Universidad de las Ciencias Informáticas
Facultad 6

Título: Diseño de arquitectura estándar para los módulos del proyecto Mapeo Cerebral Humano de Cuba

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores: Dayana De La Mella Reus
Alex Tablada Alvarez

Tutor: Ing. Mónica Teresa Llorente Quesada

Junio de 2010

"Programar sin una arquitectura en mente es como explorar una gruta sólo con una linterna: no sabes dónde estás, dónde has estado, ni hacia dónde vas"

Danny Thorpe

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

Dayana de la Mella Reus

_____

Firma del Autor

Alex Tablada Alvarez

_____

Firma del Autor

Ing. Mónica Teresa Llorente Quesada

_____

Firma del Tutor

DATOS DE CONTACTO

Mónica Teresa Llorente Quesada: Ingeniero en Ciencias Informáticas. Profesor Docente con 3 años de experiencia.

Email: mllorente@uci.cu

AGRADECIMIENTOS

A nuestros padres por su apoyo incondicional durante toda la carrera, a nuestro fraternal amigo Danoy por haber estado a nuestro lado tanto en los momentos buenos como en los difíciles durante estos cinco años, a la profesora Liudmila y a nuestro compañero Francisco por la ayuda que nos brindaron en todo momento. Agradecer también a todos nuestros amigos de primer año, así como a nuestros compañeros de aula y de residencia, por todos los momentos intensos que hemos vivido juntos. Pero en especial le queremos agradecer a la Revolución y en particular a Fidel, ya que gracias a ellos pudimos estudiar y graduarnos en una universidad de excelencia.

DEDICATORIA

Dayana:

A mi abuela y abuelo que siempre han estado a mi lado apoyándome.

A mi mamá por su cariño y apoyo incondicional durante toda mi vida.

A mi papá por su amor y ayuda constante durante estos largos años de estudio.

Y a ambos por la confianza que siempre han depositado en mí, sin ellos no estaría aquí.

A mí querida hermana que siempre ha estado conmigo en todo momento ayudándome y queriéndome mucho.

A mi nene que es todo para mí y siempre ha estado a mi lado incondicionalmente en las buenas y malas, gracias por darme siempre lo mejor de ti.

A mi prima Ana Ivis por estar siempre pendiente de mí, te quiero mucho.

A todos mis amigos y compañeros de aula por siempre estar ahí cuando los necesito y por depositar en mí toda su confianza.

Alex:

A mi abuelita María Lola y mi abuelo Alberto, que donde quiera que estén, siempre los llevaré en mi corazón.

A mi mamá y a mi papá, que día a día me han dado todo el amor y el cariño del mundo, guiándome siempre por el buen camino y ayudándome a formarme en lo que soy hoy.

A mi hermano por inspirarme a superarme cada día, para tratar de ser siempre un ejemplo para él.

Al amor de mi vida, que además de ser mi amiga y compañera de tesis, ha estado todo el tiempo a mi lado, soportándome y brindándome todo su amor durante toda la carrera, y por haberse esforzado muchísimo en el desarrollo de la tesis.

A todos mis amigos que me han apoyado y han confiado en mí todo el tiempo y que juntos hemos vencido muchos obstáculos durante estos cinco años.

RESUMEN

El presente trabajo surge por la necesidad de lograr una estandarización de los módulos de la Plataforma Mapeo Cerebral Humano de Cuba, proyecto desarrollado de forma conjunta por el Centro de Neurociencias de Cuba (CNEURO) y la Universidad de las Ciencias Informáticas. Para ello se decidió diseñar una arquitectura que permita esta estandarización, proporcionándole los elementos necesarios para el desarrollo de una plataforma que le brinde a los especialistas de CNEURO las funcionalidades necesarias para el desarrollo de sus investigaciones.

Las decisiones arquitectónicas que fueron tomadas durante el desarrollo de la aplicación, la definición de las características fundamentales de las tecnologías y aspectos esenciales de diseño, la descripción de la arquitectura por medio de las vistas arquitectónicas, así como la realización de una evaluación de la arquitectura obteniendo resultados satisfactorios, además de tener en cuenta que las herramientas informáticas y tecnologías usadas en el desarrollo del sistema estuvieran bajo licencia libre; conllevan en gran medida al éxito de la aplicación.

PALABRAS CLAVES

Arquitectura de Software

Tecnologías

Vistas arquitectónicas

INDICE

INTRODUCCIÓN

Cuando se concluye el programa del genoma humano, que fuera en su tiempo un desafío colosal para la humanidad y hoy una gran hazaña, se presenta entonces para los científicos otro reto de alcance similar "El Proyecto del Mapeo Cerebral Humano". El hecho de hacer un mapa de cuáles son las diferentes funciones del cerebro que representa tanto la superficie cortical como los núcleos internos es en realidad un logro para la ciencia moderna. [1]

El Proyecto Mapeo Cerebral Humano de Cuba fue lanzado en el año 1993 con el objetivo fundamental de avanzar en el discernimiento e interpretación de la relación que existe entre la estructura y el funcionamiento del cerebro humano. [2]

Cuba a pesar de estar bloqueado económicamente se ha integrado al proyecto gracias a la experiencia de los especialistas del Centro de Neurociencias de Cuba (CNEURO) en el procesamiento de neuroimágenes. El proyecto cubano no solo ha introducido novedosas técnicas, sino también es el primero a nivel internacional en incorporar el mapeo de la actividad eléctrica cerebral, a través del Electroencefalograma (EEG). [3]

Esto colocará al país en el centro del desarrollo del proyecto de Mapeo Cerebral Humano Internacional que adquiere particular importancia en este Siglo del Cerebro; siendo el primero en combinar estudios electro-fisiológicos (con el MEDICID), anatómicos y de Resonancia Magnética.

El objetivo principal de este proyecto es crear una gigantesca base de datos que incluya toda la información del cerebro conocida hasta la fecha. Los científicos tratan de agrupar lo obtenido, organizarlo y crear así, una cartografía de las regiones del cerebro a todos los niveles: químicos, biológicos y su relación con las enfermedades. [1]

La Universidad de las Ciencias Informáticas y en particular la Facultad VI, cumpliendo con la vinculación estudio-trabajo como modelo de formación, desempeña una meritoria tarea en su colaboración con el Centro de Neurociencias de Cuba (CNEURO), en la creación del proyecto "Mapeo Cerebral Humano de Cuba". Contribuyendo así al incremento del desarrollo de la Bioinformática en el país.

En la actualidad del proyecto Mapeo Cerebral Humano de Cuba se ha concluido y liberado un primer módulo denominado Examen Clínico (EC), encargado de gestionar toda la información de los exámenes que se le realizan a los pacientes de CNEURO, y se continúa trabajando en otro módulo llamado

Imágenes de Resonancia Magnética (MRI), encargado del procesamiento de imágenes de forma distribuida.

El proyecto además integra Servidor de Base de Datos, Servidor FTP, dos Interfaces Web y la utilización de una plataforma distribuida.

Lograr que un sistema casi terminado tenga un desempeño superior, sea modificable, seguro o tolerante a fallas, es casi imposible si no se piensa desde su concepción, en la incorporación de esas cualidades conscientemente en la arquitectura del sistema.

Hasta el momento el proyecto no cuenta con una arquitectura que permita entender el sistema, organizar su desarrollo, plantear la reutilización del software y hacerlo evolucionar, en cuanto a usabilidad, eficiencia, restricciones económicas y tecnológicas; debido a que sus módulos trabajan con diferentes herramientas y tecnologías de desarrollo, así como con distintas librerías y lenguajes de programación que generan incompatibilidades en el código, además de no contar con un diseño uniforme de sus interfaces gráficas, restándole estética a la aplicación. Por lo tanto, esta problemática representa una limitante significativa tanto para el equipo de trabajo del proyecto, como para los investigadores que utilizan la aplicación.

Teniendo en cuenta lo analizado se plantea como **Problema Científico** de la investigación:

¿Cómo lograr que el Sistema Mapeo Cerebral Humano de Cuba obtenga un diseño arquitectónico estándar en sus componentes y módulos?

En correspondencia con el problema científico planteado se define como **objeto de estudio** de la investigación la Arquitectura de Software, cuyo **campo de acción** está enmarcado en la Arquitectura de Software para los módulos del sistema Mapeo Cerebral Humano de Cuba.

El **objetivo general** de la investigación es presentar un diseño de arquitectura que estandarice todos los componentes y módulos del Proyecto Mapeo Cerebral Humano. Desglosándose en los siguientes objetivos específicos:

1. Seleccionar estilos y patrones de arquitectura.
2. Diseñar las vistas del sistema.
3. Describir la arquitectura del sistema.

4. Evaluar el diseño arquitectónico propuesto.

Con el fin de dar cumplimiento a los objetivos de esta investigación, se plantean las siguientes **tareas de la investigación**:

1. Revisión de los estilos arquitectónicos existentes.
2. Revisión de los patrones de arquitectura existentes.
3. Identificación de los estilos arquitectónicos a utilizar, fundamentación de estos estilos, así como su aplicación.
4. Identificación de los patrones arquitectónicos a utilizar, fundamentación de estos patrones, así como su aplicación.
5. Definición y fundamentación de las herramientas a utilizar para el desarrollo de la plataforma.
6. Análisis y diseño de las vistas del sistema.
7. Representación del diseño arquitectónico propuesto.
8. Selección y aplicación de un método para evaluar el diseño arquitectónico propuesto.

El presente trabajo de diploma se estructura en tres capítulos:

Capítulo 1: Fundamentación Teórica.

En este capítulo se presentan diferentes conceptos de estilos y patrones arquitectónicos más usados actualmente a nivel mundial, así como su fundamentación y aplicación, además de lenguajes, tecnologías y herramientas existentes; haciendo una selección de aquellos que son usados para dar solución al problema científico.

Capítulo 2: Descripción de la arquitectura.

En este capítulo se realiza un diseño de las vistas arquitectónicas, así como la descripción de la arquitectura de los módulos de la plataforma.

Capítulo 3: Evaluación del diseño arquitectónico propuesto.

En este capítulo se presentan métodos para evaluar un diseño arquitectónico, seleccionando uno de ellos, a partir del cual se hace un análisis de la solución propuesta.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1. Introducción

En el presente capítulo se plantean los principales conceptos que se relacionan con el objeto de estudio. Se estudian puntos importantes de la Arquitectura de Software (AS), como son: el análisis de los estilos arquitectónicos y los patrones de arquitectura más utilizados, así como tecnologías, lenguajes de programación y herramientas a utilizar para el desarrollo del de la aplicación.

1.2. Arquitectura de Software

Dentro de las diferentes metodologías de desarrollo del software, se encuentra un elemento fundamental para el éxito del producto: la Arquitectura de Software. Ésta es considerada como una radiografía del sistema que se está desarrollando, lo suficientemente completa como para que todos los implicados en el desarrollo tengan una idea clara de qué es lo que se está construyendo.

El legendario Edsger Dijkstra, en sus tempranas inspiraciones propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera.

La AS quedó en estado de vida latente durante unos cuantos años, hasta comenzar su expansión explosiva con los manifiestos de Dewayne Perry de AT&T Bell Laboratories de New Jersey y Alexander Wolf de la Universidad de Colorado. Puede decirse que Perry y Wolf fundaron la disciplina. [4]

A pesar de que existen numerosas definiciones de arquitectura, de las cuales ninguna está respaldada unánimemente por la totalidad de los arquitectos, existe un acuerdo que se refiere a la estructura a grandes rasgos del sistema, estructura consistente en componentes y relaciones entre ellos.

En el año 2000 la *IEEE* hace la definición “oficial” de AS en su documento *IEEE* 1471, el cual plantea que:

“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”.

Esta definición, deja claro que la AS no se inscribe en ninguna metodología de desarrollo, sino que es en sí, una disciplina que se desarrolla durante todo el ciclo de desarrollo del software.

1.3. Estilos Arquitectónicos

Un estilo arquitectónico define una familia de sistemas de software en términos de su organización estructural. Representa los componentes y las relaciones entre ellos con las restricciones de su aplicación y las asociaciones y reglas de diseño para su construcción. Shaw y Garlan precisan además, que un estilo arquitectónico define un vocabulario de componentes y tipos de conectores. [6]

A continuación se exponen algunos de los principales estilos arquitectónicos más usados en la actualidad:

- **Estilos de flujo de datos:**

  Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Un ejemplar de la misma serían las arquitecturas de tubería-filtros.

  - **Arquitecturas en tubería y filtros:**

    El sistema tubería y filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. El supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente.

    Ventajas:

  - ✓ Es simple de entender e implementar. Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea.
  - ✓ Los filtros se pueden empaquetar, y hacer paralelos o distribuidos.

    Desventajas:

  - ✓ Eventualmente pueden llegar a requerirse buffers de tamaño indefinido, por ejemplo en las tuberías de clasificación de datos.
  - ✓ El estilo no es apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla.

- **Estilos centrados en datos:**

Esta familia de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.

- **Arquitecturas de Pizarra o Repositorio**

  En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él.

  Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados.

➢ **Estilos de Llamada y Retorno:**

Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

- **Modelo Vista Controlador (MVC):**

  Reconocido como estilo arquitectónico por Taylor y Medvidovic, aunque en ocasiones se le define más bien como un patrón de diseño.

  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:

✓ 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.

Tanto la vista como el controlador dependen del modelo, el cual no depende de las otras clases.

Ventajas:

- ✓ Soporte de vistas múltiples. Dado que la vista se halla separada del modelo y no hay dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.
- ✓ Menor acoplamiento, porque desacopla las vistas de los modelos y desacopla los modelos de la forma en que se muestran e ingresan los datos.
- ✓ Mayor cohesión debido a que cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).

  Desventajas:
- ✓ Costo de actualizaciones frecuentes. Desacoplar el modelo de la vista no significa que los desarrolladores del modelo puedan ignorar la naturaleza de las vistas.

Para el desarrollo del presente trabajo se tomó al Modelo Vista Controlador como estilo y patrón, ya que abarca ambas definiciones independientemente del nivel de abstracción donde se aplique. Además por ser Spring el framework utilizado para el desarrollo de la plataforma, el cual contiene un módulo que utiliza Modelo Vista Controlador (Spring MVC), por lo que se hace necesario guiarse por la arquitectura que éste plantea.

- **Arquitecturas en Capas**

  Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

  Ventajas:
- ✓ El estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- ✓ Proporciona amplia reutilización.

  Desventajas:
- ✓ Muchos problemas no admiten un buen mapeo en una estructura jerárquica.

- ✓ A veces es también extremadamente difícil encontrar el nivel de abstracción correcto.

- **Arquitecturas Orientadas a Objetos**

  Los componentes del estilo se basan en principios Orientados a Objetos 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.

  Ventajas:

- ✓ Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- ✓ Es posible descomponer problemas en colecciones de agentes en interacción.

  Desventajas:

- ✓ El principal problema del estilo se manifiesta en el hecho de que para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

- **Arquitecturas Basadas en Componentes**

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

  Ventajas:

- ✓ Permite alcanzar un mayor nivel de reutilización de software.
- ✓ Permite que las pruebas sean ejecutadas probando cada uno de los componentes antes de probar el conjunto completo de componentes ensamblados.
- ✓ Simplifica el mantenimiento del sistema, cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema.

  Desventajas:

- ✓ Las actualizaciones de los componentes adquiridos no están en manos de los desarrolladores del sistema.
- ✓ Si no existen los componentes, toca desarrollarlos y se puede perder mucho tiempo, así como que estos componentes pueden tener conflictos si de estos sale una nueva versión y no está estandarizado con lo que se ha desarrollado en la aplicación ensamblada.

- **Estilos de Código Móvil:**

  Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando.

  - **Arquitectura de Máquinas Virtuales**

    La arquitectura de máquinas virtuales se ha llamado también intérpretes basados en tablas. Todo intérprete involucra una máquina virtual implementada en software. Se puede decir que un intérprete incluye un seudo-programa a interpretar y una máquina de interpretación.

    El estilo comprende básicamente dos formas o sub-estilos, que se han llamado intérpretes y sistemas basados en reglas.

    Ventajas:

  - ✓ La principal ventaja es la portabilidad del código.
  - ✓ La generación de una aplicación requiere el conocimiento concreto de la plataforma en la que se va a ejecutar el código, pero sin embargo al utilizar una máquina virtual este problema desaparece.

    Desventajas:

  - ✓ No son tan rápidos como los lenguajes compilados, pero si son bastante más rápidos que los interpretados.

- **Estilos heterogéneos:**

  En este grupo se incluyen formas compuestas o que no se ajustan a la clasificación en las categorías anteriormente descritas.

  - **Sistemas de control de procesos**

    Los sistemas de control de procesos se caracterizan no sólo por los tipos de componentes, sino por las relaciones que mantienen entre ellos. El objetivo de un sistema de esta clase es mantener ciertos valores dentro de ciertos rangos especificados, llamados puntos fijos o valores de calibración; el caso más clásico es el de los termostatos.

    La ventaja señalada de este estilo radica en su elasticidad ante perturbaciones externas.

  - **Arquitecturas Basadas en Atributos**

    La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman. La intención de estos autores es asociar a la definición del estilo arquitectónico un framework de razonamiento (ya sea cuantitativo o cualitativo) basado en modelos de atributos

específicos. Además de especificar los habituales componentes y conectores, los estilos basados en atributos incluyen atributos de calidad específicos que declaran el comportamiento de los componentes en interacción.

- **Estilos *Peer-to-Peer*:**

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes.

- **Arquitecturas Basadas en Eventos**

Las arquitecturas basadas en eventos se han llamado también de invocación implícita. Se vinculan históricamente con sistemas basados en actores y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos.

Ventajas:

- ✓ Se optimiza el mantenimiento haciendo que procesos de negocios que no están relacionados sean independientes.
- ✓ Se puede agregar un componente registrándolo para los eventos del sistema; se pueden reemplazar componentes.

Desventajas:

- ✓ El estilo no permite construir respuestas complejas a funciones de negocios**.**
- ✓ Un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea.

- **Arquitecturas Orientadas a Servicios(SOA)**

SOA es una arquitectura que permite desglosar las aplicaciones en términos de servicios, resultando ser más flexibles y por tanto son más fáciles de modificar ante los cambios del negocio.

Ventajas:

- ✓ Al reutilizar los servicios hay mayor calidad y reducción de riesgos en las aplicaciones, pues estos servicios ya han sido probados y en muchos casos mejorados.

- ✓ El especialista encargado de llevar a cabo la tarea de programar se ve aliviado de la carga de trabajo, pues se evita implementar funciones que ya están realizadas.

  Desventajas:
- ✓ Dentro de los campos donde no se aconseja introducir SOA cabe mencionar aquellos donde frente a la flexibilidad se prefiera una centralización de la información, y ya se cuente con los programas y aplicaciones para ello.

- **Arquitecturas Basadas en Recursos**

  En síntesis muy apretada, podría decirse que la Arquitectura Basada en Recursos define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde las URLs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación.

1.4. Patrones de Software

La idea de consolidar la experiencia de profesionales en un ámbito del conocimiento humano mediante patrones que registren problemas y soluciones típicas de su dominio, ha sido explotada ampliamente en muchas disciplinas, en particular, en el proceso de desarrollo de software, y ha producido importantes innovaciones. [8]

Los patrones permiten y han permitido en diferentes áreas del conocimiento humano rehusar la esencia de la solución de un problema al enfrentar nuevos problemas similares. Es así que los patrones constituyen una especie de mecanismo de registro y concentración de experiencia. [8]

Según el arquitecto Christopher Alexander, "*Cada patrón describe un problema que ocurre una y otra vez, y describe la solución a ese problema, de tal manera que dicha solución pueda ser usada un millón de veces más, sin hacerlo necesariamente dos veces del mismo modo*".[9]

¿Por qué son útiles?

Ayudan a aliviar la complejidad del software en varias fases en el ciclo de vida. En las fases de análisis y diseño, pueden servir de guía en la selección de arquitecturas de software que han probado ser exitosas. En las fases de implementación y mantenimiento, ayudan a documentar las propiedades estratégicas de sistemas de software en un nivel de abstracción más alto que el código fuente. Capturan ideas y

soluciones obtenidas por experiencia y las hace accesible para los desarrolladores, analistas e ingenieros menos expertos o principiantes. [10]

Elementos esenciales en la descripción de un patrón:

Aunque existen múltiples formatos de presentación de un patrón, como mínimo debiera contener ciertos componentes esenciales: [10]

- **Nombre**
- **Intención**
- **Descripción del problema**
- **Solución**
- **Consecuencias**

Los patrones no son privativos del diseño de software. Existen en diferentes disciplinas. Y dentro del área del software, aparecen en diferentes enfoques: patrones de análisis, patrones de arquitectura, patrones de expresiones idiomáticas, patrones de diseño. Tampoco están confinados a la comunidad de orientación a objetos, existen para el modelado de datos, para bases de datos relacionales y otros. [10]

- ✓ **Patrones de análisis**: Usualmente específicos de aplicación.
- ✓ **Patrones de arquitectura**: Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos, problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, modularidad, acoplamiento.
- ✓ **Patrones de idioma**: Regulan la nomenclatura en la cual se escriben, se diseñan y desarrollan los sistemas.
- ✓ **Patrones de diseño:** Fueron construidos dado los problemas con la claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes.

1.4.1. Patrones arquitectónicos

Describen los principios fundamentales de la arquitectura de un sistema de software. Identifica los subsistemas, define sus responsabilidades y establece las reglas y guías para organizar las relaciones entre ellos.

- ✓ Ayudan a especificar la estructura fundamental de una aplicación.
- ✓ Cada actividad de desarrollo es gobernada por esta estructura.
- ✓ Cada patrón de arquitectura ayuda a conseguir una propiedad específica en el sistema global.

Catálogo de patrones arquitectónicos:

- **Tubería y filtros:** Provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante tubos.
- **Pizarra o repositorio**: Útil para sistemas en que no se conoce una solución o estrategia determinista. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial.
- **Capas:** Permite estructurar aplicaciones que se pueden descomponer en grupos de subtareas, donde cada grupo está en un determinado nivel de abstracción.
- **MVC** (Modelo-Vista-Controlador): Divide una aplicación interactiva en tres componentes: el Modelo que contiene la funcionalidad y los datos, la Vista que despliega la información al usuario y el Controlador que maneja la entrada y coordina la actividad de la Vista.

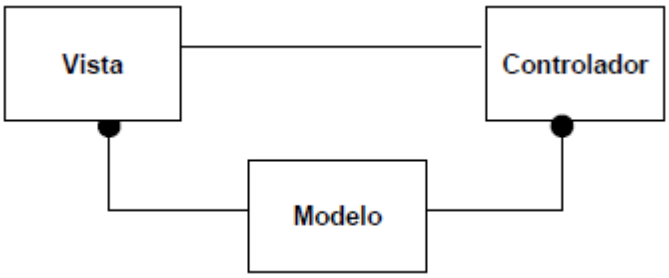

Figura1. Patrón Modelo Vista Controlador

- **PAC** (Presentation-Abstraction-Control): Define una estructura para software interactivo como una jerarquía de agentes cooperantes. Cada agente es responsable de un aspecto específico de funcionalidad de la aplicación y consiste en tres componentes: la Presentación, la Abstracción, y el Control. Esta subdivisión separa la interacción hombre-máquina de la funcionalidad y de la comunicación con otros agentes.
- **Broker**: Puede ser usado en sistemas distribuidos con clientes que interactúan por invocaciones a un servidor remoto con bajo nivel de acoplamiento. Un broker o corredor es responsable para coordinar la comunicación entre cliente y server.
- **Reflection:** Proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. La aplicación se divide en dos partes: un meta-nivel y un nivel-base. El meta-nivel hace al software auto consciente.

- **Microkernel:** Separa un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente.

1.4.2. Patrones de diseño

Los Patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan.

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular.

La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles y difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos. En los patrones GRASP (acrónimo que significa General Responsibility Asignment Software Patterns, patrones generales de software para asignar responsabilidades) se codifican algunos de ellos, que se aplican al preparar los diagramas de interacción, cuando se asignan las responsabilidades o durante ambas actividades.

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones.

A continuación se exponen las características de los 5 principales patrones GRASP: [11]

Experto

Problema: ¿Cuál es el principio fundamental para asignar las responsabilidades en el diseño orientado a objetos?

Solución: Asignar una responsabilidad al experto en información, es decir, a la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Explicación: Experto es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos.

Beneficios:

- 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 manteniendo. (Bajo Acoplamiento es un patrón GRASP que se examinará más adelante).

- 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 (patrón del que se abordará más adelante).

Creador

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

La creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilizabilidad.

Solución: Asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega los objetos de A.
- B contiene los objetos de A.
- B registra las instancias de los objetos de A.
- B utiliza específicamente los objetos de A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado (así que B es un Experto respecto a la creación de A).

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

Beneficios: Se brinda soporte a un bajo acoplamiento (se describirá más adelante), lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

Bajo acoplamiento

Problema: ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización?

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas.

Solución: Asignar una responsabilidad para mantener bajo acoplamiento.

Explicación: El Bajo Acoplamiento es un principio que debemos recordar durante las decisiones de diseño, es la meta principal que es preciso tener presente siempre. Es un patrón evaluativo que el diseñador aplica al juzgar sus decisiones de diseño.

El Bajo Acoplamiento soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad.

Beneficios

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

Alta cohesión

Problema: ¿Cómo mantener la complejidad dentro de límites manejables?

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

Solución: Asignar una responsabilidad de modo que la cohesión siga siendo alta.

Explicación: Como el patrón Bajo Acoplamiento, también Alta Cohesión es un principio que se debe tener presente en todas las decisiones de diseño, es la meta principal que ha de buscarse en todo momento. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos.

Beneficios:

- 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.

Controlador

Problema: ¿Quién debería encargarse de atender un evento del sistema?

Un evento del sistema es un evento de alto nivel generado por un actor externo, es un evento de entrada externa.

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones: controlador de fachada, controlador de tareas o controlador de casos de uso.

Explicación: La mayor parte de los sistemas reciben eventos de entrada externa, los cuales generalmente incluyen una interfaz gráfica para el usuario operado por una persona. En todos los casos, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada.

Beneficios:

- Mayor potencial de los componentes reutilizables.
- Reflexionar sobre el estado del caso de uso.

A continuación se listan patrones de diseño más habituales publicados en el libro "Design Patterns", escrito por los que comúnmente se le conoce como GoF (gang of four, por sus siglas en inglés) o pandilla de los cuatro:

Patrones de creación

- **Abstract Factory**: Proporciona una interfaz para crear familias de objetos que dependen entre sí, sin especificar sus clases concretas.
- **Builder:** Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.
- **Factory Method:** Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.
- **Singleton:** Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

Patrones estructurales

- **Adapter:** Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían, por tener interfaces incompatibles.
- **Bridge:** Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

- **Composite View:** Un objeto vista que está compuesto de otros objetos vista. Por ejemplo, una página JSP que incluye otras páginas JSP y HTML usando la directiva include o el action include.

- **Decorator:** Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- **Facade:** Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.
- **Flyweight:** Sirve para eliminar o reducir la redundancia cuando tenemos gran cantidad de objetos que contienen información idéntica.
- **Proxy:** Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Patrones de comportamiento

- **Command:** Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer las operaciones.
- **Interpreter:** Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- **Iterator:** Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.
- **Observer:** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.
- **State:** Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.

Como patrones de diseño se propone la utilización de los cinco patrones GRASP explicados anteriormente, debido a que estos patrones siempre deben estar presentes cuando se hace uso de la programación orientado a objetos, también se propone la utilización del Composite View y el Facade, como patrones GoF.

En el Capítulo 2 se especifica detalladamente la utilización de estos dos últimos patrones GoF.

1.5. Lenguaje Unificado de Modelado (UML)

UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema de software orientado a objetos. En el proceso de creación de UML han

participado empresas de gran peso en la industria como Microsoft, Hewlett- Packard, Oracle o IBM, así como grupos de analistas y desarrolladores. Esta notación ha sido ampliamente aceptada debido al prestigio de sus creadores y debido a que incorpora las principales ventajas de cada uno de los métodos particulares en los que se basa. [12]

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagramas, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo.

UML ha mejorado el desarrollo de software al establecer un estándar común que simplifica la comunicación entre desarrolladores de software. Sus principios fundamentales son fáciles de entender y de aprender. Es utilizado no sólo para la especificación de un sistema sino también para propósitos de comunicación entre los involucrados en el desarrollo de un sistema (ingenieros, científicos del área de computación, administradores, líderes y otros), o para la documentación de software existente.

Como resultado de las ventajas que ofrece UML, expuestas anteriormente, se seleccionó como lenguaje de modelado para describir la arquitectura, por poseer un propósito general y comprensible.

1.6. Metodología de desarrollo de software

La metodología de desarrollo es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a realizar nuevo software. Una metodología representa el camino para desarrollar software de una manera sistemática.

1.6.1. Rational Unified Process (RUP)

RUP es el resultado de varios años de desarrollo y uso práctico en el que se han unificado técnicas de desarrollo, a través del UML, y trabajo de muchas metodologías utilizadas por los clientes. Fue creado por Jacobson, Rumbaugh y Booch, este utiliza el UML como lenguaje de representación visual, es orientado a objetos, unifica los mejores elementos de metodologías anteriores y está preparado para desarrollar grandes y complejos proyectos.

Las principales características de **RUP** son:

- **Guiado por casos de uso**
- **Centrado en la arquitectura**

- **Iterativo e incremental**

La metodología **RUP** divide en 4 fases el desarrollo del software:

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

Cada una de estas etapas es desarrollada mediante un ciclo de iteraciones, el cual consiste en reproducir el ciclo de vida en cascada a menor escala.

En **RUP** se han agrupado las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales. Los 6 primeros son conocidos como flujos de ingeniería y los tres últimos como de apoyo.

Flujos de trabajo:

- **Modelamiento del negocio**
- **Requerimientos**
- **Análisis y diseño**
- **Implementación**
- **Prueba (Testeo)**
- **Instalación**
- **Administración del proyecto**
- **Administración de configuración y cambios**
- **Ambiente**

1.6.2. Proceso unificado abierto (OpenUp/Basic)

OpenUP/Basic es un proceso iterativo cuyas iteraciones se distribuyen a través de cuatro fases: Concepción, Elaboración, Construcción, y Transición.[13]

Cada fase podrá tener tantas iteraciones como se requiera dependiendo del grado de novedad del dominio de negocio, de la tecnología a ser utilizada, de la complejidad de la arquitectura de la solución y del tamaño del proyecto, entre otros factores. [13]

Para que los equipos de trabajo planeen sus iteraciones de forma rápida, el OpenUP/Basic provee plantillas con estructuras para la división del trabajo (EDT) en cada iteración y una plantilla de EDT para todo el proceso considerado de principio a fin. [13]

OpenUP/Basic es un proceso de desarrollo de software de código abierto diseñado para pequeños equipos organizados quienes quieren tomar una aproximación ágil del desarrollo.

OpenUP/Basic es un proceso iterativo que es Mínimo, Completo, y Extensible. Se valora la colaboración y el aporte de los stakeholders sobre los entregables. [14]

OpenUP se caracteriza por cuatro principios básicos que se soportan mutuamente:

- Colaboración para alinear los intereses y un entendimiento compartido.
- Balance para confrontar las prioridades (necesidades y costos técnicos) para maximizar el valor para los *stakeholders*.
- Enfoque en articular la arquitectura para facilitar la colaboración técnica, reducir los riesgos y minimizar excesos y trabajo extra.
- Evolución contínua para reducir riesgos, demostrar resultados y obtener retroalimentación de los clientes.

Está compuesto por los siguientes Flujos de Trabajo, los cuales se dividen en Desarrollo y Soporte.

Desarrollo: Requerimientos, Arquitectura, Implementación, Prueba.

Soporte: Gestión de Configuración, Gestión de proyecto.

Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por los casos de uso.

Los ingenieros de procesos de software pueden usar EPF Composer para extender y modificar OpenUP/Basic. Las modificaciones pueden ser tan simples como alterar las plantillas para los productos de trabajo o tan sofisticadas como adicionar actividades necesarias para crear software en su ambiente específico, así como auditoría para sistemas de seguridad crítica.

Beneficios en el uso del OpenUP:

- Ya que es apropiado para proyectos pequeños y de bajos recursos permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.
- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP.
- Por ser una metodología ágil tiene un enfoque centrado al cliente y con iteraciones cortas.

OpenUp fue la metodología seleccionada por ser un proceso ágil y ligero que promueve el desarrollo del software a través de las mejores prácticas, es extensible porque puede ser utilizada como base para agregar o adaptar artefactos según las necesidades, y es aplicable a un amplio sistema de plataformas y de usos del desarrollo. Además de su utilización para proyectos pequeños, y también teniendo en cuenta la decisión del polo de Bioinformática de utilizar esta metodología en sus proyectos.

1.6.3. Extreme Programing (XP)

XP es una de las metodologías de desarrollo de software utilizadas para proyectos de corto plazo, consiste en una programación rápida o extrema, es una metodología ágil centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo. XP se basa en realimentación contínua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios. Se define especialmente para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

1.6.4. Microsoft Solution Framework (MSF)

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

- Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
- Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.7. Herramientas CASE

CASE (Computer Aided Software Engineering) en su traducción al Español significa Ingeniería de Software Asistida por Computadora.

Se puede definir a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del ciclo de vida de desarrollo de un software. Los pasos en el ciclo de vida de desarrollo de un software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación.

1.7.1. Rational Rose

Es una herramienta de software para el Modelado Visual mediante UML de sistemas de software. Permite especificar, analizar y diseñar el sistema antes de codificarlo. Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Características: [15]

Desarrollo Iterativo: Rational Rose utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones.

Trabajo en Grupo: Rose permite que haya varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo.

Generador de Código: Se puede generar código en distintos lenguajes de programación a partir de un diseño en UML.

Ingeniería Inversa: Rational Rose proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

1.7.2. Visual Paradigm

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. [16]

Ofrece:

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

La herramienta CASE seleccionada fue Visual Paradigm, por su potencial para diseñar un producto con calidad y de forma rápida, además por contar la Universidad con la licencia para su uso, se considera la más adecuada para la modelación de las vistas de la arquitectura y para los demás artefactos generados en el proyecto.

1.8. Lenguaje de programación

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo.

La programación orientada a objetos (POO) es una de las formas más populares de programar y viene teniendo gran acogida en el desarrollo de proyectos de software desde los últimos años. Esta acogida se debe a sus grandes capacidades y ventajas frente a las antiguas formas de programar.

1.8.1. C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la European Computer Manufacturers Association (ECMA, por sus siglas en inglés) y la Organización Internacional para la Estandarización (ISO, por sus siglas en inglés).

Este lenguaje de programación combina los mejores elementos de múltiples lenguajes de amplia difusión como C++, Java, Visual Basic o Delphi. Su creador, Anders Heljsberg fue también el creador de muchos

otros lenguajes y entornos como Turbo Pascal, Delphi o Visual J++. La idea principal detrás del lenguaje es combinar la potencia de lenguajes como C++ con la sencillez de lenguajes como Visual Basic.

1.8.2. Java

Java es un lenguaje de programación relativamente joven, a pesar de ello, desde su aparición hasta el momento, su crecimiento ha sido vertiginoso, esto se debe en gran parte a la naturaleza misma del lenguaje que ha permitido a los creadores de Java añadir al lenguaje grupos de clases que pueden ser fácilmente adaptadas a las necesidades del desarrollador de software. A estos grupos de clases se les conoce comúnmente como *API*s, y los hay para diferentes propósitos que van desde la implementación de una interfaz gráfica para el usuario, como sería Swing Set, hasta la conectividad de bases de datos, como es el caso de *JDBC* (Java Data Base Conectivity).

Dentro de sus características principales se encuentran: [17]

Orientado a Objetos: Java soporta las características esenciales del paradigma de la programación orientada a objetos: encapsulación, herencia y polimorfismo. Java hace uso de la definición de entidades formadas por métodos y variables que reciben el nombre de clases, la instancia de alguna clase cualquiera en Java recibe el nombre de objeto.

Robusto: Java elimina el uso de apuntadores para referenciar localidades de memoria, Java además libera al desarrollador de la necesidad de desalojar la memoria que la aplicación ya no usa, aunado a esto Java requiere la declaración explícita tanto de los tipos de datos como de métodos. Es importante mencionar que Java verifica que no haya problemas tanto en tiempo de ejecución como en tiempo de compilación. Finalmente Java hace uso de excepciones para notificar al usuario acerca de las fallas que puedan encontrarse en el sistema y recuperar dicho sistema.

Independiente de plataforma: Java ofrece la posibilidad de que los archivos que son generados para una aplicación sean independientes de la plataforma, es decir, que se compilen una vez y se ejecuten en cualquier plataforma. Esto es posible gracias a que las aplicaciones hechas en Java generan archivos conocidos como bytecode. Estos archivos no corresponden a algún procesador o sistema operativo en particular, digamos Intel o Motorola, Unix o Windows, sino que al momento de ser ejecutados un intérprete propio de cada plataforma interpreta el bytecode al correspondiente sistema y procesador en el cual se está ejecutando. Cada plataforma (Macintosh, Windows, Linux, etc.) tiene su propio intérprete de Java, pero el archivo bytecode es el mismo para todas las plataformas.

Multitarea: A pesar de que las capacidades de multitarea que pueden ser implementadas en Java dependen en gran parte del sistema operativo en el cual se ejecuten, digamos Windows o Unix, dichas capacidades superan en gran medida a los entornos de flujo único (single-thread) que ofrecen otros lenguajes de programación. Al ser multitarea Java permite la ejecución concurrente de varios procesos ligeros o hilos de ejecución.

El lenguaje de programación que se seleccionó para el desarrollo de la aplicación fue Java, por ser orientado a objetos y por ser independiente de la plataforma, característica imprescindible que se requiere para la aplicación.

1.8.3. Java Server Page (JSP)

Java Server Pages TM (JSP) es un conjunto de tecnologías que permiten la generación dinámica de páginas web combinando código Java (scriptlets) con un lenguaje de marcas como *HTML* ó *XML*, para generar el contenido de la página.

Como parte de la familia de la tecnología Java, con JSP se pueden desarrollar aplicaciones web independientes de la plataforma. Una característica importante es que permite separar la interfaz del usuario de la generación del contenido dinámico, dando lugar a procesos de desarrollo más rápidos y eficientes.

La especificación JSP es el producto de una colaboración amplia de varias de las industrias líderes en el desarrollo de software, liderados por Sun Microsystems. Es conveniente resaltar, que la tecnología JSP es un componente clave de la plataforma Java 2 Enterprise Edition (J2EE) propuesta por Sun Microsystems.

En resumen, las tecnologías JSP y *Servlets* son una alternativa importante para la programación de web de contenido dinámico que permiten: [25]

- Independencia de la plataforma.
- Rendimiento mejorado.
- Separación de la lógica de la aplicación de la presentación de los datos.
- Uso de componentes (Java Beans).
- Facilidad de administración y uso.
- El respaldo importante de la tecnología sólida Java.

1.9. Entorno integrado de desarrollo

Un entorno de desarrollo integrado o IDE (acrónimo en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación, o bien puede utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI). Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

JBuilder

Es un entorno de desarrollo integrado para el lenguaje de programación Java en su totalidad y soporta los últimos estándares Java y proporciona una plataforma estable y universal para desarrollar aplicaciones. JBuilder permite un proceso de desarrollo más eficiente. Sus herramientas visuales y asistentes facilitan desarrollar aplicaciones rápidamente. Proporciona Enterprise Java Beans (EJB) y otros componentes de código probados y confiables que pueden ser reutilizados, de modo que los desarrolladores emplearán el tiempo más eficientemente, codificando instancias múltiples de funcionalidades similares únicamente una vez. Este IDE de desarrollo solo puede ser ejecutado sobre el sistema operativo Windows. [18]

NetBeans

El NetBeans es un IDE de código abierto escrito completamente en Java usando la plataforma NetBeans. Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles). Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE NetBeans.

El IDE NetBeans es un producto libre y gratuito sin restricciones de uso.

Eclipse

Eclipse es un entorno de desarrollo integrado de código abierto multiplataforma para integrar herramientas de desarrollo, con una arquitectura abierta y basada en *plug-ins*. Este entorno de desarrollo integrado ofrece, el control del editor de código, del compilador y del depurador desde una única interfaz de usuario.

Además, Eclipse da soporte a todo tipo de proyectos que abarcan todo el ciclo de vida del desarrollo de aplicaciones, incluyendo soporte para modelado.
Según el lenguaje de programación seleccionado se hará uso del Eclipse, por ser este IDE compatible con Java, multiplataforma, además conserva el registro de las versiones, o sea, genera y mantiene la documentación de cada etapa del proyecto, logrando una integración del entorno de desarrollo con Subversion.

1.10. Sistema de Control de Versiones

Un sistema de control de versiones es un sistema de gestión de archivos y directorios, cuya principal característica es que mantiene la historia de los cambios y modificaciones que se han realizado sobre ellos a lo largo del tiempo. De esta forma, el sistema es capaz de "recordar" las versiones antiguas de los datos, lo que permite examinar el histórico de cambios o recuperar versiones anteriores de un fichero, incluso aunque haya sido borrado.

Concurrentes (CVS)

CVS es un sistema de mantenimiento de código fuente (grupos de ficheros en general) extraordinariamente útil para grupos de desarrolladores que trabajan cooperativamente usando alguna clase de red. Permite a un grupo de desarrolladores trabajar y modificar concurrentemente ficheros organizados en proyectos, esto significa que dos o más personas pueden modificar un mismo fichero sin que se pierdan los trabajos de ninguna. Además, las operaciones más habituales son muy sencillas de usar.
Es una herramienta que facilita registrar todos los cambios efectuados sobre los archivos de un proyecto, permite recuperar versiones anteriores del código de un proyecto, proporciona conocer qué cambios se han efectuado sobre un archivo determinado, quién los ha realizado y cuando, admite gestionar los conflictos que pueden producirse en entornos en los que los desarrolladores se encuentran distribuidos geográficamente.

Subversion

Subversion, también conocido como SVN, es un sistema de control de versiones que se ha popularizado bastante, en especial dentro de la comunidad de desarrolladores de software libre. Está preparado para funcionar en red, y se distribuye bajo una licencia libre de tipo Apache.
SVN surge con la intención de sustituir y mejorar al conocido CVS (Concurrent Versions System). SVN mantiene las ideas fundamentales de CVS pero suple sus carencias y evita sus errores.
Las principales características de SVN y sus mejoras frente a CVS son:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- Mantiene versiones de los metadatos asociados a los directorios.
- Mantiene la historia de todas las operaciones de cada elemento de los documentos, incluyendo la copia, cambio de directorio o de nombre.
- Atomicidad de las actualizaciones.
  Posibilidad de elegir el protocolo de red. Además de un protocolo propio (svn), puede trabajar sobre http (o https) mediante las extensiones *WebDAV*.
- Soporte tanto de ficheros de texto como de binarios.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.

Se decidió usar el sistema de control de versiones Subversion, debido a los beneficios de esta herramienta al reducir las inconsistencias y agilizar las transferencias y actualizaciones de datos, ya que es necesario que los equipos tengan disponible y actualizada las informaciones que requieren.

1.11. Frameworks

Un framework, en el desarrollo de software es una estructura de soporte definida en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software, para ayudar a desarrollar y unir los diferentes componentes de un proyecto.
Los objetivos principales que persigue un framework son: acelerar el proceso de desarrollo, reutilizar código ya existente y promover buenas prácticas de desarrollo como el uso de patrones.

Struts

Es una herramienta de soporte para el desarrollo de aplicaciones Web bajo el patrón MVC bajo la plataforma J2EE (Java 2, Enterprise Edition). Struts se desarrollaba como parte del proyecto Jakarta de la Apache Software Fundation, pero actualmente es un proyecto independiente conocido como Apache Struts.
Struts permite reducir el tiempo de desarrollo. Su carácter de "software libre" y su compatibilidad con todas las plataformas en que Java Entreprise esté disponible, lo convierte en una herramienta altamente disponible.

JSF

Es un framework web J2EE de la familia de código fuente abierto, basado en componentes de interfaz de usuario con estado, que facilita y agiliza el desarrollo de aplicaciones web.
JSF es una tecnología que permite construir aplicaciones web que soportan diferentes dispositivos como clientes, por ejemplo, teléfonos celulares, agendas, etc.
JSF pretende normalizar y estandarizar el desarrollo de aplicaciones web. Hay que tener en cuenta que es posterior a Struts, y por lo tanto, se ha nutrido de la experiencia de éste, mejorando algunas de sus deficiencias.

Spring

Spring es un framework de código abierto, de desarrollo de aplicaciones para la plataforma Java. Por su diseño el framework ofrece mucha libertad a los desarrolladores en Java y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes en la industria. Además, permite configurar las clases en un archivo XML y definir en él las dependencias. Cuenta con plantillas de utilidades para Hibernate, Ibatis y JDBC, así como la integración con Struts, JSF y otros frameworks. [19]
En lugar de que Spring proponga su propio módulo ORM (Object-Relational Mapping), propone un módulo que soporta los frameworks ORM más populares del mercado, entre ellos Hibernate, una herramienta de mapeo *open source* muy popular, que utiliza su propio lenguaje de query llamada HQL.

Se decidió escoger Spring para el desarrollo de la aplicación, puesto que se considera un framework liviano, ya que no es una aplicación que requiere de muchos recursos para su ejecución. Además en uno

de los módulos de la aplicación se había comenzado a trabajar con este framework, por lo que cambiarlo representaría una pérdida de tiempo y esfuerzo por parte de los programadores.

Hibernate

Hibernate es una herramienta de mapeo objeto/relacional para ambientes Java. El término "mapeo objeto/relacional" (ORM por sus siglas en inglés) se refiere a esta técnica de "mapear" la representación de los datos desde un modelo de objetos hacia un modelo de datos relacional, con un esquema de base de datos basado en SQL. Realiza el mapeo entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java.

Hibernate no sólo se hace cargo del mapeo de clases Java a las tablas de una base de datos (y de los tipos Java a los tipos de la base de datos), sino que también provee utilidades para consulta y captura de datos, y puede reducir considerablemente el tiempo que, de otra manera, habría que invertir con el manejo manual de datos mediante SQL. [20]

Hibernate implementa un muy eficiente algoritmo de búsqueda. Puede ser utilizado tanto en la web como en aplicaciones convencionales; esto no solo brinda la inversión de tiempo en Hibernate de cara al futuro, sino que, de ser útil, hace a los desarrolladores totalmente independientes del mismo. La meta de Hibernate es aliviar al programador del 95% de las tareas más comunes relacionadas con persistencia. Hibernate puede ayudar a encapsular o eliminar código SQL que sea específico de un proveedor de BD, y ayudar en la tarea usual de traducir desde una representación tabular a un gráfico de objetos.

Hibernate crea una capa separada que se ocupa del acceso a datos con total independencia del gestor y la base de datos, dando la oportunidad de trabajar con varios gestores y bases de datos dentro de la misma aplicación sin que esto cree ningún conflicto en el modelo de objetos.

Por las características anteriormente expuestas se decide para el desarrollo de la plataforma el uso de los framework Hibernate.

1.12. Gestor de Base de Datos

Una base de datos (cuya abreviatura es BD) es una entidad en la cual se pueden almacenar datos de manera estructurada, con la menor redundancia posible. Los Sistemas de Gestión de Bases de Datos (SGBD) son software específicos que se sirven de interfaz entre la base de datos, el usuario y las

aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta. [21]

Oracle

Oracle es un sistema de gestión de base de datos relacional (o RDBMS por el acrónimo en inglés de Relational Data Base Management System), desarrollado por Oracle Corporation.

Se considera a Oracle como uno de los sistemas de bases de datos más completos, destacando su:

- Soporte de transacciones.
- Estabilidad.
- Escalabilidad.
- Soporte multiplataforma.

MySQL

MySQL es un sistema de administración de bases de datos relacional (RDBMS). Una base de datos relacional archiva datos en tablas separadas en vez de colocar todos los datos en un gran archivo, esto permite velocidad y flexibilidad. Las tablas están conectadas por relaciones definidas que hacen posible combinar datos de diferentes tablas sobre pedido.

Características:

- Amplio subconjunto del lenguaje SQL. Algunas extensiones son incluidas igualmente.
- Disponibilidad en gran cantidad de plataformas y sistemas.
- Diferentes opciones de almacenamiento según si se desea velocidad en las operaciones o el mayor número de operaciones disponibles.
- Transacciones y claves foráneas.
- Conectividad segura.
- Replicación.
- Búsqueda e indexación de campos de texto.

PostgreSQL

PostgreSQL es un servidor de base de datos objeto relacional libre, liberado bajo la licencia *BSD*.

Sigue actualmente un activo proceso de desarrollo a nivel mundial gracias a un equipo de desarrolladores y contribuidores de código abierto.

Características de PostgreSQL:

- ✓ Alta concurrencia.

  Alta concurrencia. Mediante un sistema denominado MVCC (Acceso concurrente multiversión, por sus siglas en inglés) PostgreSQL permite que mientras un proceso escribe en una tabla, otros accedan a la misma tabla sin necesidad de bloqueos [22]
- ✓ Amplia variedad de tipos nativos. Adicionalmente los usuarios pueden crear sus propios tipos de datos, los que pueden ser por completo indexables gracias a la infraestructura GiST de PostgreSQL. Algunos ejemplos son los tipos de datos GIS creados por el proyecto PostGIS. [22]
- ✓ Funciones. Las funciones permiten subir bloques de código que se ejecuten en el servidor. Estas funciones pueden escribirse en una variedad de lenguajes, algunos de los más importantes son *PL/pgSQL.*

  ,

Ventajas de PostgreSQL

- Instalación ilimitada.
- Soporta tamaños de filas y bases de datos ilimitados.
- Soporta herencia.
- Tiene soporte para *unicode*.
- Corre en casi todos los principales sistemas operativos: Linux, Unix, BSDs, Mac OS, Windows, etc.

PostgreSQL administra grandes cantidades de datos, por lo cual es considerado como una de las bases de datos de código abierto (Open Source) más avanzadas del mundo. Con el empleo de este gestor garantizamos integridad de los datos y la velocidad de acceso y consultas a la base de datos.

Según las características anteriormente expuestas, se ha decidido utilizar PostgreSQL como Sistema Gestor de Base de Datos, por la necesidad de este proyecto de manejar grandes volúmenes de datos, además posee una gran escalabilidad (característica que no implementa MySQL), y es capaz de ajustarse al número de CPUs y a la cantidad de memoria que posee el sistema de forma óptima, haciéndolo capaz de soportar una mayor cantidad de peticiones simultáneas.

1.13. Servidor de Aplicaciones

Un servidor de aplicaciones es un software que proporciona aplicaciones a los equipos o dispositivos cliente, por lo general a través de Internet y utilizando el protocolo http. Los servidores de aplicaciones se distinguen de los servidores web por el uso extensivo del contenido dinámico y por su frecuente integración con bases de datos.

Un servidor de aplicación maneja la mayoría de las transacciones relacionadas con la lógica y el acceso a los datos de la aplicación. [23]

WebSphere

WebSphere es un galardonado servidor de aplicaciones web conforme con J2EE. Puede utilizarse en entornos de clústeres, permite la conmutación por anomalía y tiene un alto nivel de escalabilidad. Dispone de asistentes que le ayudarán en la configuración y el mantenimiento de las aplicaciones, incluye agrupaciones de conexiones JDBC, integra un juego de herramientas XML y también incluye un mecanismo de registro cronológico integrado.

Los tres problemas principales de WebSphere son el coste, la complejidad y los requisitos de hardware. Por otro lado, WebSphere es un producto complejo que incluye varios entornos integrados.

Apache Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Tomcat implementa las especificaciones de los servlets y de Java Server Pages (JSP) de Sun Microsystems. Incluye un contenedor Web que permite servir páginas dinámicas.

Tomcat es gratis, es fácil de instalar, se ejecuta en máquinas más pequeñas y es compatible con las *API* más recientes de Java. Puede descargarse, instalarse y probarse en menos de una hora. Tomcat ocupa muy poco espacio, teniendo su código escrito en Java, con un tamaño total de apenas un megabyte, de modo que no es raro que se ejecute tan deprisa. Otra ventaja de Tomcat es que es muy fiable, su solidez se basa en que miles de desarrolladores contribuyen con código. Tomcat pone a disposición de todo el mundo las últimas actualizaciones de Java. [24]

Se seleccionó Tomcat como servidor de aplicaciones, puesto que consume menos recursos que otros servidores de aplicaciones, es gratis, fácil de instalar y en él puede "correr" Servlets de Java y JSP que son los lenguajes que se utilizarán para la confección de la aplicación.

1.14. Asistente matemático

Matlab

Matlab (Matrix Laboratory, “laboratorio de matrices”) es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio. Está disponible para las plataformas Unix y Windows. Es una poderosa herramienta para la resolución numérica de problemas. [10]

Es un excelente software matemático el cuál va orientado a matrices, las cuáles poseen más de 1000 funciones predefinidas dentro del mismo, listas para ser utilizadas.

Con Matlab se pueden realizar análisis estadísticos y de datos, así como el desarrollo de aplicaciones, diseñar sistemas de control y análisis, biología computacional, pruebas y medidas, proceso de imagen, modelado y análisis financiero, y una gran variedad de informes y conexión a bases de datos. [25]

1.15. Características del sistema

El sistema que se desea construir proporcionará al Centro de Neurociencias de Cuba, principal cliente y beneficiario de este proyecto, el almacenamiento, gestión y consulta de datos con fines estadísticos de análisis científico para los estudios que en el campo de la anatomía cerebral humana realizan en ese centro. Este sistema podría resultar beneficioso para otros centros de características similares en cualquier lugar del mundo, con especial énfasis en Latinoamérica. El sistema debe permitir a los usuarios almacenar, procesar y consultar datos relacionados con los estudios de neurociencias, es decir, un sistema que le facilite la gestión y conservación de los estudios que se le realizan a los sujetos. Toda la información se recogerá y guardará en forma de reportes mediante una aplicación web, la cual se podrá abrir desde cualquier parte del centro que se desee utilizar.

Permitirá al Centro de Neurociencias el control de todas las actividades (gestión de los diferentes tipos de pruebas y gestión del procesamiento de imágenes), lo cual supondrá un acceso más rápido y sencillo a los datos, gracias a interfaces gráficas sencillas y amigables. Además los datos accedidos estarán siempre actualizados, lo cual es un factor muy importante para poder llevar un control centralizado de los diferentes departamentos. También ayudará a que exista una mejor comunicación entre los doctores, matemáticos, científicos y físicos de los diferentes departamentos. Desarrollará herramientas que de manera automática obtengan las imágenes anatómicas, de difusión y el Electroencefalograma (EEG),

almacenados en la base de datos y realice el procesamiento según el estándar definido en el protocolo del Proyecto de Mapeo Cerebral Humano Cubano.

Descripción de los módulos

El módulo de Exámenes Clínicos recogerá toda la información de los pacientes, la misma se obtendrá mediante encuestas aplicadas, esta información se utilizará para hacer reportes de forma dinámica.

El módulo de Imágenes de Resonancia Magnética (MRI) se encargará de gestionar el procesamiento de las imágenes de resonancia magnética correspondientes a un sujeto del estudio Mapeo, de forma distribuida.

1.16. Conclusiones

En este capítulo se han abordado diferentes tipos de metodologías y tecnologías más utilizadas en el mundo de la informática con el objetivo de lograr una solución que permita una estandarización arquitectónica en los módulos de la plataforma Mapeo Cerebral Humano de Cuba. Después de haber realizado el estudio de cada uno de estos temas y teniendo en cuenta las características de la plataforma fue seleccionado dentro de la familia de estilos y patrones arquitectónicos, Modelo Vista Controlador, debido a las ventajas que proporciona en cuanto a la separación en tres clases diferentes: el modelo, la vista y el controlador, logrando una mayor independencia entre ellos.

Se propone aplicar los patrones de diseño GRASP y GoF, ya que el framework utilizado, Spring, facilita enormemente la implementación de estos patrones de diseño.

Se seleccionó como metodología de desarrollo OpenUP/Basic, para describir la arquitectura y como lenguaje de modelado se utiliza UML; así como Visual Paradigm como herramienta CASE, ya que además de ser multiplataforma, la universidad cuenta con su licencia.

Se escogió como lenguaje de programación Java, específicamente JSP para la realización de la vista del sistema, utilizando Eclipse como entorno de desarrollo y Subversion como sistema de control de versiones.

Se seleccionó como framework Hibernate para el acceso a datos y Spring para la lógica del negocio. El gestor de BD seleccionado fue PostgreSQL, como servidor de aplicaciones se escogió Tomcat, y Matlab como asistente matemático.

CAPÍTULO 2: DESCRIPCIÓN DE LA ARQUITECTURA

2.1. Introducción

En el presente capítulo se realiza la descripción del diseño de la arquitectura para la plataforma, así como la descripción de las distintas vistas arquitectónicas, estableciendo metas y restricciones que se deben cumplir en el sistema para su mejor funcionamiento.

2.2. Organización del sistema

En la Figura 2 se muestra la forma en que se propone organizar el sistema, para ofrecer una mejor idea y facilitar la comprensión de la arquitectura propuesta, siguiendo el diseño del patrón Modelo Vista Controlador, que propone el framework Spring.

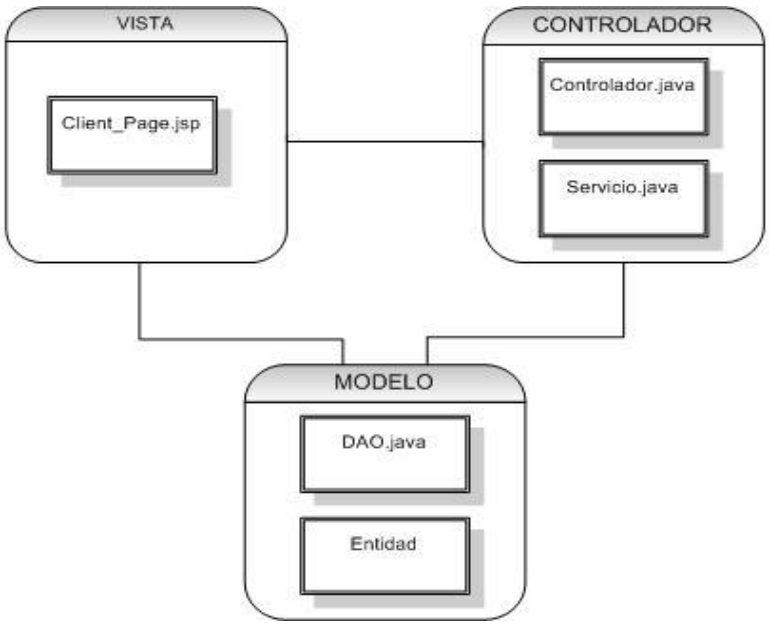

Figura 2. Organización del sistema

El sistema se encontrará estructurado de la siguiente manera:

La Vista tendrá contenida todas las clases que generan las interfaces gráficas de la aplicación. Ejemplo: EstudioList.jsp. Esta interface muestra un listado con los estudios realizados por los especialistas.

El Controlador contendrá las clases que se encargan de procesar la información necesaria para manejar y responder las solicitudes del usuario. Ejemplo: EstudiosServices.java, EstudiosListController.java.

El Modelo tendrá contenido los datos que proveen de información al usuario, así como las clases DAO, las cuales son las encargadas de acceder a los datos. Ejemplo: EstudiosDAOImpl.java, Estudio.hbm.
Los ejemplos fueron tomados de clases del módulo MRI.

2.3. Metas y restricciones arquitectónicas

Requerimientos no funcionales:

Son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. En muchos casos los requerimientos no funcionales son fundamentales en el éxito del producto. Normalmente están vinculados a requerimientos funcionales, es decir, una vez se conozca lo que el sistema debe hacer se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser.

De Software: Se deberá disponer para instalar la aplicación del Sistema Operativo Windows 98 o superior, o cualquier distribución de Linux; se debe disponer además de la máquina virtual de Java versión 1.5 o superior y como Asistente Matemático Matlab versión 7.0 o superior, para la instalación en los clientes. Mientras que se debe contar con PostgreSQL 8.3 y Apache Tomcat para los servidores de Base de Datos y de Aplicación Web respectivamente. Para el servidor de la aplicación el sistema operativo recomendado es GNU/Linux.

De Hardware: Se requiere disponer de una computadora personal Pentium IV con al menos 512 MB de memoria RAM y procesador de 3.0 GHz, para las máquinas clientes, mientras que los servidores no deben tener menos de 4.0 Gb de RAM y un procesador no menos de 3.0 GHz, con un mínimo de 160 Gb de espacio en disco. Conexión con un entorno de procesamiento distribuido *Grid*, para un mejor funcionamiento. Se deberá contar con impresora en las computadoras clientes que interactúen con la aplicación.

Apariencia o interfaz externa: La interfaz de manera general debe ser sencilla, permitiéndole al usuario ejecutar opciones del menú a través de combinaciones de teclas.

Soporte

Cuando el sistema sea instalado en el Centro de Neurociencia se impartirá un curso de adiestramiento y familiarización con el mismo. También se le dará seguimiento al funcionamiento de la aplicación.

Confiabilidad

El sistema debe ser confiable y preciso en la información que le suministra al usuario para evitar cualquier tipo de error.

Portabilidad

El sistema debe ser ejecutado sobre los sistemas operativos Linux y Windows, por su característica de ser multiplataforma.

Seguridad

Debe contar con protección contra acciones no autorizadas o que puedan afectar la integridad de los datos.

2.4. Vistas Arquitectónicas

Se propone para describir la arquitectura el Modelo 4+1 Vistas de Philippe Kruchten, el cual organiza la descripción de la arquitectura del software usando cinco vistas concurrentes, cada una de las cuales está dirigida a un conjunto específico de conceptos. Los arquitectos exponen sus decisiones de diseño en cuatro vistas y usan la quinta vista para ilustrar y validar dichas decisiones.

1. Vista lógica: muestra la estructura estática del sistema.
2. Vista de procesos: muestra los hilos y procesos de ejecución, así como la comunicación entre estos.
3. Vista de despliegue: muestra un mapeado del software sobre el hardware.
4. Vista de implementación: muestra la organización estática de módulos en el entorno de desarrollo.
5. Vista de casos de uso: contiene requisitos desarrollados en las restantes vistas.

2.4.1. Vista de Casos de Uso

Es usada para definir los requerimientos funcionales y la visión que los usuarios del negocio tienen de la aplicación mediante la representación de los actores y casos de usos más importantes. Esta vista muestra los subsistemas y módulos en los que se divide la aplicación y la funcionalidad que brinda dentro de cada uno de ellos.

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

Caso de Uso (CU): Representa una secuencia de acciones con un orden lógico y que producen un resultado observable para ciertos actores.

El sistema cuenta con tres actores: el Especialista, el Administrador y el Reloj del Sistema. A continuación serán descritos:

- El Especialista, representa el usuario que hará uso del sistema, teniendo la posibilidad de interactuar con todas las funcionalidades de éste.
- El Administrador, es también un Especialista, se encarga además de realizar funciones técnicas del sistema.
- El Reloj del Sistema, es el encargado de cada cierto tiempo, activar el proceso para buscar nuevos estudios en el servidor FTP.

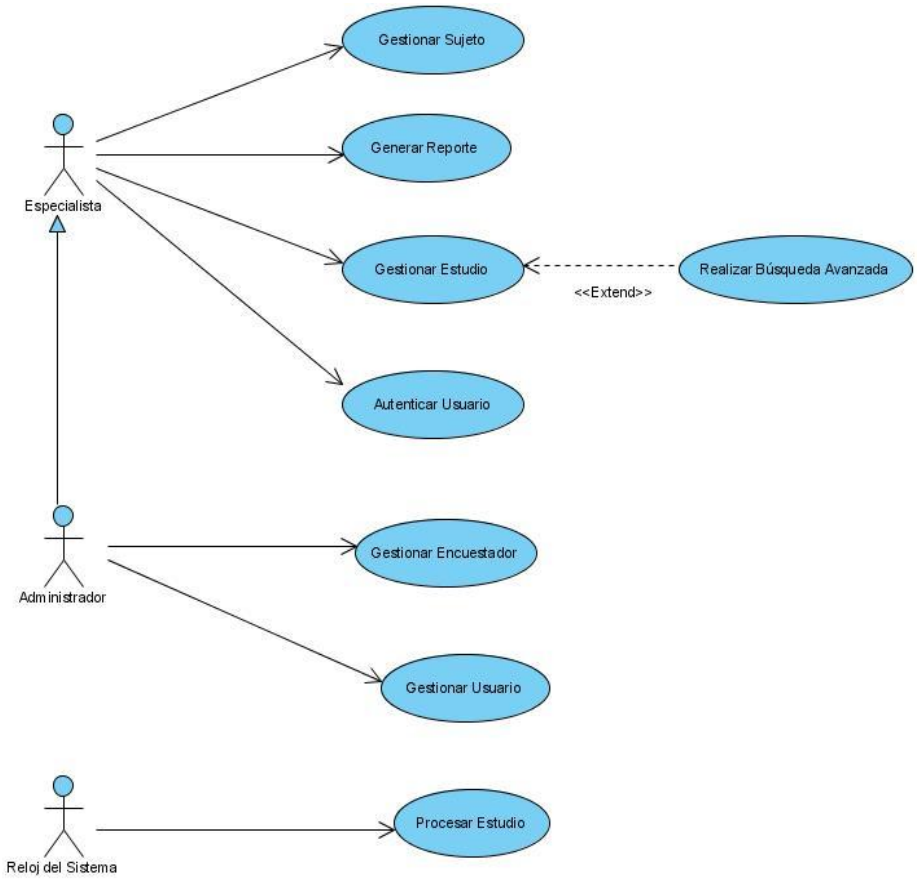

Figura 3. Vista de casos de uso

A continuación se describen los casos de uso que se muestran en la Figura 3.

Gestionar Sujeto: Con este caso de uso el especialista **l**leva a cabo una serie de operaciones con los sujetos presentes en el estudio como son: Insertar Sujeto, Buscar Sujeto, Visualizar Sujeto y Modificar

Sujeto. El sistema muestra la interfaz correspondiente según la solicitud hecha por el usuario y ejecuta las acciones necesarias.

Generar Reporte: Este caso de uso le muestra al especialista todos los reportes acerca de las acciones y resultados alcanzados hasta el momento en el estudio.

Gestionar Estudio: Con este caso de uso el especialista lleva a cabo una serie de operaciones como son: Visualizar cantidad de estudios realizados, Mostrar listado de estudios realizados. El sistema muestra la interfaz correspondiente según la solicitud hecha por el usuario y ejecuta las acciones necesarias.

Realizar Búsqueda Avanzada: Con este caso de uso el especialista podrá realizar la búsqueda de imágenes teniendo en cuenta determinados parámetros como: raza, sexo, edad, manualidad, etc.

Autenticar Usuario: Este caso de uso le permitirá al especialista autenticarse con su usuario y contraseña para entrar en la aplicación a realizar cierta operación, los datos introducidos por el usuario son verificados por el sistema, habilitándole la entrada si es un usuario autorizado y otorgándole los permisos según sus roles.

Gestionar Encuestador: Con este caso de uso el administrador **l**leva a cabo una serie de operaciones con los encuestadores del centro como son: Insertar Encuestador, Modificar Encuestador y Eliminar Encuestador. El sistema le muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias.

Gestionar Usuario: Con este caso de uso el administrador **l**leva a cabo una serie de operaciones con los usuarios del centro como son: Insertar Usuario, Visualizar Usuario, Modificar Usuario y Eliminar Usuario.
El sistema muestra la interfaz correspondiente según la solicitud hecha por el usuario y ejecuta las acciones necesarias.

Procesar Estudio: Con este caso de uso el Reloj del Sistema busca nuevos estudios adicionados al sistema sin haber sido procesados. El sistema solicita el procesamiento de las imágenes procedentes de los estudios, actualizando la información resultante.

2.4.2. Vista Lógica

Esta vista representa un subconjunto del artefacto Modelo de Diseño, representando los elementos de diseño más importantes para la arquitectura del sistema, se describen las clases más importantes, su organización en paquetes y subsistemas.

Paquetes de Diseño: Son usados para estructurar el modelo de diseño dividiéndolo en partes más pequeñas. Los paquetes de diseño deben usarse fundamentalmente como herramienta organizacional del modelo para agrupar elementos relacionados.

Subsistema de Diseño: Los subsistemas de Diseño son una forma de organizar los artefactos del modelo de diseño en piezas más manejables. Puede contener clases del diseño, realizaciones de casos de uso, interfaces y otros subsistemas. Pueden proporcionar interfaces que representan la funcionalidad que exportan en términos de operaciones.

Un Subsistema de Diseño cumple la misma función que un paquete, pero tiene la diferencia que aquí el Todo (subsistema) es mayor que la simple suma de las partes que lo componen, ya que éstas se encuentran interrelacionadas de forma tal que puedan satisfacer ciertos servicios,

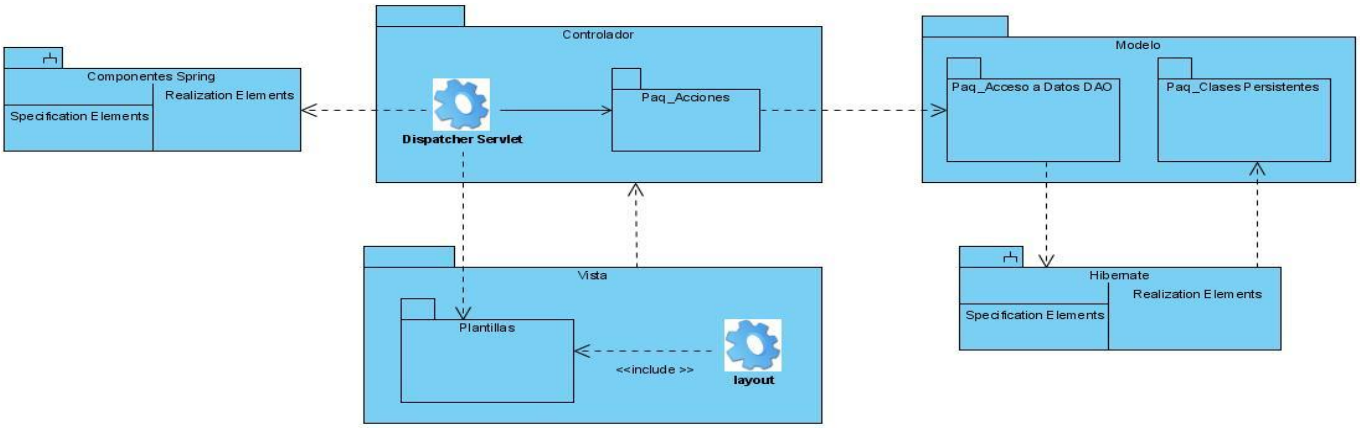

Figura 4. Vista Lógica

A continuación se expone la descripción de los paquetes y subsistemas del diseño que conforman la vista lógica.

- **Paquete Vista**

La vista se encarga de producir páginas que se muestran como resultado de las acciones.

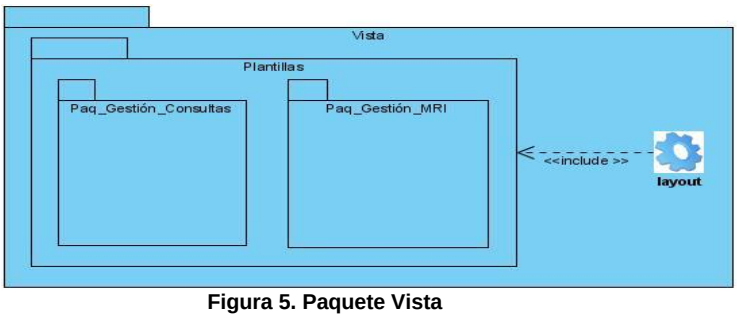

Figura 5. Paquete Vista

- ✓ **La clase *layout*** contiene los elementos que se muestran de forma idéntica para las páginas web a lo largo de toda la aplicación.

  El patrón utilizado en este caso fue el Vista Compuesta (Composite View). Este patrón hace que la representación de vistas sea más manejable, ya que gestiona los diferentes elementos de una página por medio de una plantilla. Frecuentemente las páginas web contienen una combinación de contenido dinámico y elementos estáticos, tales como cabeceras, pies, logos, imágenes de fondo, etc. La parte dinámica es particular para cada página, pero los elementos estáticos suelen ser los mismos para todas las páginas.

  La plantilla de la vista compuesta captura esas características comunes. La integración debe ser dinámica, siendo el Composite View básicamente un *layout* (diseño, esquema) que componga dicha página.Con la aplicación de este patrón de diseño mejora la modularidad y la reutilización, mejora también la flexibilidad, el mantenimiento, y la manejabilidad.

- ✓ **Paquete Plantillas:** Agrupa todas las plantillas o páginas de cada módulo que se encargan de visualizar las variables definidas en el paquete del controlador. Constituyen la presentación de los datos de la acción que se está ejecutando. Dentro del cual se encuentran:
  - ✓ **Paquete Gestión de Consultas:** Contiene las clases que generan las páginas interfaces de usuario, para la gestión de las consultas.
  - ✓ **Paquete Gestión de MRI**: Contiene las clases que generan las páginas interfaces de usuario, para la gestión de MRI.

- ➢ **Paquete Controlador**

  Contiene las clases que encapsulan la lógica del dominio, y se encargan del acceso a los datos almacenados en el gestor de base de datos.

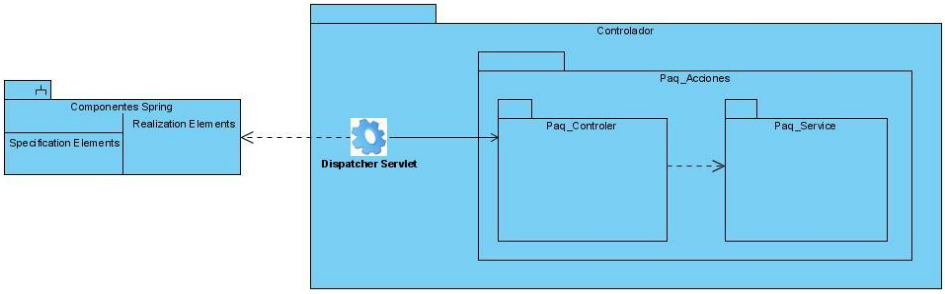

Figura 6. Paquete Controlador

- ✓ **Dispatcher Servlet:** Controlador que atiende las peticiones realizadas por los usuarios, delega a otro componente de Spring el procesamiento de la solicitud; es quien determina cuál será el controlador que recibirá la petición del usuario.
- ✓ **El Paq_Acciones**: Encierra los paquetes Paq_Controller y Paq_Service de cada módulo.
- ✓ **El Paq_Controller:** Se encarga de recuperar la información necesaria mediante comunicaciones que establece con las diferentes clases contenidas en el Paq_Service.
- ✓ **Paq_Service:** Ccontiene las clases que prestan servicios.
- ✓ **Subsistema Componentes Spring:** Representa todas las clases del framework Spring que serán utilizadas durante el funcionamiento del sistema.
- ✓ **Handler Mapping:** Clase dedicada a manejar las peticiones que se reciben desde el cliente, entregándole al Dispatcher Servlet el controlador a ejecutar.
- ✓ **View Resolver:** Es el componente dedicado a conocer cómo generar las vistas y qué tecnología utilizar para esto.

Se utiliza el patrón Facade (o Fachada como también se le conoce), éste es un patrón de software cuyo objetivo se basa en simplificar o en hacer más amigable la complejidad asociada a una librería de software. Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

El uso de del patrón Facade está recomendado para:

- Simplificar el uso y comprensión de una librería de software.
- Encapsular una interfaz de librería poco amigable en un interfaz más coherente o mejor estructurado.
- Centralizar las dependencias externas hacia la librería en uno o pocos puntos de entrada.

➢ **Paquete Modelo**

Contiene las clases que encapsulan la lógica del dominio, y se encargan del acceso a los datos almacenados en el gestor de base de datos.

- ✓ **El Paq_Acceso a Datos DAO**: Contiene las clases de acceso a datos.

  Se diseñó un componente para el acceso a los datos aplicando el patrón Data Access Object (DAO), que le permite al sistema gran escalabilidad pues desacopla la lógica de negocio de la lógica de acceso a datos, de manera que se pueda cambiar la fuente de datos fácilmente, y además reduce la complejidad del código de los objetos de negocio.

El DAO implementa el mecanismo de acceso requerido para trabajar con la fuente de datos. Los componentes de negocio que tratan con el DAO utilizan una interface simple expuesta por el DAO para sus clientes. El DAO oculta completamente los detalles de implementación de la fuente de datos a sus clientes. Como la interface expuesta por el DAO no cambia cuando cambia la implementación de la fuente de datos subyacente, este patrón permite al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. Esencialmente el DAO actúa como un adaptador entre el componente y la fuente de datos.

Se utilizará el patrón DAO por la transparencia en el acceso a los datos, y porque permitiría de ser necesario en un futuro, la migración más fácil a un gestor de base de datos diferente, y por tanto, a una implementación de la capa de acceso a datos diferente, también por su aporte a la reducción de la complejidad del código de los objetos de negocio y la centralización de todos los accesos a datos en una capa independiente.

- ✓ **El Paq_Clases Persistentes:** Contiene las clases entidades.
- ✓ **El subsistema Hibernate:** En el diseño de una aplicación una parte muy importante es la manera en la cual se acceden a los datos en la base de datos (en adelante BBDD), determinar esta parte se convierte en un punto crítico para el futuro desarrollo de una aplicación. Lo claro es que se debe separar el código de las clases de negocio de la realización de sentencias SQL contra la BBDD. Por lo tanto, Hibernate es el puente entre la aplicación y la BBDD, sus funciones van desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.

2.4.3. Vista Despliegue

La vista de despliegue define la arquitectura física del sistema por medio de nodos interconectados. Estos nodos son elementos hardware sobre los cuales pueden ejecutarse los elementos software.
Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a menudo, capacidad de procesamiento.

A continuación se hace una breve descripción de cómo será desplegada la aplicación, mediante la vista de despliegue, mostrada en la figura 8:

La PC cliente, la cual se podrá encontrar en cualquier lugar de CNEURO, contará con una impresora conectada por puerto USB. De igual forma esta PC cliente se conectará a un servidor Web mediante el protocolo HTTP, para hacer sus peticiones. Existirá un servidor de BD, que se conectará al servidor Web por medio de JDBC, para realizar las consultas y accesos a la Base de Datos. Se contará con un servidor FTP, al cual la PC cliente se conectará mediante una conexión FTP, para que el especialista almacene en él las Imágenes de Resonancia Magnética, y que puedan ser procesadas posteriormente. Para realizar el procesamiento de las Imágenes de Resonancia Magnética, el servidor FTP se conecta mediante RMI a un Servidor Distribuidor, el cual tiene asociado una serie de PCs Ociosas, conectados por medio del protocolo RMI, que ayudan en dicho procesamiento. Una vez procesadas las imágenes, la PCs Ociosas guardarán sus direcciones en la Base de Datos, y el Servidor Distribuidor enviará las imágenes nuevamente al servidor FTP, mediante una petición de este último, para ser almacenadas en él y que puedan ser consultadas por los especialistas posteriormente.

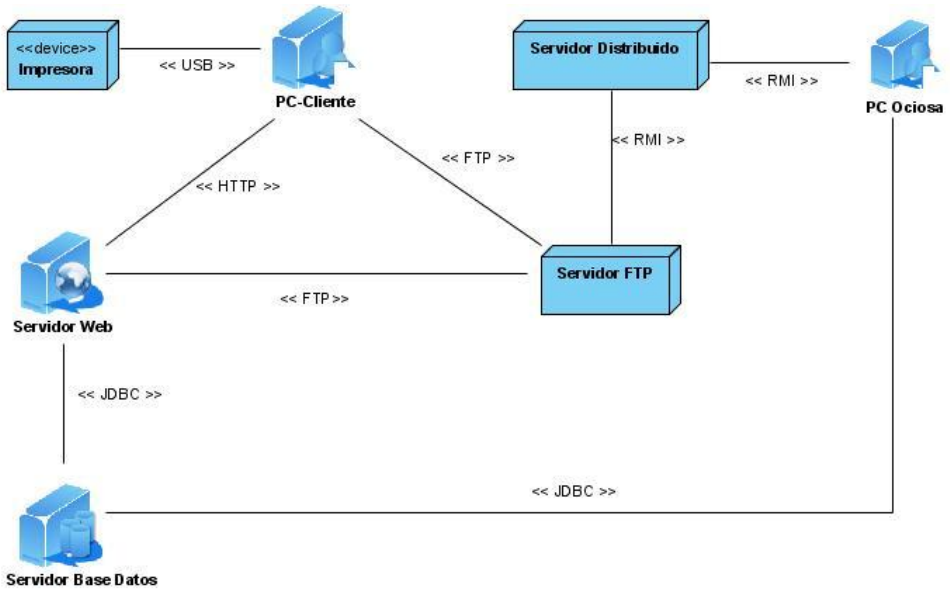

Figura 7. Vista de Despliegue

2.4.4. Vista Implementación

La vista de implementación muestra el software como los elementos físicos que lo integran: componentes, ficheros, librerías. Los componentes representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas, pueden ser simples archivos, paquetes, bibliotecas cargadas

dinámicamente, etc. Los subsistemas organizan la vista de realización de un sistema, cada subsistema puede contener componentes y otros subsistemas.

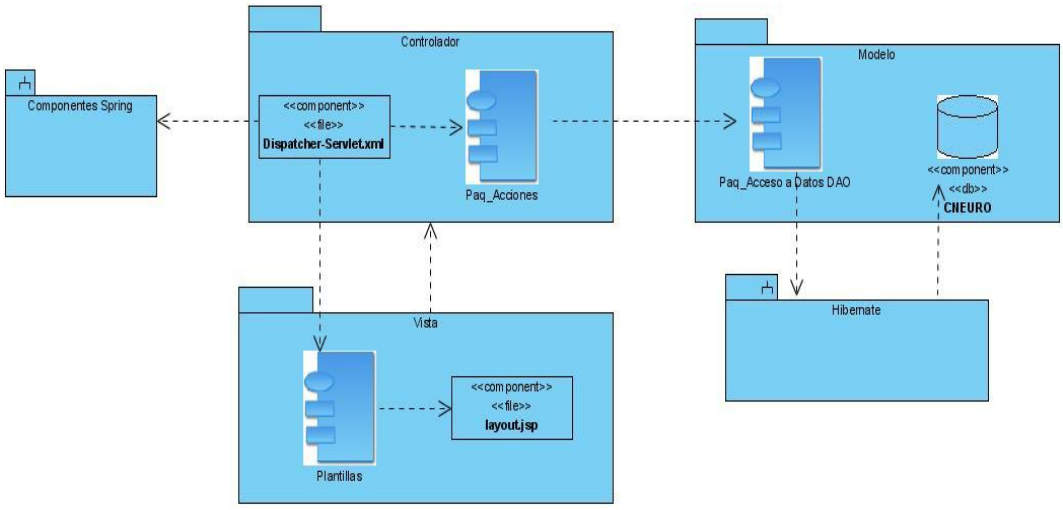

Figura 8. Vista de Implementación

- **Componente layout.jsp:** Componente que implementa la clase *layout* del diseño. Contiene los elementos que se muestran de forma idéntica a lo largo de toda la aplicación.
- **Paquete de componente Plantillas:** Agrupa los componentes que implementan el código correspondiente a cada plantilla que utilizan los módulos.
- **Paquete de componente Paq_Acciones:** Encapsula los componentes Controller y Service de cada módulo.
- **Componente Dispatcher Servlet:** Servlet encargado de gestionar el flujo de la aplicación. Se debe crear un fichero xml para configurar los controladores de la aplicación. Este servlet se encarga de cargar el fichero. La estructura es <nombreContexto-servlet.xml>
- **Paquete de componente Paq_Acceso a Datos DAO:** El DAO es el Data Access Object, es decir, la clase donde reside la lógica de manejo de Hibernate, de esta forma se consigue que la lógica de negocio no sepa nada de Hibernate, y siempre que se quiera acceder a los datos se hará usando esta clase.
- **Subsistema Hibernate:** Se especifica el subsistema Hibernate que es el ORM que utiliza Spring, el cual proporciona persistencia para los objetos y un servicio de consultas. Hibernate a su vez

proporciona una interfaz entre el código java y el código SQL de la base de datos, permitiendo cambiar fácilmente de sistema gestor de base de datos.

- **Componente Base Datos CNEURO:** Encapsula todos los datos del sistema.

2.4.5. Distribución Física de los Componentes.

En la figura 9 se muestra la distribución de los componentes en los nodos físicos, así como otros componentes que son reutilizados y que constituyen programas independientes.

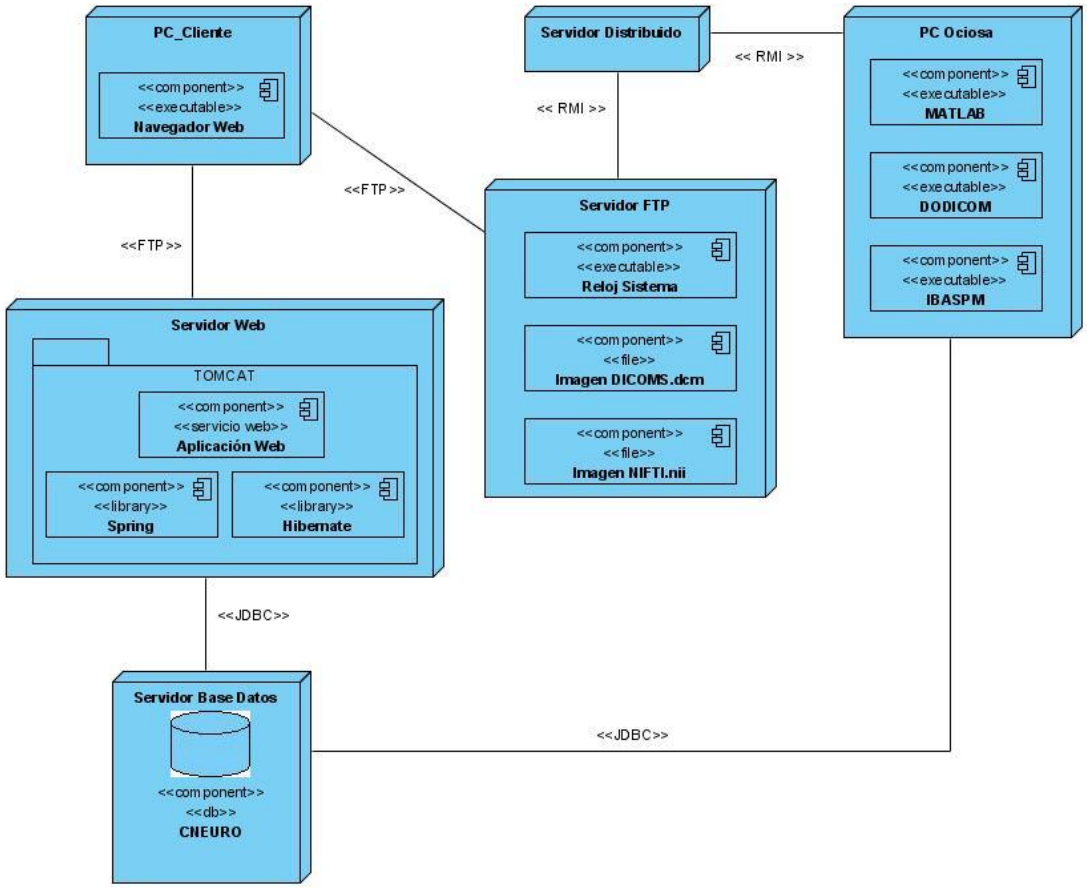

Figura 9. Distribución física de componentes

- **Nodo PC_Cliente:** Tendrá instalado un explorador web Internet Explorer (5.5 o superior) o Mozilla Firefox 2.0 o superior para que el cliente interactúe con la aplicación.

- **Nodo Servidor Web:** Tendrá instalado el servidor de aplicaciones Tomcat, donde se alojarán los servicios Web que tendrá la plataforma y las librerías de los frameworks Spring e Hibernate.
- **Nodo Servidor Base Datos:** Se encontrará la base de datos de la plataforma, con los datos de los estudios de neuroimágenes.
- **Nodo Servidor FTP:** Almacenará las imágenes de Resonancia Magnética realizadas a los sujetos.
    - **Reloj Sistema:** Se encargará de funcionar como un centinela, es decir, este servicio estará constantemente velando en el servidor FTP, por la llegada de nuevos estudios al mismo. En caso de que la respuesta sea positiva el servicio ordena el procesamiento de los nuevos estudios sobre la plataforma de cómputo distribuida.
    - **Imagen DICOMS.dcm:** Los científicos de CNEURO realizan una serie de pruebas como la de Resonancia Magnética, obteniendo como resultado un conjunto de imágenes en formato DICOMS (.dcm). Estas imágenes serán almacenadas en el servidor FTP para luego ser procesadas.

      DICOMS (Digital Imaging and Communications in Medicine) es un estándar reconocido mundialmente para el intercambio de imágenes médicas pensado para el manejo, almacenamiento, impresión y transmisión de imágenes médicas.
    - **Imagen NIFTI.nii:** Luego de ser almacenadas las imágenes de cada estudio, de forma automática, la herramienta DODICOM las convierte a formato NIFTI (.nii), este proceso da como resultado una serie de imágenes divididas en cuatro carpetas (Anatómica, Difusión, Fase y Magnitud) de las cuales se toma la carpeta Anatómica que sirve de entrada a otra herramienta llamada IBASPM, para un último procesamiento que arroja como resultados otra serie de imágenes, las que finalmente son analizadas por los especialistas en sus estudios.

      NIFTI: Es un formato estándar para imágenes que fue diseñado para el análisis científico de las imágenes cerebrales. El formato es simple, compacto y versátil.
- **Nodo Servidor Distribuido:** Se encargará de distribuir el trabajo a las PCs Ociosas para lograr un mejor rendimiento en cuanto a la realización del procesamiento de imágenes de forma automática y distribuida, logrando aminorar el tiempo dedicado al procesamiento de los estudios.

- **Nodo PC Ociosa:** Estarán todos los componentes necesarios para realizar el procesamiento de las imágenes. A pesar de que en el diagrama se muestra solo una PC Ociosa por cuestión de espacio, pudieran existir las que fuesen necesario para realizar un correcto procesamiento.
    - **DODICOM:** Herramienta utilizada para el procesamiento de las imágenes DICOMS.
    - **IBASPM:** Herramienta que procesa las imágenes NIFTI.
    - **MATLAB:** Asistente matemático utilizado para el cálculo en el procesamiento de las imágenes.

Los programas reutilizados son: DODICOM, IBASPM, MATLAB.

2.5. Conclusiones

En este capítulo se dieron a conocer detalles de la organización que tendrá el sistema para un mejor entendimiento. Se plasmaron los requisitos no funcionales que harán que el funcionamiento de la plataforma sea el mejor posible. Se definieron y explicaron detalladamente las diferentes vistas arquitectónicas: Vista de CU, Vista Lógica, Vista de Despliegue y Vista de Implementación.

CAPÍTULO 3: EVALUACIÓN DEL DISEÑO ARQUITECTÓNICO PROPUESTO

3.1. Introducción

En el presente capitulo se realiza la evaluación del diseño arquitectónico propuesto, puesto que no sirve de nada un sistema que no cumpla con los atributos de calidad que se especificaron en los requerimientos no funcionales de los clientes. Por lo que diseñar una correcta arquitectura va a determinar el éxito o el fracaso de un sistema de software, en la medida que ésta cumpla o no con sus objetivos. Debido a esto para reducir tales riesgos, y como buena práctica de ingeniería, es recomendable realizar evaluaciones a la arquitectura.

3.2. Evaluación de arquitecturas de software

¿Por qué evaluar una arquitectura?

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad.

Objetivos de evaluar una arquitectura

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders.

La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo, los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos.

Kazman propone un esquema general en relación a la evaluación de una arquitectura con respecto a sus atributos de calidad. En este sentido, Kazman y sus colegas afirman que de la evaluación de una Arquitectura no se obtienen respuestas del tipo “si - no”, “bueno – malo” o “6.75 de 10”, sino que explica cuáles son los puntos de riesgo del diseño evaluado.

El método de diseño de arquitecturas planteado por Bosch tiene como principal característica la evaluación explícita de los atributos de calidad de la arquitectura durante el proceso de diseño de la

misma. En este sentido, Bosch plantea las técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia.

Tanto Bosch como Kazman indican la importancia de la especificación exhaustiva de los atributos de calidad como base para efectos de la evaluación de una arquitectura de software.

El punto es definir los atributos de calidad en función de sus metas y su contexto, y no como cantidades absolutas.

¿Cuándo una arquitectura puede ser evaluada?

De esta manera, el interés se centra en determinar el momento propicio para efectuar la evaluación de una arquitectura. En este sentido, Kazman amplía el panorama clásico, indicando las ocasiones en que se hace posible hacer la evaluación de una arquitectura. Su planteamiento establece que es posible realizarla en cualquier momento, pero propone dos variantes que agrupan dos etapas distintas: temprana y tarde.

Temprana: No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

Tarde: Consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

¿Quiénes participan en una evaluación?

Generalmente las evaluaciones a la arquitectura se hacen por miembros del equipo de desarrollo, arquitecto, diseñador, entre otros. Sin embargo puede haber también situaciones en las que intervengan especialistas en el tema, además de los clientes que también se interesan por los resultados de la evaluación, ya que en dependencia de los resultados pueden tomar decisiones de continuar o no con el proyecto.

¿Por qué cualidades puede ser evaluada una arquitectura?

A grandes rasgos, Bass establece una clasificación de los atributos de calidad en dos categorías:

- Observables vía ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.
- No observables vía ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

La descripción de algunos de estos atributos se presenta a continuación:

- Atributos de calidad Observables vía ejecución:

- **Disponibilidad**. Es la medida de disponibilidad del sistema para su uso.
- **Confidencialidad**. Es la ausencia de acceso no autorizado a la información.
- **Funcionalidad**. Habilidad del sistema para realizar el trabajo para el cual fue concebido.
- **Desempeño**. Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
- **Confiabilidad**. Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.
- **Seguridad interna**. Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.

➢ Atributos de calidad no Observables vía ejecución:

- **Configurabilidad**. Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- **Integrabilidad**. Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
- **Integridad**. Es la ausencia de alteraciones inapropiadas de la información.
- **Modificabilidad**. Es la habilidad de realizar cambios futuros al sistema.
- **Mantenibilidad**. Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
- **Portabilidad**. Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
- **Reusabilidad**. Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
- **Escalabilidad**. Es el grado con el que se pueden ampliar el diseño arquitectónico de datos o procedimentalmente.

Técnicas de Evaluación

En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer

una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño. Bosch propone diferentes técnicas de evaluación de arquitecturas de software, a saber: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia.

Por lo regular, las técnicas de evaluación cualitativas son utilizadas cuando la arquitectura está en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada. [28]

A continuación se describirá detalladamente la técnica de evaluación basada en escenarios, la cual fue seleccionada, debido a que los métodos de evaluación que se describirán posteriormente se enfocan en esta técnica.

Evaluación basada en escenarios

De acuerdo con Kazman, un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Un escenario consta de tres partes: el estímulo, el ambiente y la respuesta.

Estímulo: Es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración y otros.

Ambiente: Describe las condiciones en la cual se encuentra el sistema en el momento que se recibe el estímulo.

Respuesta: Describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Permite establecer cuál es el atributo de calidad asociado.

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes, a saber: el Utility Tree propuesto por Kazman, y los Perfiles, propuestos por Bosch.

Utility Tree:

Según Kazman, un Utility Tree es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular.

Perfiles:

Un perfil (profile) es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Cada atributo de calidad tiene un perfil asociado. Algunos perfiles pueden ser usados para evaluar más de un atributo de calidad.

3.3. Métodos de Evaluación de Arquitecturas

Hasta hace poco no existían métodos de utilidad general para evaluar arquitecturas de software. Si alguno existía, sus enfoques eran incompletos y no repetibles, y no brindaba mucha confianza. En virtud de esto, múltiples métodos de evaluación han sido propuestos. A continuación se explican algunos de los más importantes.

- **Método de Análisis de Arquitecturas de Software (SAAM)**

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro.

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

El método SAAM lo sugerimos cuándo el atributo de calidad Modificabilidad es el de mayor interés.

El método de evaluación SAAM comprende seis pasos. Ver [31]

- **Método de Análisis de Acuerdos de Arquitectura(ATAM)**

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que

revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos.

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros.

El método ATAM es más profundo para evaluar aspectos más relacionados con la arquitectura, como la performance o la confiabilidad.

El método de evaluación ATAM comprende nueve pasos, agrupados en cuatro fases. Ver [30]

- **Método de Revisiones Activas para Diseños Intermedios (ARID)**

Hacerle revisiones a la arquitectura en las etapas intermedias nos provee de una valiosa visón de la viabilidad de la arquitectura a construir, así como también nos permite descubrir errores e inconsistencias.

De acuerdo con Kazman el método Active Reviews for Intermediate Designs (ARID) es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente.

Kazman propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente, propone la utilización de las características que proveen tanto ADR como ATAM por separado. De ADR, resulta conveniente la fidelidad de las respuestas que se obtienen de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software. El método ARID evalúa mejor la factibilidad de la arquitectura.

Se basa en ensamblar el diseño de los stakeholders para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Revisión) [29].

A Continuación se presentan los pasos del método de evaluación ARID, con una breve descripción de cada uno:

Fase 1: Actividades Previas

1. Identificación de los encargados de la revisión: Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.

2. Preparar el informe de diseño: El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.

3. Preparar los escenarios base: El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.

4. Preparar los materiales: Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.

Fase 2: Revisión

5. Presentación del ARID: Se explica los pasos del ARID a los participantes.

6. Presentación del diseño: El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.

7. Lluvia de ideas y establecimiento de prioridad de escenarios: Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.

8. Aplicación de los escenarios: Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos:

Se agota el tiempo destinado a la revisión.

Se han estudiado los escenarios de mayor prioridad.

El grupo se siente satisfecho con la conclusión alcanzada

Puede suceder que el diseño presentado sea conveniente con la exitosa aplicación de los escenarios, o por el contrario, no conveniente cuando el grupo encuentra problemas o deficiencias.

9. Resumen: Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión, y agradece por su participación.

El método escogido para realizar la evaluación de la arquitectura de la plataforma fue el ARID, debido a que es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Es un híbrido entre el Active Design Review (ADR) y el ATAM, obteniendo las mejores prácticas de ambos. Además es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos.

3.4. Evaluando la arquitectura de la Plataforma Mapeo Cerebral

Con vistas a evaluar el diseño arquitectónico antes propuesto, se decidió utilizar el método ARID, que utiliza la técnica de evaluación basada en escenarios con el instrumento perfiles, donde se hace un análisis de los principales escenarios arquitectónicos. Para ello se seleccionaron los atributos de calidad más importantes en aquellos escenarios de acuerdo a un perfil específico.

Debido a que en la presente investigación los encargados de llevar a cabo la evaluación de la arquitectura propuesta serán los integrantes del grupo de arquitectura que definieron la misma, no se siguieron estrictamente los pasos propuestos por el ARID. Los arquitectos del sistema tienen un dominio completo del diseño arquitectónico del mismo, y presentan conocimientos del método a aplicar para la evaluación, por lo que no se realizó la Fase1, pasando directamente a aplicarse los tres últimos pasos de la Fase 2.

Atributos de calidad seleccionados:

1. Eficiencia.
2. Mantenibilidad.
3. Portabilidad.
4. Confiabilidad.
5. Seguridad.
6. Usabilidad.

A continuación se exponen los escenarios seleccionados por su interés dentro de la arquitectura del sistema.

Escenario 1: Búsqueda de información de forma rápida.

Atributo de calidad: Eficiencia

Perfil: Desempeño

Estímulo: Realizar búsqueda de información por parte de múltiples usuarios.

Respuesta: Muestra los datos rápidamente.

Relación Atributo-Escenario: Las decisiones arquitectónicas que garantizan una respuesta satisfactoria del sistema ante la conexión concurrente de múltiples usuarios a la aplicación haciendo diferentes o las mismas peticiones, son la selección del SGBD PostgreSQL, capaz de manejar las conexiones concurrentes de muchos usuarios de forma eficiente, y el Framework Hibernate, que con su caché de dos niveles ofrece una gran flexibilidad y rendimiento, ya que al almacenar las estructuras de objetos en memoria, evita volver a buscar dichos objetos en la base de datos ahorrando multitud de accesos, y por consiguiente tiempo. Además incluye una caché de consultas que da la posibilidad de obtener rápidamente resultados que ya habían sido consultados previamente.

Escenario 2: Migración del sistema gestor de base de datos.

Atributo de calidad: Mantenibilidad

Perfil: Mantenimiento

Estímulo: Migrar de PostgreSQL a otro SGBD.

Respuesta: Cambiar el gestor de base de datos PostgreSQL por un nuevo gestor definido.

Relación Atributo-Escenario: En el caso que sea necesario en algún momento la migración de sistema gestor de bases de datos, esta no proporcionará grandes problemas ya que la plataforma ha sido desarrollada utilizando el framework Hibernate, el cual se encarga de crear una capa separada que se ocupa del acceso a datos con total independencia del gestor y la base de datos, dando la oportunidad de trabajar con varios gestores y bases de datos dentro de la misma aplicación sin que esto cree ningún conflicto en el modelo de objetos. Esto posibilita realizar de forma sencilla el cambio de SGBD, incluso, la existencia de varios de ellos.

Escenario 3: El sistema puede cambiar alguna de sus partes fácilmente.

Atributo de calidad: Mantenibilidad

Perfil: Mantenimiento

Estímulo: Modificar algunas de las partes del sistema.

Respuesta: Cambio de partes del sistema fácilmente.

Relación Atributo-Escenario: Gracias a la implementación del patrón arquitectónico MVC se consigue un mantenimiento más sencillo del sistema, puesto que acciones como sustituir las vistas, agregar nuevas funcionalidades al controlador o cambiar de base de datos, no afecta a los elementos externos al cambio.

Escenario 4: Cambio de sistema operativo.
Atributo de calidad: Portabilidad
Perfil: Portabilidad
Estímulo: Necesidad de cambiar la plataforma de sistema operativo de forma temporal o definitiva.
Respuesta: Cambiar de sistema operativo de forma temporal o definitiva.
Relación Atributo-Escenario: Una decisión crucial para lograr la obtención de un producto portable fue la selección de herramientas y tecnologías para el desarrollo de la plataforma, ya que el hecho de que la misma sea implementada en el lenguaje de programación Java y que use como gestor de BD PostgreSQL, permite que ésta corra tanto sobre el sistema operativo Linux como Windows, o sobre cualquier sistema operativo que posea una máquina virtual de Java.

Escenario 5: Los datos introducidos por el usuario deben ser los más correctos posible.
Atributo de calidad: Confiabilidad
Perfil: Confiabilidad
Estímulo: Introducir datos en el sistema correctamente.
Respuesta: El sistema lleva a cabo la validación de los formularios para lograr una mayor confianza sobre los datos con que opera.
Relación Atributo-Escenario: El funcionamiento básico de este mecanismo consiste en que si el usuario introduce datos no válidos y envía el formulario, la próxima página que se muestra contiene los datos con los mensajes de error, posibilitando así al usuario arreglarlos antes de volver a enviar el formulario.

Escenario 6: Autenticar los usuarios para permitir el uso adecuado de la aplicación.
Atributo de calidad: Seguridad
Perfil: Seguridad
Estímulo: Intento de entrada al sistema por parte de usuario no autorizado.
Respuesta: El servidor debe mostrar una página de autenticación a los usuarios para que introduzcan sus credenciales, impidiendo así brindar servicios a los usuarios no autorizados.
Relación Atributo-Escenario: Las decisiones arquitectónicas que responden a la seguridad en el acceso a la aplicación y permitir por consiguiente su uso solo por personas autorizadas son las facilidades que brinda el Framework Spring de implementar la seguridad utilizando uno de los Framework de los que se

compone: Framework Acegi Security, el cual se encarga de garantizar la autenticación de usuarios y la autorización al acceso de los recursos. Proporciona la funcionalidad necesaria para adoptar mecanismos de seguridad en aplicaciones Java utilizando características de programación orientada a aspectos, de forma transparente para el desarrollador, utilizando para ello el soporte prestado por el Framework Spring; este Framework permite definir reglas de acceso basadas en roles o tipos de usuario para los recursos y funcionalidades de la aplicación.

Escenario 7: El sistema debe ser entendible y fácil de usar.
Atributo de calidad: Usabilidad
Perfil: Usabilidad
Estímulo: La plataforma debe mostrarle al usuario una interfaz amigable y entendible.
Respuesta: El sistema presenta al usuario una interfaz atractiva e interactiva, con un menú general que lo guía en la búsqueda de la información que necesita.
Relación Atributo-Escenario: Gracias a la implementación del patrón de diseño Composite View, se garantiza una estructura afín a todas las páginas de los módulos, estableciendo iguales íconos para acciones similares y regularidades en el orden de aparición de algunos campos de formularios.

3.5. Conclusiones

En este capítulo se expusieron las principales técnicas y métodos usados para evaluar una arquitectura de software. Tras el estudio de los métodos de evaluación se eligió para su utilización el método ARID, permitiendo así, tener una evaluación de la arquitectura en una etapa temprana del diseño, y mostrando cómo se puede analizar el cumplimiento de los atributos de calidad. De esta forma se concluye que la arquitectura cumple con los requerimientos no funcionales previstos para el desarrollo del sistema.

CONCLUSIONES GENERALES

La arquitectura de software descrita da solución al problema planteado, permitiendo organizar el proceso de desarrollo y definir la estructura que tendrá el sistema, ya que las decisiones en las que se basa garantizan la implementación y evolución exitosa de la aplicación web requerida por el Centro de Neurociencias de Cuba.

En el presente trabajo se hizo un análisis de cada una de las herramientas candidatas, de las cuales la mayoría están basadas en software libre, para cubrir las necesidades de los componentes de la arquitectura.

- Se seleccionó el estilo y patrón arquitectónico Modelo Vista Controlador, así como otros patrones de diseño, proporcionando una mayor calidad en la estructura y el diseño de la aplicación.
- Se diseñaron las vistas del sistema utilizando el modelo de las 4+1 de Philippe Kruchten, de las cuales se representaron y describieron la Vista de Casos de Uso, la Vista Lógica, la Vista de Despliegue y la Vista de Implementación.
- Se describió la arquitectura, permitiendo el entendimiento y comprensión de la misma por parte del equipo de desarrollo.
- Se realizó un análisis de los métodos de evaluación de arquitectura de software, evaluando la misma con el método ARID, evidenciándose el cumplimiento de los atributos de calidad seleccionados.

RECOMENDACIONES

Una vez finalizado el desarrollo de esta investigación, arribamos a realizar las siguientes recomendaciones:

- Se recomienda aplicar otros métodos de evaluación de la arquitectura para aumentar la fortaleza del sistema.
- Se recomienda analizar este diseño arquitectónico por todas aquellas aplicaciones que tengan características similares a ésta.
- Se recomienda poner en práctica el diseño arquitectónico propuesto.
- Se recomienda aplicar una arquitectura que integre todos los componentes y módulos del Proyecto Mapeo Cerebral Humano de Cuba.
- Se recomienda utilizar un framework especializado en el mapeo cerebral.

REFERENCIAS BIBLIOGRÁFICAS

1. Llanes, Clara Carmen. Mapeo cerebral humano: un reto del siglo XXI . Mapeo cerebral humano: un reto del siglo XXI . [Consultado el :octubre 19, 2009].Disponible en: http://www.voltairenet.org/article128202.html.

2. López, Roxana y Benítez, Eric. *Liberación del Sistema de Gestión de la Información del Proyecto Mapeo Cerebral Humano Cubano: Modulo de Examen Clínico*.UCI Facultad 6.[Consultado en: 2009].

3. Avedaño, Bárbara. "Trazos sobre una carta inconclusa". bohemia. [Consultado el: octubre, 2009]. Disponible en:http://www.bohemia.cu/2008/01/23/cienciatecnologia/1-mapa.html.

4. Reynoso, C.B. "Introducción a la Arquitectura de Software". 2004 [Consultado en: noviembre 2009];
   Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdf.

5. Perry, Dewayne y Wolf Alexander L. "Foundations for the study of software architecture".
   ACM SIGSOFT Software Engineering Notes, 17(4), pp. 40–52, [Consultado en: Octubre de 2009].

6. Camacho, E., F. Cardeso, y G. Nuñez. Arquitecturas de Software. 2004 [Consultado: noviembre 2009]; Disponible en:
   http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

7. Ridao, M. Uso de patrones en el proceso de construcción de escenarios. 2006
   [Consultado en: noviembre 2009]; Disponible en:
   http://postgrado.info.unlp.edu.ar/Carrera/Magister/Ingenieria%20de%20Software/Tesis/Ridao.pdf

8. Ishikawa, Alexander y Silverstein, Jacobson y Fidsdahl-King, Angel. "A Pattern Language". Oxford University Press, New York, 1977.

9. Larman, C., UML y PATRONES. 2004 ed.[Consultado: 2010].

10. Rosanigo, Z.B. Modelos y Patrones. 2000 [Consultado: diciembre 2009]. Disponible en: http://postgrado.info.unlp.edu.ar/Carrera/Magister/Ingenieria%20de%20Software/Tesis/Rosanigo.pdf.

11. García, Joaquín. Patrones de Diseño [Consultado: diciembre 2009]; Disponible en: http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

12. Sánchez, D.M. Técnicas y Metodologías en el desarrollo de Software. 2007 [Consultado: diciembre 2009]; Disponible en: http://kybele.escet.urjc.es/documentos/HC/HC4GL2007-T2-TecnicasI.pdf.

13. OpenUP/Basic Lifecycle. [Consultado: diciembre 2009]; Disponible en: http://epf.eclipse.org/wikis/openupsp/openup_basic/deliveryprocesses/openup_basic_process_0uyGoMlgEdmt3adZL5Dmdw_desc.html

14. Introducción a OpenUP/Basic. [Consultado: diciembre 2009]; Disponible en: http://epf.eclipse.org/wikis/openupsp/openup_basic/customcategories/introduction_to_openup_basic,_BTJ_YMXwEduywMSzPTUUwA.html

15. Sierra,M. INGENIERÍA DEL SOFTWARE I Práctica 1Trabajando con Visual Paradigm for UML, Univ. Cantabria – Fac de Ciencias. [Consultado: noviembre 2009]; Disponible en:http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf

16. Booch, Grady. 1996. Análisis y Diseño Orientado a Objetos. 2da edición. Ed. Addison-Wesley / Díaz de Santos. Pressman, Robert. Ingeniería de Software. [Consultado: 2009]. Disponible en: http://www-01.ibm.com/software/rational/

17. Ascui, I.C. Java y la Programación Orientada a Objetos. 2006 [Consutado: diciembre 2009]; Disponible en: http://iverclaros.blog.galeon.com/1141775160/.

18. JBuilder! Escrito por Audisoft Ltda! [Consultado el: Martes, 23 de Enero de 2010] Disponible en: http://www.audisoft.com/index.php?option=com_content&view=article&id=19

19. Sánchez Rico, Mario Alfredo. Spring, un framework de aplicación.26 [Consultado:Enero 2010]Disponible en; http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/sanchez_r_ma/capitulo3.pdf

20. Vázquez Rodríguez, Josefina Adriana. Creación de un repositorio para programas de Java utilizando la herramienta de ORM Hibernate. [Consultado: 2 febrero 2010] Disponible en: http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/vazquez_r_ja/capitulo2.pdf

21. Introducción - Bases de datos [Consultado: 10 Febrero 2010].Disponible en: http://es.kioskea.net/contents/bdd/bddintro.php3

22. PostgreSQL, BBDD libre. [Consultado: Febrero 2010] Disponible en: http://www.deambulando.com/2007/01/15/postgre/

23. Romero, Diego F. "¿Qué Es Un Servidor De Aplicaciones?". [Consultado: 9 Marzo 2010] Disponible en: http://www.editum.org/Que-Es-Un-Servidor-De-Aplicaciones-p-473.html

24. Don Denoncourt. Elección del servidor de aplicaciones web. [Consultado: febrero 2010]. Disponible en: http://www.help400.es/asp/scripts/nwart.asp?Num=131&Pag=10&Tip=T

25. Java Server Pages TM (JSP) [Consultado: 20 Febrero 2010]. Disponible en: http://mipagina.cantv.net/williamyanez/jsp/default.htm

26. Matlab, Poderosa Herramienta Matemática, [Consultado: 2010]. Disponible en: http://www.programasde.com/matlab-76-poderosa-herramienta-matematica/

27. Carrascoso, Yoan Arlet y Chaviano, Enrique y Céspedes, Anisleydi. *Procedimiento para la evaluación de arquitecturas de software basadas en componentes*. [Consultado:2010]. Disponible en: http://www.gestiopolis.com

28. Gómez, Omar Salvador. Evaluando Arquitecturas de Software. Parte 1. 01, México: Brainworx S.A, [Consultado: 2010].

29. Gómez, Omar Salvador. Evaluando Arquitecturas de Software. Parte 2. Métodos de Evaluación. 02, México: Brainworx S.A [Consultado: 2010].

30. Camacho, Erika y Cardeso, Fabio y Nuñez, Gabriel. Arquitectura de software Guía de estudio. [Consultado: Abril 2010]. pp 45. tabla 17

31. Camacho, Erika y Cardeso, Fabio y Nuñez, Gabriel. Arquitectura de software Guía de estudio. [Consultado: Abril 2010]. pp 46. tabla 18.

.

BIBLIOGRAFÍA

- Suárez González, Héctor. Manual Hibernate2003. Disponible en:http://www.javahispano.org/contenidos/es/manual_hibernate/
- Larman, C., UML y PATRONES. 2004 ed.
- Java Server Pages TM (JSP). Disponible en: http://mipagina.cantv.net/williamyanez/jsp/default.htm
- Romero, Diego F. "¿Qué Es Un Servidor De Aplicaciones?. 25 Diciembre 2007. [Consultado: 09 Marzo 2010]. Disponible en: http://www.editum.org/Que-Es-Un-Servidor-De-Aplicaciones-p-473.html
- Vázquez Rodríguez, Josefina Adriana. Creación de un repositorio para programas de Java utilizando la herramienta de ORM Hibernate. [Consultado: 24 abril 2010]. Disponible en:http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/vazquez_r_ja/capitulo2.pdf
- Evaluando Arquitecturas de Software. Parte 1. Panorama General. Gómez, Omar Salvador. 01, México: Brainworx S.A.
- Camacho, Erika y Cardeso, Fabio y Nuñez, Gabriel. Arquitecturas de Software.
- Bosch, J. (2000). Design & Use of Software Architectures. Addison-Wesley.
- Bass, Len. y Rick Kazman, Software Architecture in Practice. 2003: Addison-Wesley Professional.pp 560.
- Kruchten, P. Architectural Blueprints— The “4+1” View Model of Software Architecture. 1995 [Consultado: noviembre 2010]; disponible en: http://www.cs.ubc.ca/~gregor/teaching/papers/4+1viewarchitecture.pdf.
- Pressman, R.S., Ingienería del Software. Un enfoque práctico. V ed. Vol. I. 2005.

GLOSARIO DE TÉRMINOS

API: *Application Program Interface* (programa de aplicación de interfaz). Conjunto de especificaciones de comunicación entre componentes de software. Representa un método para conseguir abstracción en la programación.

GRID: Es una infraestructura que permite compartir recursos geográficamente dispersos para resolver problemas de gran escala. Estos recursos compartidos pueden ser hardware, software, datos e información, dispositivos electrónicos, etc.

JDBC: Es el acrónimo de *Java Database Connectivity*, un API que permite la ejecución de operaciones sobre base de datos desde el lenguaje de programación Java.

Open Source: Cualidad de algunos software de incluir el código fuente en la distribución del programa. En general se usa para referirse al software libre.

Plug-ins: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

PC: Es la expresión estándar que se utiliza para denominar a las computadoras personales en general.

RMI: Invocación a métodos remotos, tecnología java para el trabajo distribuido.

WebDAV (más conocido como DAV, protocolo que amplía las posibilidades del HTTP/1.1 añadiendo nuevos métodos y cabeceras.)

Servlet: Objeto Java que se ejecuta dentro del contexto de un servidor web y que sirve para manipular peticiones y respuestas del cliente web. La mayor parte de los frameworks utilizan un servlet como núcleo.

XML: Siglas en inglés de extensible Markup Language (lenguaje extensible de marcas).

IEEE: (Institute of Electrical and Electronics Engineers). Asociación de profesionales norteamericanos que aporta criterios de estandarización de dispositivos eléctricos y electrónicos.

Stakeholder: Grupos y/o individuos que puedan afectar o que son afectados por el logro de los objetivos de la organización.

HTML: Siglas de HyperText Markup Language (Lenguaje de Marcado de Hipertexto), es el lenguaje de marcado predominante para la elaboración de páginas web.

BSD: Son las iniciales de Berkeley Software Distribution (en español, Distribución de Software Berkeley) y se utiliza para identificar un sistema operativo derivado del sistema Unix nacido a partir de los aportes realizados a ese sistema por la Universidad de California en Berkeley.

Commit: Es una sentencia en SQL que finaliza una transacción de base de datos dentro de un sistema gestor de base de datos relacional (RDBMS) y pone visibles todos los cambios a otros usuarios.

PL/pgSQL: (Procedural Language/PostgreSQL Structured Query Language) es un lenguaje imperativo provisto por el gestor de base de datos PostgreSQL. Permite ejecutar comandos SQL mediante un lenguaje de sentencias imperativas y uso de funciones, dando mucho más control automático que las sentencias SQL básicas.

Unicode: Es un estándar de codificación de caracteres diseñado para facilitar el tratamiento informático, transmisión y visualización de textos de múltiples lenguajes y disciplinas técnicas además de textos clásicos de lenguas muertas.

Layout: Es la ordenación y colocación de todos los elementos que componen una página web, es decir textos, imágenes, tablas, gráficos, etcétera. También son elementos del layout los colores y el tipo de letra.

EPF Composer: Es una herramienta que permite generar marcos de procesos de software para una organización. Se basa en la primicia de modelos reutilizables que permiten tomar las mejores prácticas del mercado e integrarlas en el framework de procesos organizacionales. Basada por completo en el ambiente de desarrollo de Eclipse.