcionalidades de los objetos de acceso a datos referenciados.

Ejemplo.

```
<bean id="applicationService" class="CargarRegistro.ApplicationServiceImp">
    <property name="dao" ref="applicationDao" />
</bean>
```

En el ejemplo se hace referencia al bean *applicationService*, lo que quiere decir que, al invocar el constructor del Servicio se está invocando al objeto del DAO. De esta forma quedan conectadas las dos capas.

Acceso a Datos – Base de Datos:

La conexión al repositorio de datos desde la capa de acceso a datos es proporcionado por el *SessionFactory*; este es un objeto (bean) del marco de trabajo Hibernate.

Ejemplo.

```
<hibernate-configuration>
    <session-factory>
        <!-- Configuracion de conexión a la base de datos -->
        <property name="connection.driver_class">org.h2.Driver</property>
        <property name="connection.url">jdbc:h2:file:/mibdH2;TRACE_LEVEL_FILE=3</property>
        <property name="connection.username">bdh2</property>
        <property name="connection.password">bdh2</property>
        <property name="connection.pool_size">1</property>
        <property name="dialect">org.hibernate.dialect.H2Dialect</property>
        <property name="current_session_context_class">thread</property>
        <property name="cache.provider_class">org.hibernate.cache.NoCacheProvider</property>
        <property name="show_sql">true</property>
        <property name="hbm2ddl.auto">validate</property>
        <!-- Direccion de los ficheros XML de mapeo -->
        <mapping resource="hibernate/Test.hbm.xml"/>
    </session-factory>
</hibernate-configuration>
```

Este bean es la fábrica de sesiones, es decir, las conexiones que se crean al repositorio de datos. Contiene los datos necesarios para realizar la conexión, los que son pasados como propiedad al bean. Por ejemplo: el driver que identifica el tipo de base de datos a la que se va conectar, el tipo de conexión, el modo en va a ser ejecutada, el nombre de la base datos y el usuario y la contraseña para conectarse.

Restricciones.

La principal restricción que se le impone a los componentes de esta arquitectura es:

- Los compontes que pertenecen a las capas superiores solo pueden hacer uso de los componentes de sus capas inmediatas inferiores o verticales, que se extiendan a su nivel.

La organización de los componentes en cada una de las capas es la principal función de este estilo arquitectónico. Se debe tener en cuenta una adecuada ubicación para cada componente según su función y sus responsabilidades.

3.2. Frameworks de Desarrollo.

A partir de la organización de la arquitectura según el estilo arquitectónico definido (Capas), se propone la utilización de tres frameworks, repartidos en las tres capas del sistema.

- **Capa de Presentación:** Framework de Aplicaciones Swing (JSR 296).
- **Capa de Lógica de Negocio:** Framework Spring 2.5.
- **Capa de Acceso a Datos:** Framework Hibernate 3.2.

Sus características y fundamentación se encuentran en el acápite 2.2.

3.3. Sistema Gestor de Base de Datos H2 Database Engine.

Como fue visto en el acápite 2.7, H2 es un SGBD embebido que se caracteriza principalmente por su velocidad y modos de compatibilidad con otros SGBD. Además de poder ser ejecutado y soportar distintos tipos de conexión.

A continuación se detalla cómo se establece la comunicación con la BD H2 de acuerdo al modo en que es ejecutada.

- **Modo Embebido.**

  En este modo la base de datos solo podrá ser accedida por la aplicación, la cual al finalizar su ejecución detendrá el servidor de BD. Este modo puede ser ejecutado de dos formas, con diferentes modos de conexión:

  - En Disco: La base de datos será creada y accedida por la aplicación desde el disco duro, en la ruta que le sea especificada.

    jdbc:h2:[file:][<path>]<nombreBD>
    jdbc:h2:~/test (para señalar a partir del directorio raíz)
    jdbc:h2:file:/data/ejemplo
    jdbc:h2:file:C:/data/ejemplo (solamente en Windows)

    Se recomienda usar cuando se deseen conservar los objetos manejados por el sistema

una vez terminada su ejecución.

- En Memoria: la base de datos se creará en memoria. Cuando la aplicación termine su ejecución, la base de datos se perderá.

  jdbc:h2:mem:<nombreBD>
  jdbc:h2:mem:test_mem

  Se recomienda usar solo para las pruebas de unidad mediante el framework JUnit. Durante las pruebas no es necesario guardar la base de datos en disco, por lo que se realizarán las operaciones sobre la base de datos con mayor rapidez. Aunque esto conlleva a un aumento del consumo de la memoria de la computadora.

- **Modo Servidor.**

  En modo servidor la base de datos puede ser accedida remotamente a través de la red, permitiendo ser utilizada por el sistema desde cualquier estación de trabajo en la red. El tipo de conexión se puede establecer por dos vías:

  - TCP/IP.

    jdbc:h2:tcp://<server>[:<port>]/[<path>]<nombreBD>
    jdbc:h2:tcp://localhost/~/test
    jdbc:h2:tcp://dbserv:8084/~/ejemplo

  - SSL/TLS.

    jdbc:h2:ssl://<server>[:<port>]/<nombreBD>
    jdbc:h2:ssl://secureserv:8085/~/simple

  Se recomienda usar este modo en el caso de futuras funcionalidades del sistema que permitan al usuario a través de la red acceder a su base de datos previamente ejecutada en este modo y poder realizar su trabajo. La configuración del servidor se realizaría mediante el sistema para comodidad del usuario.

3.4. Objetivos y Restricciones Arquitectónicas.

Existen algunos objetivos y restricciones que tienen un impacto significativo sobre la arquitectura. A continuación serán descritas.

1. **Requerimientos de Hardware.**

   Estaciones de Trabajo

   - Periféricos Teclado y Mouse PS2 o USB.
   - 1 MB de cache L2.
   - 512 MB o superior de memoria RAM.
   - 1 GB de espacio libre en disco.
   - CPU Pentium IV a 1.00 GHz o superior.

2. **Requerimientos de Software.**

   Estaciones de Trabajo

   - Sistema Operativo: Windows 2000/XP/Vista, Unix , GNU-Linux.
   - Maquina Virtual de Java: versión 1.6.

3. **Rendimiento.**

   - El sistema debe garantizar un tiempo de respuesta de 3 a 5 segundos en dependencia de la complejidad del servicio.
   - El sistema necesita un servidor de base de datos de 1Gb de RAM, garantizando rapidez en las consultas que se realicen para el acceso a los datos necesarios y de allí que el sistema de una respuesta rápida.

4. **Seguridad.**

   - El sistema garantizará que la información esté protegida de acceso no autorizado, divulgación, contra la corrupción y estados inconsistentes por las personas autorizadas.
   - La seguridad de la aplicación estará respaldada de la tecnología Java.

- Se aplicarán las reglas de la “programación segura”, mediante el tratamiento de excepciones.
- Los dispositivos o mecanismos utilizados para lograr la seguridad, no contribuirán a retrasar a los trabajadores de la entidad que interactúen con el sistema, permitiendo que a la hora de utilizarlo para sus necesidades se logre una mayor aceptación.
- El sistema permitirá además la verificación sobre acciones irreversibles; es decir, se le solicitará al usuario la confirmación al realizar operaciones como la eliminación.
- El sistema debe tener habilitada la protección contra fallos en caso de errores.
- La contenido de las bases de datos estará encriptado.

5. Reusabilidad.

- Las funcionalidades del sistema se desarrollarán en forma de componentes con el objetivo de la reutilización de estos en futuras versiones del sistema y en el desarrollo de nuevos sistemas y aplicaciones.
- La documentación de la ayuda y del sistema en general, deberá estar estructurada de tal manera que pueda ser reutilizada y permita la creación de una familia o línea de productos dentro del polo. El sistema debe permitir posteriores modificaciones y actualizaciones, así como modificar elementos ya existentes.
- El sistema estará dividida por módulos funcionales independientes.
- El sistema permitirá la integración de nuevas funcionalidades mediante la incorporación de componentes desarrollados por cualquier empresa.

6. Portabilidad.

- El sistema debe ser multiplataforma, siendo compatible con cualquier tipo de sistema operativo (SO GNU/Linux, SO Microsoft Windows).

7. Escalabilidad.

- El sistema se desarrollará bajo los estándares internacionales y el uso patrones, que permita la posterior integración al sistema de nuevas funcionalidades.
- El sistema permitirá la actualización de las funcionalidades con los que cuenta.
- El despliegue del sistema deberá hacerse con la menor cantidad gastos y ajustarse a los

recursos con los que cuenta la empresa.

8. Restricciones de acuerdo a la estrategia de diseño.

- El diseño de los componentes y el sistema en general se realizará bajo los principios y uso de la Programación Orientada a Objetos (POO).
- Se utilizará la tecnología implícita en los frameworks seleccionados para cada una de las capas del sistema.

    - Capa de Presentación: Framework de Aplicaciones Swing (JSR 296).
    - Capa de Negocio: Framework Spring 2.5.
    - Capa de Acceso a Datos: Framework Hibernate 3.2.

9. Herramientas de Desarrollo.

- Para el modelado con Visual Paradigm for UML es necesario 512 MB de memoria RAM. Recomendado 1 GB.
- Servidor de control de versiones Subversion; es necesario como mínimo 256 MB de memoria RAM, una capacidad de 30 GB de almacenamiento en disco.
- Para la implementación del sistema mediante el IDE NetBeans 6.5; es necesario 768 MB de memoria RAM. Recomendado 1 GB.
- Las librerías necesarias para la implementación serán:

    - JFreeChart, permite la construcción de gráficas en 2D y 3D.
    - Hibernate 3.2, framework.
    - H2Dialect.java.patch, es el parche para H2Dialect del framework Hibernate.

- Como SGBDR se utilizará H2.
- Para la creación visual de la base de datos H2 se utilizará Mozilla Firefox 1.5 o superior.

3.5. Estructura del Equipo de Desarrollo.

El equipo de trabajo está distribuido de la siguiente forma:

- Líder de proyecto.

- Arquitecto.
- Analista principal.
- Implementador.
- Implementador integrador.
- Diseñador de base de datos.

Los módulos que componen el sistema son:

- Cargar Archivo.
- Representación del Registro.

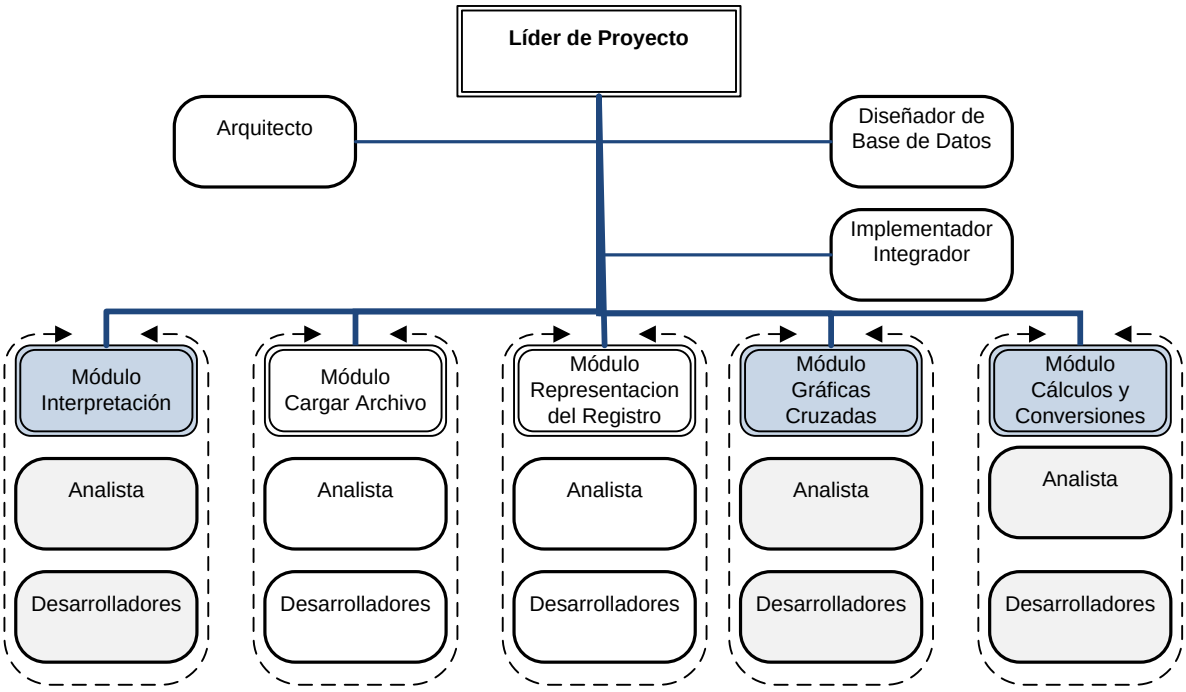

Figura 12: Estructura del equipo de desarrollo (incluye los módulos que se proponen para la segunda iteración del sistema).

Configuración de los puestos de trabajo por roles.

- Rol Analista.
    - PC con teclado y mouse.
    - Instalación de Visual Paradigm for UML 6.4.

- Instalación de un paquete ofimático (Office / OpenOffice).

- Rol Implementador.

  - PC con teclado y mouse.
  - Instalación del IDE NetBeans 6.5 y los plugins (JUnit, HibernateTool).
  - H2 Database Engine v 1.1.113.
  - Mozilla Firefox 3.0.
  - Instalación de JDK v.1.6.

- Rol Diseñador de base de datos.

  - PC con teclado y mouse.
  - Instalación de Visual Paradigm for UML 6.4.
  - H2 Database Engine v 1.1.113.
  - Mozilla Firefox 3.0.
  - Instalación de JDK v.1.6.

3.6. Representación de la Arquitectura.

La representación de la arquitectura se realizará mediante el marco de trabajo arquitectónico “4+1” vistas que propone la metodología RUP, compuesto por cinco vistas fundamentales.

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

Estas vistas ilustran los elementos arquitectónicamente significativos. Serán realizadas utilizando el Lenguaje de Modelado Unificado (UML) mediante la herramienta CASE Visual Paradigm for UML 6.4.

3.6.1. Vista de Casos de Uso.

Esta vista proporciona una base para planificar el contenido técnico de cada iteración, es decir ayuda a la selección del conjunto de casos de uso que son el centro de una iteración. Describe e ilustra el

conjunto de casos de uso que implican riesgos técnicos o de comportamiento arquitectónicamente significativos. Esta vista se perfecciona y contempla al inicio de cada iteración.

Los casos de uso del Sistema de Análisis Petrofísico y que serán descritos más adelante en esta sección son:

- Cargar Archivo.
- Seleccionar Plantilla.
- Gestionar Pistas.
- Editar Representación del Registro de Pozo por Criterios.
- Imprimir Representación del Registro de Pozo.

Estos casos de uso son iniciados por el actor Petrofísico.

Casos de Uso Arquitectónicamente Significativos.

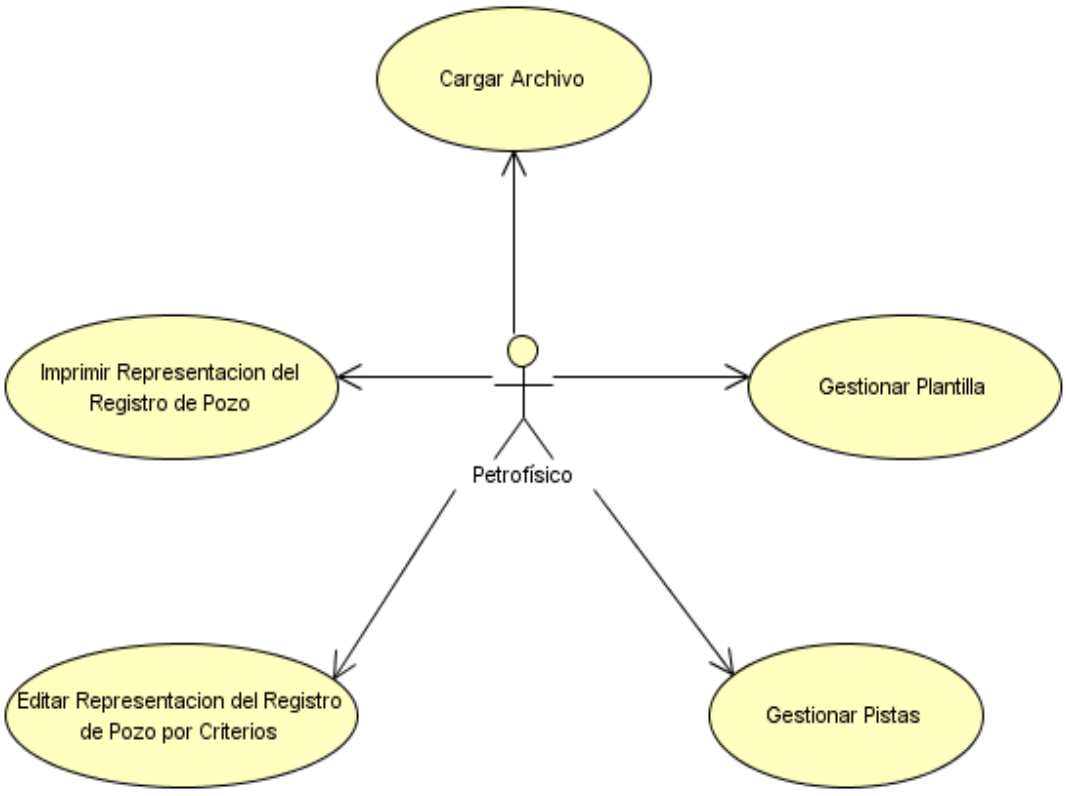

Figura 13: Vista de Casos de Uso.

Cargar Archivo.

Este caso de uso es el encargado de llevar al sistema toda la información contenida dentro de los registros de pozo. Estos pueden tener dos formatos: LAS y txt.

Seleccionar Plantilla.

Este caso de uso es el encargado de gestionar todo lo concerniente a las plantillas. Las plantillas contendrán todos los componentes visuales necesarios para la visualización de los datos del registro de pozo, una vez cargado en el sistema.

Gestionar Pistas.

Este caso de uso es el encargado de gestionar todo lo referente a las pistas. Las pistas están contenidas dentro de las plantilla y constituyen el principal componente visual para presentar información al usuario.

Editar Representación del Registro de Pozo por Criterios.

Permite al usuario modificar los parámetros de los componentes visuales que componen las plantillas, así como la forma en que son visualizados los datos del registro de pozo.

Imprimir Representación del Registro de Pozo.

Este caso de uso permite al usuario imprimir la visualización del registro de pozo.

3.6.2. Vista Lógica.

Esta vista describe las clases más importantes, su organización en paquetes de servicio y subsistemas, y la organización de estos subsistemas en capas. Se describen los paquetes de forma abstracta, así como las relaciones que existen entre ellos.

La figura ilustra la división del sistema en paquetes. Esto se realiza con el objetivo de proveer al sistema de una mayor reutilización y facilitar su mantenimiento.

La vista lógica del Sistema de Análisis Petrofísico está compuesta por tres paquetes principales:

- **Paquete Interfaz de Usuario:** contiene las clases para cada ventana, componente y formulario que el usuario usará para comunicarse con el sistema. Las interfaces soportará la carga de los registros de pozo, la interacción con la visualización de los registros y la impresión de esta visualización.
- **Paquete Lógica de Negocio:** contiene las clases para los servicios de negocio del sistema. Estas clases son las encargadas de recibir las peticiones de la capa de presentación, procesar la lógica de negocio relacionada con la carga, visualización y la impresión de los registros de pozo.
- **Paquete de Acceso a Datos:** contiene las clases entidades del negocio, así como los servicios de acceso a dato y las clases encargadas de la persistencia de los objetos. Los objetos persistentes identificados hasta el momento son: el objeto creado a partir de la lectura del registro de pozo y el objeto que contiene la visualización del registro.

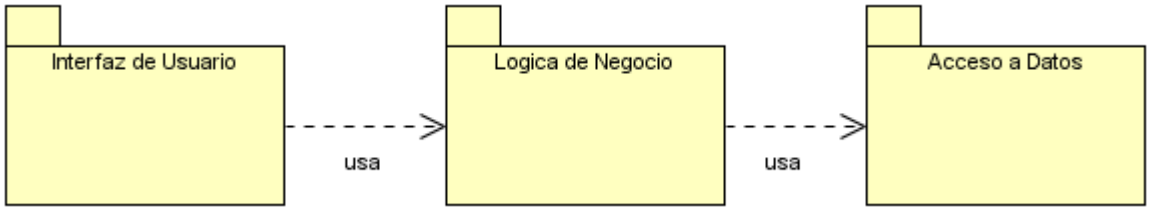

Figura 14: Distribución del Sistema en Capas.

Los casos de uso arquitectónicamente significativos se agruparon en los siguientes módulos:

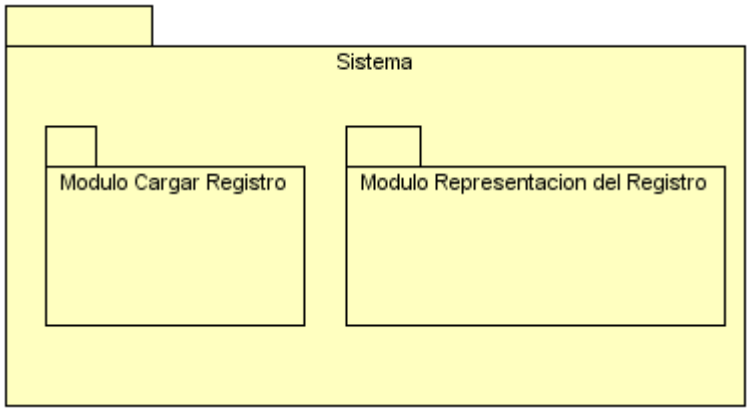

Figura 15: División del sistema en Módulos.

- **Módulo Cargar Registro:** este módulo contiene todo lo referente a la lectura e identificación de los datos del registro de pozo.

  Casos de uso contenidos:

  - Cargar Archivo.

- **Módulo Representación del Registro:** este módulo contiene todo lo referente a la visualización de los datos leídos del registro de pozo.

  Casos de uso contenidos:

  - Seleccionar Plantilla.
  - Gestionar Pistas.
  - Editar Representación del Registro de Pozo por Criterios.
  - Imprimir Representación del Registro de Pozo.

La distribución de la aplicación para cada uno de los módulos se hará de la siguiente forma:

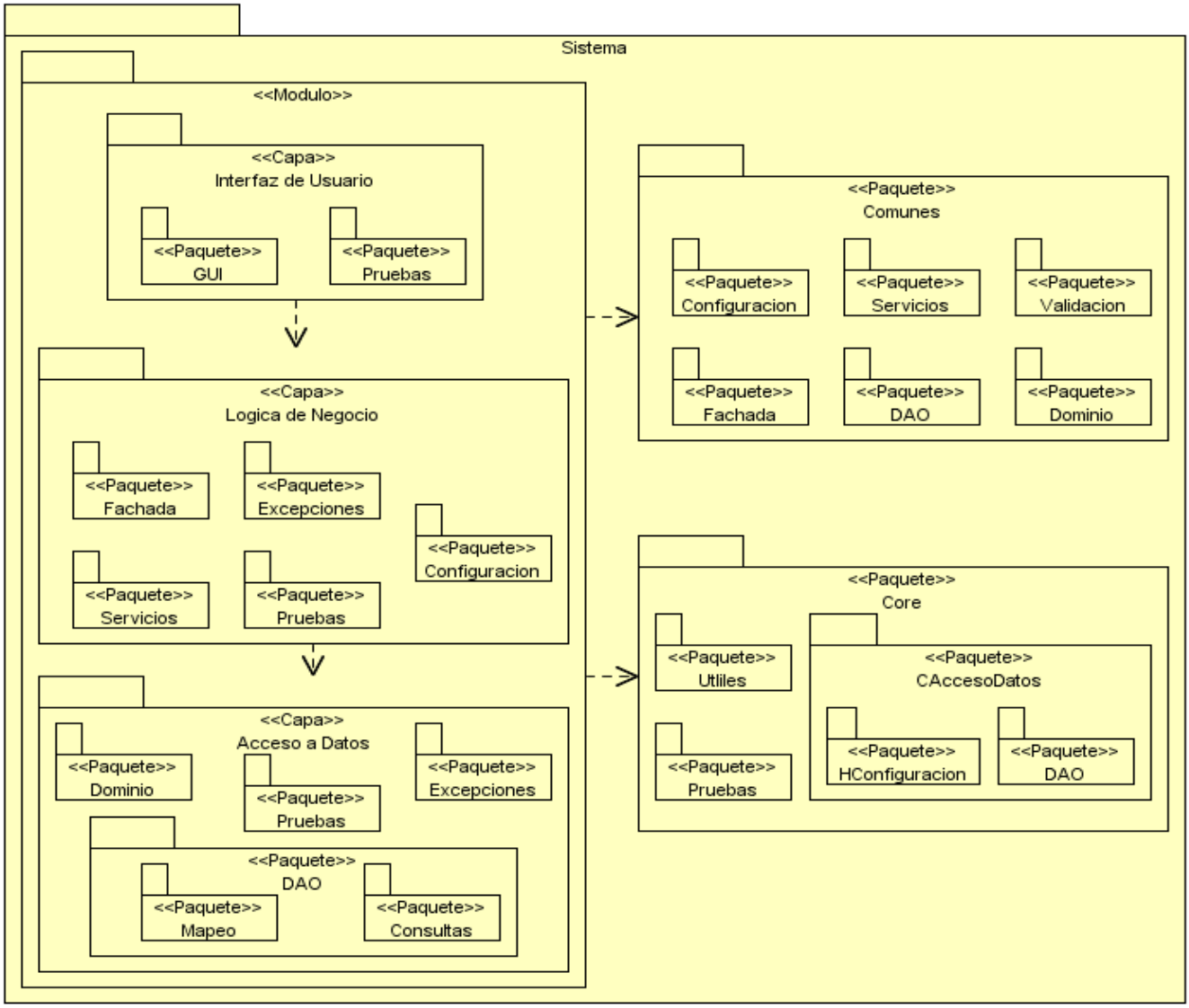

Figura 16: Distribución para cada módulo.

Paquete	Descripción
Interfaz de Usuario	
GUI	El paquete contiene las clases de interfaz gráficas de usuario que permiten a este intercambiar información con la aplicación.
Pruebas	El paquete contiene las clases para realizar las pruebas de unidad.
Lógica de Negocio	
Fachada	El paquete contiene las interfaces e implementaciones de los servicios de

	las fachadas específicas del módulo.
Configuracion	El paquete contiene los ficheros XML para la configuración de los Beans del módulo.
Pruebas	El paquete contiene las clases para realizar las pruebas de unidad para cada uno de los servicios implementados.
Servicios	El paquete contiene las interfaces que representan las operaciones de lógica de negocio que se van brindar o a consumir como servicios, además de las implementaciones de estas interfaces.
Excepciones	El paquete contiene las clases para el manejo de las excepciones de la capa.
Acceso a Datos	
Dominio	El paquete contiene las clases entidades persistentes del módulo.
Pruebas	El paquete contiene las clases para realizar las pruebas de unidad para cada uno de los DAO implementados.
Excepciones	El paquete contiene las clases para el manejo de las excepciones de la capa.
DAO	El paquete contiene las implementaciones de las interfaces DAO. Además contiene dos paquetes:
Mapeo	El paquete contiene los ficheros de mapeo de la base de datos específicos del módulo.
Consultas	El paquete contiene el fichero (XML) con las consultas a la base de datos en HQL.
Core	
Utilies	El paquete contiene un grupo de clases y paquetes que aseguran el correcto funcionamiento del sistema.
Pruebas	El paquete contiene las clases para realizar las pruebas de unidad del paquete Core.
CAccesoDatos	El paquete contiene los paquetes relacionados con el mecanismo genérico de acceso a datos. Los paquetes que contiene son:
HConfiguracion	El paquete contiene los *ApplicationContext* que instancian los objetos necesarios para proveer la conexión a la base de datos, así como el pool de conexiones y el administrador de transacciones (Transaction Manager).

DAO	El paquete contiene la interfaz genérica de acceso a datos *BaseDao*, donde se definen las operaciones básicas (Crear, Leer, Actualizar, Eliminar), que serán compartidas por todos los objetos de acceso a datos del sistema.
Comunes	
Configuracion	El paquete contiene los ficheros relacionados con la configuración general del sistema, los *ApplicationContext* del framework Spring, los ficheros de propiedades (.properties) del framework Hibernate y cualquier otro fichero de uso común en el sistema.
Fachada	El paquete contiene la declaración e implementaciones de las interfaces de las fachadas generales del negocio comunes en el sistema.
Servicios	El paquete contiene la declaración de la interfaz *BaseServicios*.
DAO	El paquete contiene la declaración de las interfaces DAO comunes en sistema,.
Validacion	El paquete contiene la interfaz y implementada por las clases encargadas de validar los datos de la capa de presentación.
Dominio	El paquete contiene las clases entidades persistentes de uso común en el sistema.

3.6.3. Vista de Implementación.

La vista de implementación proporciona una descripción de las principales capas y subsistemas de componentes de la aplicación.

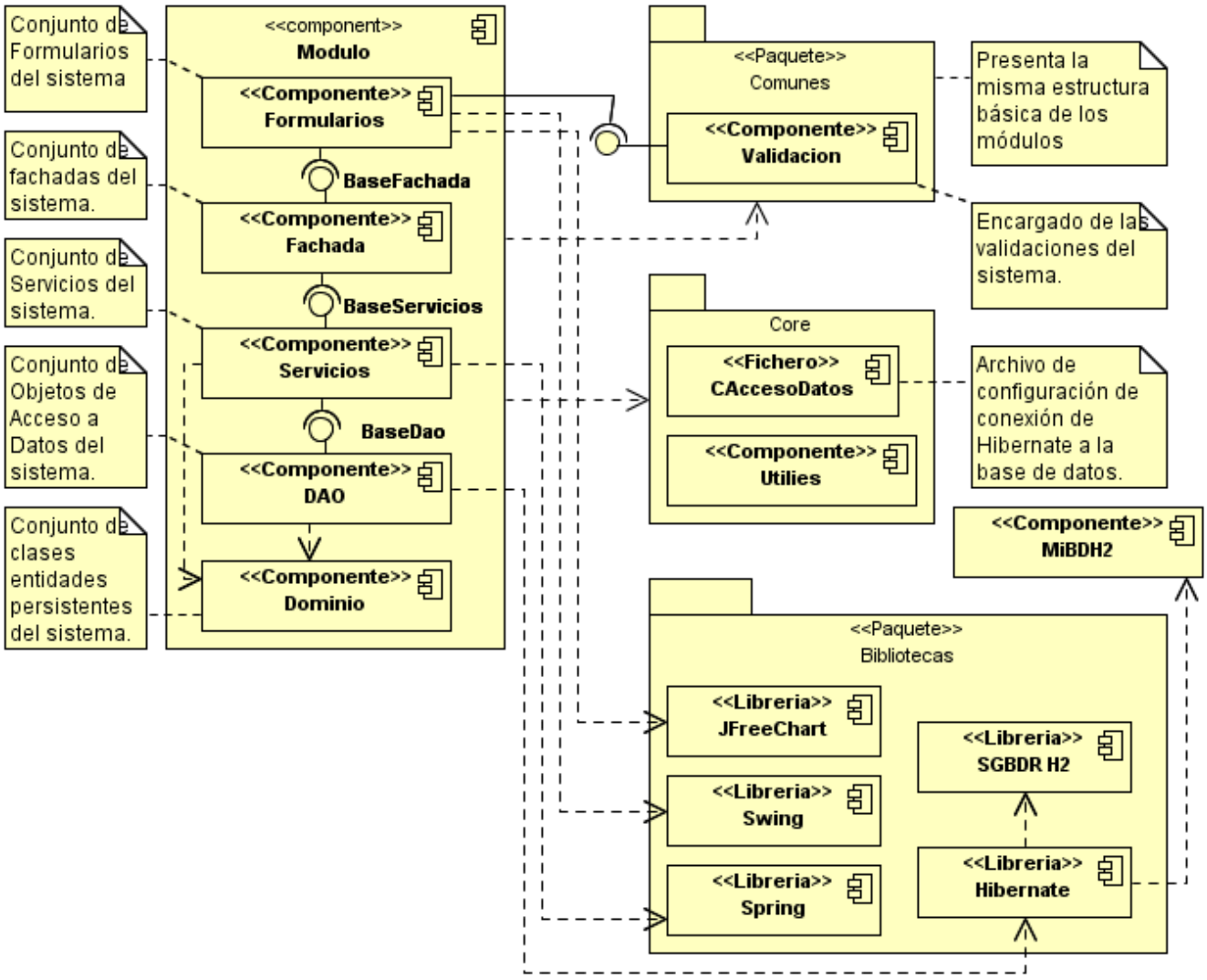

Figura 17: Vista de Implementación.

3.6.4. Vista de Despliegue.

La vista de despliegue propone la distribución física del sistema y cada uno de sus componentes, mediante la asignación de los componentes ejecutables a los nodos. Los nodos son elementos de la computadora sobre los cuales pueden ejecutarse los elementos de un sistema. Se satisface además parte de los requisitos no funcionales (hardware y software).

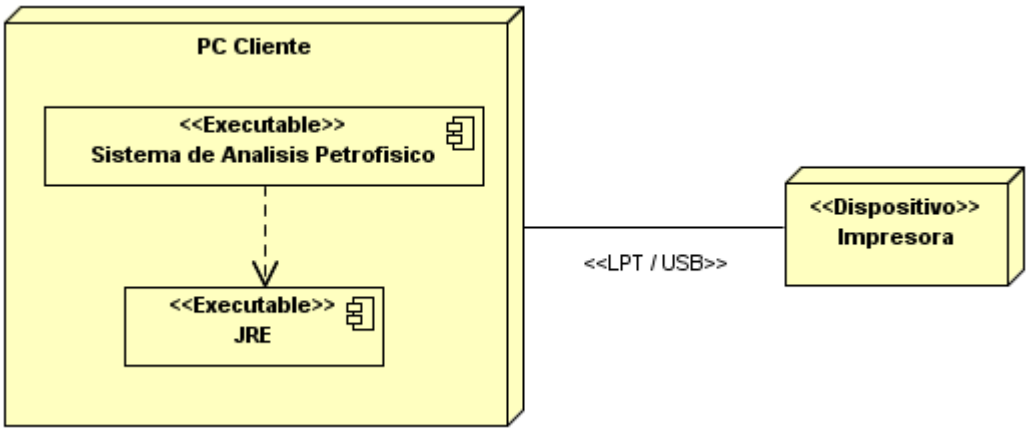

Figura 18: Vista de Despliegue.

- **Nodo PC Cliente:** Contiene el ejecutable del sistema y la Máquina Virtual de Java (JVM).
- El dispositivo Impresora satisface el requerimiento de impresión de la visualización del registro de pozo.

Conclusiones Parciales.

En este capítulo, se presenta la propuesta de arquitectura que intenta dar solución al problema planteado en esta investigación. Este capítulo es de obligada consulta y constante refinamiento por parte del arquitecto de software. Se exhibe la descripción arquitectónica del Sistema de Análisis Petrofísico, 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

En desarrollo de la presente investigación se hizo un estudio de los principales aspectos arquitectónicos. Así como la descripción de la arquitectura propuesta, basada en el estilo seleccionado, sus componentes, configuraciones y restricciones.

Se realizó además el estudio de los estilos arquitectónicos que más se ajustan a la solución del problema. Se fundamentaron los patrones de diseño que proporcionaran al sistema calidad, claridad y sencillez en la estructura y diseño de la solución, y facilitaran el mantenimiento del código y su reutilización.

Dada las restricciones impuestas por requisitos no funcionales del cliente y en función de los objetivos del Polo Productivo Petrosoft de la Facultad 9 de la UCI, se definieron las principales tecnología y herramientas a utilizar en el desarrollo del sistema. Así como la configuración de los puestos de trabajo y asignación de roles al equipo de desarrollo según lo establece la metodología RUP.

Se considera que la propuesta cumple con los requisitos que se necesitan para dar solución al problema planteado, es totalmente desarrollable por parte de los implementadores, puede ser aprobada y administrada por los arquitectos y jefe de proyecto.

Recomendaciones

Se recomienda al Polo Productivo Petrosoft y en especial a los líderes y arquitectos del área temática del Sistema de Análisis Petrofísico:

- Efectuar un continuo refinamiento de la arquitectura en cada iteración dentro del 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.
- Aplicar métodos de evaluación de la arquitectura, con el objetivo de identificar posibles debilidades, tanto en la descripción arquitectónica como en el propio diseño arquitectónico.
- Tener en cuenta la propuesta de arquitectura, para futuros desarrollos dentro del área temática y para formar parte de una Línea de Productos dentro del polo.

Referencias Bibliográficas

(1). PRESSMAN, R. S. *Ingeniería del Software: Un Enfoque Práctico.* 5ta ed. La Habana: Félix Varela, 2005. vol. I, 238 - 246 p.

(2). IEEE. *Standards Collection: Software Engineering.* 1993,

(3). SCHLUMBERGER. *SEED (Schlumberger Excellence in Educational Development)* [Consultado el: 24/03/2009 Disponible en: http://www.seed.slb.com/v2/FAQView.cfm?ID=914&Language=ES.

(4). CASTRO, O. Petrofísica. En *CEINPET. 12/01/2009 2009*.

(5). BROWN, T.; BURKE, T.*, et al.* Entrega de Datos a Tiempo. *Oilfield Review*, 2000, nº p. 22.

(6). PERRY, D. E. y WOLF, A. L. Foundations for the Study of Software Architecture. *Software Engineering Notes*, Octubre 1992, vol. 17, nº 4, p. 13.

(7). IEEE. *ANSI/IEEE Std 1471-2000, Recommended Practice for Architectural Description of Software-Intensive Systems* Disponible en: http://www.sei.cmu.edu/architecture/published_definitions.html#Modern.

(8). BASS, L.; CLEMENT, P.*, et al. Software Architecture in Practice.* 2da ed. Addison-Wesley, 2003, Disponible en: http://www.google.com.cu/books?id=mdiIu8Kk1WMC.

(9). GARLAN, D. y SHAW, M. *An Introduction to Software Architecture.* Enero 1994, Disponible en: http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf.

(10). REYNOSO, C. y KICCILLOF, N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004, Disponible en: www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

(11). CANAL, C. *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software.* Doctoral, Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga, 2000.

(12). RUMBAUGH, J.; JACOBSON, I.*, et al. El Lenguaje Unificado de Modelado. Manual de referencia.* Madrid: Pearson Educación, 2000, 526 p. Disponible en: http://bibliodoc.uci.cu/pdf/reg03050.pdf.

(13). LARMAN, C. *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* La Habana: Felix Varela, 2004. vol. I,

(14). CMU. *Software Ingineering Institute* Carnegie Mellon University, Disponible en: http://www.sei.cmu.edu/architecture/glossary.html.

(15). CAMACHO, E.; CARDESO, F.*, et al. Arquitectura de Software. Guía de Estudio.* Abril, 2004, p. 58 p.

(16). PÉRISSÉ, M. C. *Proyecto Informático.* Argentina, publicado el: 26/03/2009 de 2001, última actualización: 26/03/2009. Disponible en: http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/index.htm.

(17). LOUDEN, K. C. *Lenguajes de Programación: Principios y prática.* Cengage Learning Editores, publicado el: 26/03/2009 de 2004, última actualización: 26/03/2009. Disponible en: http://books.google.com.cu/books?id=MnrVc_GVKbMC&printsec=frontcover.

(18). LLAVORI, R. B. y QUEREDA, J. M. I. *Introducción a la programación con Pascal.* Universidad Jaume I, 2000, Disponible en: http://books.google.com.cu/books?id=OZFmZfOkz94C&printsec=frontcover.

(19). IGAC. *Sistemas de Gestión de Bases de Datos.* Instituto Geográfico Agustín Codazzi, [Consultado el: 22/05/2009 Disponible en: http://www.igac.gov.co:8080/igac_web/UserFiles/File/ciaf/TutorialSIG_2005_26_02/paginas/ctr_sistemasdegestiondebasededatos.htm.

(20). HYDROCARBON DATA SYSTEMS, I. *HDS 2000 Log Anslysis Software* [Consultado el: 12/03/2009 Disponible en: http://www.hds-log.com/.

(21). SENERGY. *Interactive Petrophysics*™ [Consultado el: 12/04/2009 Disponible en: http://www.senergyworld.com/iptm.

(22). LTD, C. D. P. P. *Powerlog: Advanced Log Analysis Software* [Consultado el: 15/03/2009 Disponible en: http://www.petrolog.net/about-us.asp.

(23). GARLAN, D. y SHAW, M. An Introduction to Software Architecture. *CMU Software Engineering Institute Technical Report*, 1994, nº Disponible en: http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf.

(24). JACOBSON, I.; BOOCH, G.*, et al. El Proceso Unificado de Desarrollo de Software.* La Habana: Felix Varela, 2004. vol. I.

(25). APEXNET. *Apexnet* [Consultado el: 28/03/2009 Disponible en: http://www.apexnet.com.ar/index.php/news/main/38/event=view.

(26). BOGGS, W. y BOGGS, M. *Mastering UML with Rational Rose 2002.* Editado por: Crossman, D. 2002, Disponible en: http://bibliodoc.uci.cu/pdf/0782140173.pdf.

(27). PARADIGM, V. *Visual Paradigm for the Unified Modeling Language:VP-UML 6.1 User's Guide* 6.1 ed. Visual Paradigm, 2007,

(28). BECERRIL, F. *Java a su alcance.* Editado por: Interamericana, M.-H. 1998, Disponible en: http://bibliodoc.uci.cu/pdf/reg01327.pdf.

(29). ARORA, G.; AIASWAMY, B.*, et al. C#.* Anaya Multimedia, publicado el: 30/03/2009 de 2002, última actualización: 30/03/2009. Disponible en: http://bibliodoc.uci.cu/pdf/reg01761.pdf.

(30). WIKIPEDIA. *C Sharp* [Consultado el: 25/03/2009 Disponible en: http://es.wikipedia.org/wiki/C_Sharp#Tipos_de_datos.

(31). AYALA, R. M. *XML.* Editado por: González, C. S. publicado el: 15/05/2009 de 2001, última actualización: 15/05/2009. Disponible en: http://bibliodoc.uci.cu/pdf/reg01501.pdf.

(32). RAY, E. T. *Learning XML.* O'Reilly, publicado el: 16/05/2009 de 2003, última actualización: 16/05/2009. Disponible en: http://www.google.com.cu/books?id=Zilck1_0c5QC.

(33). WIKIPEDIA. *SQLite* [Consultado el: 22/05/2009 Disponible en: http://es.wikipedia.org/wiki/SQLite.

(34). WIKIPEDIA. *Apache Derby* [Consultado el: 22/05/2009 Disponible en: http://es.wikipedia.org/wiki/Apache_Derby.

(35). MONTILVA, J. A. *Desarrollo de Software Basado en Líneas de Productos de Software.* Mérida, Venezuela: Dpto. de Computación, Fac. de Ingeniería de la Universidad de Los Andes, 2006, Disponible en: www.ieee.org.ar/downloads/2006-montilva-productos.pdf.

(36). MICROSYSTEMS, S. *JSR 296: Swing Application Framework.* (JSRs: Java Specification Requests). [Consultado el: 24/05/2009 Disponible en: http://jcp.org/en/jsr/detail?id=296.

(37). ALBIOL, F. R. *Introducción a Hibernate.* Septiembre 2003, Disponible en: www.froses.com/Assets/Files/Articles/Hibernate_Introduccion_es.pdf

(38). ORTEGA, Á. L. C. *Hibernate.* Universidad de Murcia: Febrero 2005, Disponible en: ditec.um.es/ssdd/trabajos/0506/Hibernate-Angel_Luis_Calvo_Ortega.pdf.

(39). WIKIPEDIA. *JUnit* [Consultado el: 22/05/2009 Disponible en: http://es.wikipedia.org/wiki/JUnit.

(40). SUN MICROSYSTEMS, I. *JavaTM Platform, Standard Edition 6: Overview* [Consultado el: 12/04/2009 Disponible en: http://java.sun.com/javase/6/docs/technotes/guides/index.html#platform.

(41). COLLINS-SUSSMAN, B.; FITZPATRICK, B. W.*, et al. Control de versiones con Subversion* [Consultado el: 10/04/2009 Disponible en: http://svnbook.red-bean.com/nightly/es/index.html.

(42). KÜNG, S.; ONKEN, L.*, et al. Un cliente de Subversion para Windows: Version 1.4.3.* 2006, Disponible en: https://forja.rediris.es/docman/view.php/123/117/TortoiseSVN-1.4.1-es.pdf.

Bibliografía

1. ÁVILA, S. J. V. *Introducción a Microsoft Solutions Framework.* 14/07/2005, Disponible en: http://www.mentores.net/articulos/intro_microsoft_sol_frame.htm.
2. CAMACHO, E.; CARDESO, F.*, et al. Arquitectura de Software. Guía de Estudio.* Abril, 2004, p. 58 p.
3. CANAL, C. *Un Lenguaje para la Especificación y Validación de Arquitecturas de Software.* Doctoral, Departamento de Lenguajes y Ciencias de la Computación. Universidad de Málaga, 2000.
4. H2. *H2 Database Engine*. 2009. [Consultado el: 22/05/2009, Disponible en: http://www.h2database.com/html/features.html#comparison.
5. HDS, *HDS Log Analysis Program: Instruction Manual*. 2001, Hydrocarbon Data Systems, Inc and Microsoft Corp.: Houston.
6. JACOBSON, I.; BOOCH, G.*, et al. El Proceso Unificado de Desarrollo de Software.* La Habana: Felix Varela, 2004. vol. I.
7. MONTILVA, J. A. *Desarrollo de Software Basado en Líneas de Productos de Software.* Mérida, Venezuela: Dpto. de Computación, Fac. de Ingeniería de la Universidad de Los Andes, 2006, Disponible en: www.ieee.org.ar/downloads/2006-montilva-productos.pdf.
8. MSDN. *El Patrón Singleton* Microsoft Corporation, [Consultado el: 22/05/2009, Disponible en: http://msdn.microsoft.com/es-es/library/bb972272.aspx.
9. ORTEGA, Á. L. C. *Hibernate.* Universidad de Murcia: Febrero 2005, Disponible en: ditec.um.es/ssdd/trabajos/0506/Hibernate-Angel_Luis_Calvo_Ortega.pdf.
10. PRESSMAN, R. S. *Ingeniería del Software: Un Enfoque Práctico.* 5ta ed. La Habana: Félix Varela, 2005. vol. I, 238 - 246 p.
11. REYNOSO, C. y KICCILLOF, N. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004, Disponible en: www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.
12. SCHLUMBERGER. *Schlumberger* [Consultado el: 24/03/2009 Disponible en: http://www.slb.com/content/services/software/geo/geoframe/petrophysics/index.asp

13. SCHMULLER, J. *Aprendiendo UML en 24 Horas.* México: Pearson Educacion, 2000, 248 p. Disponible en: http://bibliodoc.uci.cu/pdf/reg00004.pdf.

14. SENERGY. *Interactive Petrophysics*™ [Consultado el: 12/04/2009, Disponible en: http://www.senergyworld.com/iptm.

15. WIKIPEDIA. *Apache Derby* [Consultado el: 22/05/2009, Disponible en: http://es.wikipedia.org/wiki/Apache_Derby.

16. WIKIPEDIA. *GRASP* [Consultado el: 21/05/2009, Disponible en: http://es.wikipedia.org/wiki/Grasp#Indirecci.C3.B3n.

17. WIKIPEDIA. *H2* [Consultado el: 22/05/2009 Disponible en: http://es.wikipedia.org/wiki/H2.

18. WIKIPEDIA. *SQLite*. 2009. [Consultado el: 22/05/2009, Disponible en: http://es.wikipedia.org/wiki/SQLite.

Anexos

Anexo I. Tipos de Patrones.

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, rendimiento, 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.

Glosario de Términos

API's (Application Programming Interface): Conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción.

AOP (Aspect Oriented Programming): define un nuevo paradigma, la "Programación Orientada a Aspectos". La idea básica que persigue AOP es permitir que un programa sea construido describiendo cada concepto separadamente.

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

DAO (Data Access Object): es un patrón de diseño para abstraer y encapsular todo el acceso a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

Framework: es arquitectura definida para un determinado dominio de aplicación y contienen un número determinado de componentes implementados, con sus interfaces bien definidas que pueden ser reutilizados o redefinidos.

J2EE: es una plataforma para desarrollar aplicaciones empresariales distribuidas, utilizando el lenguaje de programación Java.

Lenguaje de Modelado Unificado (UML): es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

Metodología: En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo.

Petrofísica: es el área de estudio de las propiedades físicas y químicas que describen la incidencia y el comportamiento de las rocas, los sólidos y los fluidos.

POO (Programación Orientada a Objetos): es un paradigma de programación, que permite descomponer un problema en subgrupos relacionados. Cada subgrupo pasa a ser un objeto auto contenido que contiene sus propias instrucciones y datos que le relacionan con ese objeto.

Proceso de desarrollo de software: es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software.

Rational Unified Process (RUP): es un proceso de desarrollo de software.

XML: es un lenguaje basado en SGML, capaz de describir cualquier tipo de información en forma personalizada.