Universidad de las Ciencias Informáticas
Facultad 6

Título: Arquitectura de Software para el sistema alasClínicas

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

Autoras: Aliekna Rodríguez Isaac
Imara Tamayo Guerra

Tutores: Ing. Lucía Rodríguez García
Ing. Richard Díaz Pompa

Ciudad de La Habana, Junio 2010
Año 52 de la Revolución

“La pasión de saber hace que el hombre aprenda más rápidamente y aprenda en menos tiempo; la pasión de saber, la conciencia de la necesidad de saber, hace que los conocimientos se adquieran más rápidamente y, sobre todo, la vida, el trabajo práctico, los problemas diarios, constantemente nos estarán enseñando la necesidad de cada conocimiento.”

Fidel Castro Ruz

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 _____.

Aliekna Rodríguez Isaac

_____
Firma del Autor

Imara Tamayo Guerra

_____
Firma del Autor

Lucía Rodríguez García

_____
Firma del Tutor

Richard Díaz Pompa

_____
Firma del Tutor

DATOS DE CONTACTO

Tutor: Ing. Lucía Rodríguez García
Especialidad de graduación: Ingeniería en Ciencias Informáticas
Categoría docente: Instructor
Categoría Científica: no
Años de experiencia en el tema: 0
Años de graduado: 2
Correo electrónico: lrodriguezg@uci.cu
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

Tutor: Ing. Richard Díaz Pompa
Especialidad de graduación: Ingeniería en Ciencias Informáticas
Categoría docente: Instructor en Adiestramiento
Categoría Científica: no
Años de experiencia en el tema: 1
Años de graduado: 1
Correo electrónico: rdiaz@uci.cu
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba

RESUMEN

El presente Trabajo de Diploma describe una Arquitectura de Software(AS) basada en la tecnología Java 2 Enterprise Edition (J2EE)[1], que podría ser utilizada como referencia en el desarrollo de Aplicaciones Web en las que se aplique esta tecnología. La investigación surge producto de la necesidad de una arquitectura para el sistema alasClínicas, la cual debe proporcionar los elementos precisos para el desarrollo del software y la posterior incorporación de nuevas funcionalidades. Para alcanzar los objetivos propuestos, la investigación se enmarca en el estudio de los conceptos relacionados con la AS considerando las características de la tecnología J2EE, con el fin de obtener una arquitectura que responda a las buenas prácticas de programación, así como la definición y evaluación de la arquitectura propuesta, garantizando soluciones robustas que carezcan de complejidad innecesaria y con mejoras en propiedades como la escalabilidad y la reusabilidad.

Palabras Claves:

alasClínicas, aplicación Web, Arquitectura de software (AS), J2EE

[1]Define el estándar para el desarrollo de aplicaciones empresariales multicapas agregando pleno apoyo a los componentes Enterprise JavaBeans, Servlets de Java API, Java Server Pages y tecnología XML, propiciando que se puedan desarrollar aplicaciones Web bajo la tecnología Java.

ÍNDICE

INTRODUCCIÓN

La informatización de cada esfera de la sociedad es una necesidad que existe actualmente en el mundo. Llevarla a cabo ha sido una labor en la cual se han realizado intensos esfuerzos, con el objetivo de lograr que la calidad del software sea una premisa en todas las fases de desarrollo del mismo, ya que incluye todas las cualidades que lo caracterizan y determinan su utilidad y existencia: eficiencia, flexibilidad, corrección, confiabilidad, portabilidad, usabilidad, rendimiento, escalabilidad, seguridad e integridad.

Para llevar a cabo el desarrollo de proyectos informáticos de gran envergadura es de vital importancia considerar que estén desarrollados sobre una arquitectura sólida, pues la misma no sólo dicta cómo está construido el sistema, sino que determina si la aplicación tiene los atributos de calidad esenciales para un proyecto exitoso, estos incluyen usabilidad, facilidad de mantenimiento, si cumple con los requerimientos de rendimiento y estándares de seguridad, y si puede evolucionar fácilmente ante los cambios de los requerimientos, de ahí la necesidad de priorizar el diseño de la arquitectura tanto para el producto de software como para el proceso de desarrollo .

La Arquitectura de Software (AS) como un área de investigación que sustenta la calidad del proceso de desarrollo y práctica dentro de la Ingeniería de Software se convierte en un punto clave en el proceso de desarrollo del software en general. Además de buscar un diseño sólido es preciso que se haga una evaluación de las decisiones tomadas durante el diseño, pues este paso brinda la posibilidad de saber el grado con que se han alcanzado los atributos de calidad que se persiguen.

Aunque en la actualidad los países desarrollados son los mayores productores y consumidores de sistemas informáticos en el mundo; países como Cuba, Bolivia y la República Bolivariana de Venezuela se han introducido en la industria del software, con el fin de crear soluciones informáticas para la informatización de la sociedad y a su vez adentrarse más en el mercado del software a nivel mundial.

La Universidad de las Ciencias Informáticas (UCI) está llamada a convertirse en una potencia informática de desarrollo de software. En este sentido se trabaja por lograr que el trabajo en los proyectos productivos sea satisfactorio y de esta forma obtener productos con la calidad requerida y que cumplan con las expectativas de los clientes que los solicitan.

En la Universidad se desarrollan varios proyectos informáticos, uno de ellos es el proyecto Ensayos Clínicos (EC) de la facultad 6, en colaboración con el Centro de Inmunología Molecular (CIM), cuyo

propósito es la informatización de la gestión de los EC cubanos. El CIM tiene un alto reconocimiento internacional por su amplia labor en la obtención y producción de nuevos biofármacos, destinados al tratamiento del cáncer y otras enfermedades crónicas no transmisibles e introducirlos en el Sistema de Salud Pública Cubana. Uno de los problemas claves por los que atraviesa el CIM y que además no permite un avance en la conducción rápida de los EC es la inexistencia de una herramienta electrónica capaz de realizar los procesos asociados a dichos ensayos. Actualmente, los EC en el CIM y en Cuba se desarrollan de la manera clásica, toda la información se encuentra en papel, y la transmisión de información de los hospitales a los centros promotores y viceversa se hace generalmente a través del correo electrónico o telefónicamente.

Para solucionar esta problemática, el proyecto EC se ha trazado como objetivo: el desarrollo del sistema alasClínicas para el cual se realizó un estudio de varios sistemas de manejo de datos de EC en el mundo como son: Hipócrates, PhoscO, MACRO y OpenClinica. Posteriormente se tomó como punto de partida el software OpenClinica, el cual cumple con el 70 % de las funcionalidades para la gestión de los EC en nuestro país, siendo el mismo un sistema de código abierto y licencia Lesser General Public License (LGPL). [2]

OpenClinica está estructurado en cuatro módulos: Enviar Datos, Extraer Datos, Gestionar Estudio y Administrar Empresa. Cada uno de ellos brinda un grupo de funcionalidades que en general logran de manera eficiente, flexible y con bajo costo, que se gestionen varios estudios a la vez. Permite incluir pacientes al estudio, excluirlos, planificar sus momentos de seguimiento uno a uno, gestionar los datos de las hojas de CRD por momentos de seguimiento, así como otras funcionalidades. Según las necesidades del CIM, existen algunas funcionalidades que deben ser agregadas a este sistema y hay otras que deben ser cambiadas porque no cumplen con las políticas necesarias para la conducción de los EC en Cuba.

El desarrollo de alasClínicas implicará la incorporación y modificación de un conjunto de funcionalidades en el sistema OpenClinica, para lo cual convendrá lograr una evolución rápida ante los cambios realizados. Además el sistema deberá cumplir con un grupo de políticas de seguridad definidas por el cliente, y permitir entre otras características, facilidad de uso y rapidez de conexión desde diferentes hospitales del país.

2 LGPL es una licencia de software creada por la Free Software Foundation. Los contratos de licencia de la mayor parte del software están diseñados para jugar con su libertad de compartir y modificar dicho software.

Para el logro de estos requisitos y el desarrollo eficiente del sistema, alasClínicas deberá tener una serie de características como son: gran eficiencia, escalabilidad, usabilidad, rendimiento y seguridad, con el objetivo de que el sistema brinde un servicio eficiente y así darle cumplimiento a las expectativas del cliente.

Tomando en cuenta lo explicado anteriormente, se plantea como **Problema Científico** de la presente investigación**:**

¿Cómo lograr que alasClínicas sea un producto escalable, de gran rendimiento, usabilidad y que soporte la integración de nuevas funcionalidades para la gestión de los EC en el CIM?

La investigación tiene como **Objeto de Estudio**: *La arquitectura de software, enmarcado en el* **Campo de Acción**: *La arquitectura de software para el sistema alasClínicas.*

Para dar respuesta al problema científico se plantea como **Objetivo General:** *Definir una arquitectura de software que soporte las funcionalidades del sistema OpenClinica para contribuir a la gestión de los EC en el CIM.*

Para dar cumplimiento al objetivo general planteado se desglosan los siguientes **Objetivos Específicos:**

1. *Definir la línea base de la arquitectura para el sistema alasClínicas.*
2. *Describir la arquitectura de software del sistema alasClínicas.*
3. *Evaluar la arquitectura propuesta para el sistema alasClínicas.*

Se desarrollarán las siguientes **Tareas Científicas** para dar cumplimiento a los objetivos trazados:

1. *Estudio del sistema OpenClinica.*
2. *Estudio de las bibliografías existentes relacionadas con la AS teniendo en cuenta la tecnología J2EE para el desarrollo de aplicaciones web.*
3. *Estudio y selección de estilos y patrones arquitectónicos.*
4. *Estudio y definición de las tecnologías y herramientas a utilizar para el desarrollo del sistema.*
5. *Estudio de los métodos de evaluación de arquitecturas de software.*
6. *Definición de la AS con la tecnología J2EE para el sistema alasClínicas.*
7. *Diseño de las vistas arquitectónicas.*
8. *Representación del diseño arquitectónico.*
9. *Evaluación de la arquitectura diseñada a partir de uno de los métodos estudiados.*

Estructura del documento:

El trabajo de diploma está estructurado de la siguiente manera: introducción, tres capítulos, conclusiones, recomendaciones, referencias bibliográficas, bibliografía, anexos y un glosario de términos.

Capítulo 1: Fundamentación Teórica de la Arquitectura de Software. Este capítulo abarca todo el estudio de los temas relacionados con la AS como base teórica para fundamentar las decisiones tomadas, donde se analizan los principales conceptos asociados al objeto de estudio: AS, así como lo referente a los estilos arquitectónicos, patrones, lenguaje de descripción de arquitectura, la metodología de desarrollo utilizada y herramientas y tecnologías asociadas. Además aborda el estudio de los temas relacionados con la evaluación de la arquitectura.

Capítulo 2: Descripción de la Arquitectura de Software. Este capítulo comprende toda la descripción arquitectónica según los planteamientos de la metodología Rational Unified Process (RUP), detallando la misma a través de las vistas de la arquitectura.

Capítulo 3: Evaluación de la arquitectura propuesta. En este capítulo se muestra la evaluación de la arquitectura descrita en el capítulo anterior a través de uno de los métodos propuestos en la literatura, logrando así la validación de la AS definida.

Capítulo 1: Fundamentación Teórica de la Arquitectura de Software.

En el presente capítulo se hace un estudio profundo de los temas relacionados con la AS, se exponen conceptos que resultarán de vital importancia como son: patrones y estilos arquitectónicos, patrones de diseño, lenguajes y herramientas de modelado, además se abordará sobre la definición de AS que será seguida en el desarrollo, pues es un concepto muy controversial en la comunidad de la AS.

1.1 OpenClinica

OpenClinica es un software de captura electrónica de datos y de gestión de datos clínicos para EC y estudios de investigación **(1)**. Esta aplicación web está diseñada para dar cobertura a todo tipo de estudios clínicos, además está desarrollada a partir de los estándares de más prestigio para alcanzar altos niveles de interoperabilidad con otros servicios y plataformas. OpenClinica consta de 4 módulos: Administrar Empresa, Gestionar Estudio, Enviar Datos y Extraer Datos y cumple con el 70 % de las funcionalidades para el estudio de los EC cubanos.

Fue desarrollado y publicado bajo la licencia LGPL de código abierto, utilizando el marco de Java J2EE. El sistema se basa en un clásico patrón arquitectónico conocido como Modelo-Vista-Controlador o MVC con una capa de abstracción de bases de datos interoperables con 8.x de PostgreSQL y Oracle 10g. OpenClinica se ejecuta en Linux o Windows Server. Fue desarrollado para correr en Apache Jakarta Tomcat 5.x **(2)**.

La plataforma de software de OpenClinica permite:

- El manejo simultáneo de múltiples estudios (varios estudios multicéntricos o de un sólo centro) a través de una única interfaz unificada.
- La definición de los protocolos y del calendario de visitas, así como los eventos que se produzcan.
- La definición de los distintos elementos de información que componen los cuadernos de recogida de datos.
- La definición de los distintos usuarios que van a usar la plataforma, así como los permisos de acceso correspondientes.

- El filtrado y la extracción de datos (en cualquiera de los formatos más utilizados: SPSS, SAS, Excel).

1.2: Arquitectura de software

La AS es considerada un área de investigación y práctica dentro de la Ingeniería de Software. En particular, la arquitectura de sistemas grandes ha sido objeto de un interés creciente durante la pasada década. Este campo fue creado por necesidad, la realidad marcaba que los sistemas de software estaban creciendo, sistemas de cientos de miles de líneas, o incluso de millones de líneas se estaban volviendo comunes por lo que se hacía muy difícil su entendimiento y mantenimiento.

Existen muchas definiciones de AS y no parece que ninguna de ellas haya sido totalmente aceptada. Independientemente de las discrepancias entre las diversas definiciones, es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción.

Una definición reconocida es la de Clements **(3)**: “La arquitectura de software es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se le percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones.”

En “An Introduction to Software Architecture”, David Garlan y Mary Shaw **(4)** sugieren que la arquitectura es un nivel de diseño que se centra en aspectos: “Más allá de los algoritmos y estructuras de datos de la computación, el diseño y la especificación de la estructura general del sistema emergen como una clase nueva de problema. Los aspectos estructurales incluyen la estructura global de control y la organización general; protocolos de comunicación, sincronización y acceso de datos; asignación de funciones para diseñar elementos; distribución física, composición de elementos de diseño; ajuste y rendimiento; y selección entre otras alternativas de diseño”.

La definición más usada de AS es de la IEEE Std 1471-2000: “La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución” **(5)**.

En Rational Unified Process (RUP), la arquitectura de un sistema de software (en un punto determinado) es la organización o la estructura de los componentes importantes del sistema que interactúan mediante interfaces, con componentes compuestos de interfaces y componentes cada vez más pequeños.

Después de un análisis de los conceptos anteriores, para la presente investigación se asume la definición de AS propuesta por la IEEE Std 1471-2000.

1.3: Lenguajes de descripción arquitectónica

Los lenguajes de descripción de arquitecturas, o Architectecture Description Language, por sus siglas en inglés (ADLs), ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Los ADLs permiten modelar una arquitectura mucho antes que se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento. Existen ADLs de propósito general y otros de dominios específicos.

Los ADLs están diseñados específicamente para describir la arquitectura de los sistemas de software: componentes, que no son más que elementos computacionales y de datos descritos mediante los roles que juegan; los conectores, mecanismos de interacción flexibles y variados; y las configuraciones, combinación de componentes y conectores interconectados.

Entre los ADLs más destacados y usados se encuentran:

- UNICON (Shaw et al. 1995)
- Darwin (Magee y Kramer, 1996; 1999)
- Wright (Allen y Garlan, 1997; 1998)
- Executable Connectors (Ducasse y Richner, 1997)

Entre las características de estos lenguajes podemos considerar las siguientes: composición, que permiten la representación del sistema como la composición de una serie de partes. Configuración, abstracción, mediante la cual se describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura. Flexibilidad, pues permiten la definición de nuevas formas de interacción entre componentes. Reutilización, pues permiten la reutilización tanto de los componentes como de la propia arquitectura. Heterogeneidad pues pueden combinar descripciones heterogéneas, y por último Análisis, ya que se utilizan diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella **(6)**.

Existen muchas maneras de utilizar UML como un ADL aunque un gran número de arquitectos no lo consideran como tal.

En la siguiente tabla se muestra qué relación tiene cada punto de un ADL con los diagramas UML, y que le falta a UML para poder ser utilizado 100% como un ADL.

Tabla 1 Relación entre UML y ADL

Elemento de un ADL	**Relación con UML**
Componentes	Nodos, Componentes, Clases, Interfaces, Paquetes
Conectores	Puertos, Interfaces externas, relaciones
Configuraciones o sistemas	Diagramas de componentes e implementación
Propiedades	Las dependencias se pueden ver en el diagrama de implementación o empaquetamiento, y cada uno de los componentes tiene sus propiedades internas, aunque no exactamente como lo utilizaríamos.
Restricciones	No hay elemento UML que lo indique, salvo restricciones de dependencia.
Estilos	Está fuera de UML, son estilos arquitectónicos
Evolución	Refinamiento de diagramas
Propiedades no funcionales	No existe elemento UML que lo represente

Después de analizada la tabla anterior, para la descripción arquitectónica de alasClínicas se propone el uso de UML, debido a su uso habitual, soporte de las herramientas, conocimientos y procesos de los recursos.

1.4: Estilos arquitectónicos

Desde los inicios de la AS se les llamó estilos, por analogía con el uso del término en arquitectura de edificios. Un estilo describe entonces una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas.

En el texto fundacional de la AS, Perry y Wolf establecen el razonamiento sobre estilos de arquitectura como uno de los aspectos fundamentales de la disciplina. Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas

básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales **(7)**.

Mary Shaw y Paul Clements **(8)** identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo. Los componentes, incluyendo los subsistemas encapsulados, se pueden distinguir por la naturaleza de su computación: por ejemplo, si retienen estado entre una invocación y otra, y de ser así, si ese estado es público para otros componentes.

Los estilos que habrán de describirse a continuación no aspiran a ser todos los que se han propuesto en el tan amplio mundo de la AS, sino apenas los más representativos y vigentes.

Estilo 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, Vista y Controlador.

Estilos Centrados en Datos: Esta familia de estilos enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

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. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

Estilo Arquitecturas en Capas: Garlan y Shaw **(9)** 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.

Se propone utilizar en la presente investigación el estilo llamada y retorno, específicamente el sistema jerárquico en capas debido a que esta familia de estilos enfatiza la modificabilidad, la escalabilidad, la reusabilidad y la usabilidad del sistema.

1.5: Patrones

Los patrones son formas de describir las mejores prácticas, buenos diseños, y encapsulan la experiencia de forma tal que es posible para otros el reutilizar dicha experiencia. Constituyen mecanismos cuyo objetivo es la solución de problemas que ocurren repetidamente dentro de un contexto muy bien definido. Las soluciones que son propuestas a través de patrones involucran algunas clases de estructuras que permiten contemplar los requisitos no funcionales.

Los patrones son los que definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades, también tienen una serie de directivas para organizar los componentes del mismo sistema, con el objetivo de facilitar la tarea del diseño de tal sistema. **(10)**

Son soluciones generales a problemas comunes, además describen parte del sistema. Entre los principales patrones se encuentran: **(11)**

Patrones de arquitectura: relacionados a la interacción de objetos dentro o entre los niveles arquitectónicos, se aplican durante la fase de diseño inicial para resolver problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance (rendimiento), modularidad y acoplamiento. Constituyen patrones de llamada entre objetos, similar a los patrones de diseño, decisiones y criterios arquitectónicos.

Patrones de diseño: se aplican durante la fase de diseño detallado para solucionar problemas de claridad del diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes.

Patrones de análisis: usualmente específicos de aplicación, se aplican durante el análisis para solucionar problemas relacionados con el modelo de dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes.

1.5.1: Patrones arquitectónicos

En 1996, Mary Shaw bosquejó algunos patrones arquitectónicos que se asemejan parcialmente a algunos estilos. En esta oportunidad, un patrón arquitectónico se define por:

(1) un modelo de sistema que captura intuitivamente la forma en que están integrados los elementos.
(2) componentes.
(3) conectores que establecen las reglas de la interacción entre los componentes.

(4) una estructura de control que gobierna la ejecución. Entre los patrones referidos de este modo se encuentran la línea de tubería (pipeline), el patrón arquitectónico de abstracción de datos, los procesos comunicantes a través de mensajes, los sistemas de invocación implícita, los repositorios, los intérpretes, programa principal/subrutinas y sistemas en capas **(12)**.

Entre los principales patrones arquitectónicos se encuentran:

Patrón arquitectónico Vista de Plantilla: La idea básica de la vista de plantilla es incrustar marcadores en una página HTML cuando esta es escrita. Cuando la página se utiliza para solicitar un servicio, los marcadores se sustituirán por los resultados obtenidos de acuerdo con el servicio que brinda la página, como por ejemplo, el resultado de una consulta obtenida en una base de datos.

Patrón arquitectónico Página de Control: La idea básica detrás de una página de control es tener un módulo en el servidor web, dígase un controlador para cada página en el sitio web. En la práctica, no funciona exactamente un controlador por página, ya que a veces se le puede realizar un enlace a cualquier página dinámica mediante alguna función de información dinámica.

Patrón arquitectónico Modelo Vista Controlador

Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes **(13)**:

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.

Durante el estudio realizado se escogió el patrón arquitectónico Modelo-Vista-Controlador, que forma parte del estilo arquitectónico de llamada y retorno. En dicho patrón cada una de las partes son independientes, la comunicación entre ellas es mediante interfaces, que abstraen sus estructuras internas. Esto permite desarrollar el modelo, la vista y el controlador de forma independiente, así como realizar modificaciones en sus partes, sin afectar a las demás.

1.5.2: Patrones de diseño

Un patrón de diseño constituye un esquema para refinar subsistemas o componentes. Es una descripción de clases y objetos comunicándose entre sí adaptada para resolver un problema de diseño general en un contexto particular. Identifica: clases, instancias, roles, colaboraciones y la distribución de responsabilidades, además de que ayuda a construir clases y a estructurar sistemas de clases.

Los patrones de diseño son abstracciones de alto nivel que documentan soluciones de diseño exitosas. Son fundamentales para reutilizar el diseño en el desarrollo orientado a objetos **(14).**

Patrones GRASP

GRASP es un acrónimo que significa General Responsability Assignment Software Patterns (Patrones Generales de Software para Asignar Responsabilidades). El nombre se eligió para indicar la importancia de captar estos principios, si se quiere diseñar eficazmente el software orientado a objetos.

Patrón Experto

Surge como principio fundamental que hay que tener en cuenta siempre cuando se esté asignando una responsabilidad a una clase. La respuesta es asignar la responsabilidad a la clase que contenga la información necesaria para cumplir la responsabilidad, o sea, la clase debe ser la experta en la información. La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos).

Patrón Creador

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. Por tanto ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

Este patrón es muy simple y su mayor beneficio es que contribuye a soportar el bajo acoplamiento, lo cual supone menos dependencias respecto al mantenimiento y mejores oportunidades de reutilización.

Patrón Bajo Acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está relacionada a otras clases. Una clase con bajo o débil acoplamiento no depende de muchas otras. Por el contrario, una clase con alto o fuerte acoplamiento recurre a muchas otras. Este tipo de clases no es conveniente, porque un cambio en las

clases que utiliza ocasionaría cambios locales en la clase dependiente, además son más difíciles de entender cuando están aisladas y son más difíciles de reutilizar porque se requiere la presencia de otras clases de las que dependen.

Patrón Alta Cohesión

Cada elemento del diseño debe realizar una labor única dentro del sistema, lo cual expresa que la información que almacena una clase debe de ser coherente y está en la mayor medida de lo posible relacionada con la clase. . La solución es asignar a una clase responsabilidades que trabajen sobre una misma área de la aplicación y que no tengan mucha complejidad.

Patrón Controlador

¿Quién debería encargarse de atender un evento del sistema?

Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.

Asignar la responsabilidad del manejo de mensajes de los eventos del sistema a una clase que represente alguna de las siguientes opciones:

- El sistema global.
- La empresa u organización global.
- Algo activo en el mundo real que pueda participar en la tarea.
- Un manejador artificial de todos los eventos del sistema de un caso de uso (controlador de casos de uso)

Además, existen cuatro patrones GRASP adicionales:

Fabricación Pura.

Polimorfismo.

Indirección.

No hables con extraños.

Patrones GOF

El catálogo de patrones más famoso es el contenido en el libro ¨Design Patterns: Elements of Reusable Object-Oriented Software¨, el de la banda de los 4, también conocido como el LIBRO GOF (Gang-Of-Four Book). Según el libro GOF estos patrones se clasifican según su propósito en creacionales, estructurales y de composición, mientras que respecto a su ámbito se clasifican en clases y objetos.

A continuación se relacionan algunos de los patrones GOF que serán utilizados durante el desarrollo de la presente investigación:

- **Abstract Factory (Fábrica abstracta):** Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando. Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- **Iterator (Iterador**): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos. Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.
- **Template Method (Método plantilla):** Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- **Composite View (Vista compuesta)**: Es un patrón de diseño en el cual un objeto vista está compuesto de otros objetos vista o a su vez se le han agregado otros objetos subvistas. Por ejemplo, una página JSP que incluye otras JSP usando la directiva include o el action include es un ejemplo de la aplicación del patrón Composite View.
- **Data Access Object, DAO de sus siglas en inglés (Objeto de Acceso a Datos):** Consiste en utilizar un objeto de acceso a datos para abstraer y encapsular todos los accesos a la fuente de datos. El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos.

Existe una amplia gama de patrones de diseño propuestos para ser utilizados en el desarrollo de aplicaciones Web. Luego de realizar el análisis y diseño de la aplicación se proponen utilizar los siguientes patrones: Experto, Creador, Bajo Acoplamiento, Alta cohesión, Controlador, Abstract Factory, Iterator, Template Method, DAO y Composite View. En el próximo capítulo se detallará de forma minuciosa su aplicación en el sistema.

1.6 Metodología de desarrollo

El Proceso Unificado de Desarrollo RUP es una metodología **(15)** para la ingeniería de software que va más allá del mero análisis y diseño orientado a objetos para proporcionar una familia de técnicas que soportan el ciclo completo de desarrollo de software. Propone una metodología iterativa e incremental,

basada en componentes, dirigida por casos de uso, centrado en la arquitectura, que va eliminando los errores cometidos en las iteraciones previas. Logrando que a su culminación se obtenga como resultado un producto de calidad muy acorde con las necesidades y con la naturaleza cambiante de los requisitos en muchos proyectos. Utiliza el lenguaje unificado de modelado (UML). Las fases por las que está conformado son cuatros y se mencionan a continuación:

1. Inicio (puesta en marcha).
2. Elaboración (definición de la arquitectura).
3. Construcción (implementación).
4. Transición (fin del proyecto y puesta en producción).

En RUP se agrupan las actividades en grupos lógicos definiéndose 9 flujos de trabajo principales, que son: Modelado del negocio, Requerimientos, Análisis y diseño, Implementación, Prueba, Instalación, Administración de configuración y cambios, Administración del proyecto y Ambiente.

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

La arquitectura en RUP se representa mediante un número diferente de vistas de la arquitectura, que, esencialmente, son extractos que ilustran los elementos "significativos arquitectónicamente" de los modelos. Se empieza en un conjunto típico de vistas, llamado el "modelo de vista 4+1", en otras palabras el modelo de las 4+1 vistas propuestas por Kruchten en 1999 **(16)**.

Debido a esto es considerada una de las que mejor se ajusta a las necesidades para el desarrollo de software: por ser un proceso capaz de ser aplicado a cualquier proyecto sin importar su magnitud, lo que permite adaptarse a cada proyecto, incluidos los que no sean sólo de software. Por todo lo antes expuesto se selecciona como metodología de desarrollo para el sistema alasClínicas dicha metodología.

1.7 Responsabilidades del arquitecto de software

El rol del arquitecto de software dentro de un proyecto de desarrollo de software es dirigir el desarrollo de la AS del sistema, que incluye la promoción y la creación de soporte para las decisiones técnicas claves que restringen el diseño global y la implementación para el proyecto.

El arquitecto de software tiene la responsabilidad global de dirigir las principales decisiones técnicas, expresadas como la AS. Esto habitualmente incluye la identificación y la documentación de los aspectos arquitectónicamente significativos del sistema, que incluye Línea base de la arquitectura, Documento de Descripción de Arquitectura de Software, Informe del Levantamiento de Información para la Arquitectura de Información, Arquitectura de Información, Modelo de análisis, Modelo de diseño, Modelo de despliegue, Modelo de implementación, Prototipos de arquitectura, visualizar el comportamiento del sistema, crear los planos del sistema, definir la forma en la cual los elementos del sistema trabajan en conjunto, proveer una guía técnica clara y consistente. Además, establecer la estructura global de cada vista de la arquitectura: la descomposición de las vistas, el agrupamiento de elementos y las interfaces de los mismos.

La metodología RUP señala que los arquitectos moldean el sistema para darle una forma. Es esta la forma: la arquitectura, que debe diseñarse para permitir que el sistema evolucione, no sólo en su desarrollo inicial, sino también a lo largo de las futuras generaciones. Para encontrar esa forma, los arquitectos deben trabajar sobre la comprensión general de las funciones clave, es decir, sobre los casos de uso clave del sistema.

1.8 Lenguajes, tecnologías, herramientas de desarrollo y herramientas CASE

Uno de los aspectos más importantes a la hora de construir un software es la selección de las tecnologías, lenguajes y herramientas, las que sean más convenientes y beneficiosas según las características del sistema. En el caso de la presente investigación, las herramientas a utilizar para el desarrollo de las nuevas funcionalidades según las investigaciones realizadas antes al sistema OpenClinica son las siguientes:

J2EE (Java 2 Enterprise Edition)

J2EE incluye Enterprise JavaBeans, servlets, JavaServerPages y varias tecnologías de Web Services. Esto permite al desarrollador crear una Aplicación de Empresa portable entre plataformas y escalable, a la vez

integrable con tecnologías anteriores. Otros beneficios añadidos son, por ejemplo, que el servidor de aplicaciones puede manejar transacciones, la seguridad, escalabilidad, concurrencia y gestión de los componentes desplegados, significando que los desarrolladores pueden concentrarse más en la lógica de negocio de los componentes en lugar de tareas de mantenimiento de bajo nivel. Como lenguaje de programación Java puede utilizarse para realizar aplicaciones sobre distintos sistemas operativos, ya sea Windows, Linux o Unix.

Eclipse (Versión 3.4):

Eclipse es un Entorno de Desarrollo Integrado, multiplataforma, gratuito y de código abierto, que puede ser utilizado para varios lenguajes Java, C, C++, PHP, desarrollado originalmente por IBM y usado para crear entornos integrados de desarrollo. Para Eclipse existen diversos plugins o añadidos para proveer de nuevas utilidades al programa, enfocadas a diversos usos que los distintos tipos de programadores pueden necesitar, te permite el control de versiones con Subversión, compilación en tiempo real, presenta además estructura de plug-in así como posibilita añadirle nuevas características y funcionalidades.

Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código.

Apache Tomcat (Versión 5.5):

Tomcat es un servidor Web gratis, fácil de instalar con soporte de servlets y JSP´s (Java Server Pages), muchas empresas lo emplean en la actualidad en entornos de producción debido a su contrastada estabilidad. Incluye el compilador llamado Jasper, que compila las JSP´s y las convierte en servlets. Se ejecuta en máquinas más pequeñas y es compatible con las API más recientes de Java, fue escrito en Java por lo que funciona en cualquier sistema operativo que tenga instalada la Maquina Virtual de Java. Tomcat ocupa muy poco espacio, teniendo su código binario un tamaño total de apenas un megabyte, de ahí la rapidez de su ejecución. La solidez de Tomcat se basa en que miles de desarrolladores contribuyen con su código.

Gestor de Base de datos PostgreSQL (Versión 8.2.4):

PostgreSQL es un Sistema de Gestión de Bases de Datos Objeto-Relacionales (ORDBMS). PostgreSQL está considerado como la base de datos de código abierto más avanzada del mundo. La siguiente es una breve lista de algunas de esas características:

- **DBMS Objeto-Relacional:** PostgreSQL aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad son consultas SQL declarativas, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, herencia, y arrays.
- **Altamente Extensible:** PostgreSQL soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario.
- **Integridad Referencial:** PostgreSQL soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos de la base de datos.
- **Lenguajes Procedurales**: PostgreSQL tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.
- **Cliente/Servidor:** PostgreSQL usa una arquitectura proceso-por-usuario cliente/servidor. Esta es similar al método del Apache 1.3.x para manejar procesos. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

Herramienta para el Control de Versiones: Subversion (Versión 1.6)

Subversion es un sistema de control de versiones libre y de código fuente abierto diseñado específicamente para reemplazar al popular CVS (Concurrent Versions System), el cual posee varias deficiencias. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. El repositorio es como un servidor de ficheros ordinario, excepto porque recuerda todos los cambios hechos a sus ficheros y directorios. Esto le permite recuperar versiones antiguas de sus datos, o examinar el historial de cambios de los mismos. Este control de versiones puede acceder al repositorio a través de redes, lo que le permite ser usado por personas que se encuentran en distintos ordenadores. Brinda la capacidad para que varias personas puedan modificar y administrar el mismo conjunto de datos desde sus respectivas ubicaciones fomentando de esta forma la colaboración (**17**).

Eclipse (Versión 3.4) es el cliente que se propone para conectarse al servidor de Subversion previamente explicado.

1.8.1 Herramientas CASE

Existen herramientas creadas para el desarrollo de la ingeniería de software como son las herramientas CASE (Computer Aided Software Engineering), que constituyen un conjunto de aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software. Las herramientas CASE se acoplan con las metodologías para dar una forma de representar sistemas y suponen una forma de abstracción del código fuente, a un nivel donde la arquitectura y el diseño se hacen más aparentes y fáciles de entender y modificar. Cuanto mayor es un proyecto, más importante es el uso de tecnología CASE. Entre las herramientas CASE orientadas a UML están: ArgoUML, Poseidon for UML, MagicDraw UML, Visual Paradigm y Borland Together. Entre las herramientas para el modelado más utilizadas resaltan: Rational Rose, Visual Paradigm y Enterprise Architect.

Rational Rose Enterprise Edition.

Es la herramienta que permite construir y desarrollar a través del UML los casos de uso en diferentes diagramas, modelando los flujos de trabajo por los que transita el desarrollo de un software. Permite la realización de los diferentes diagramas y la posterior generación del código, todo orientado a objetos. Posee librerías que facilitan la obtención de una ingeniería inversa sobre diferentes lenguajes como Java, C++, Corba, XML_DTD, ADA, Visual Basic. Esta herramienta posibilita gestionar la evolución del ciclo de vida de un proyecto de software. Racional Rose aligera la implementación al automatizar los modelos arquitectónicos. Además, permite visualizar, entender, y refinar los requerimientos y arquitectura antes de introducirse en el código **(18)**.

Visual Paradigm Suite for UML 6.1

Es una herramienta multiplataforma, permitiendo su uso en cualquier sistema operativo. Utiliza UML como lenguaje de modelado, permitiendo una rápida construcción de las aplicaciones con alta calidad. Permite dibujar diagramas de clases, y generar script para diferentes Sistemas Gestores de Bases de Datos. Permite una integración con sistemas de control de versiones que almacenan centralmente los artefactos y realizan un seguimiento de los cambios realizados sobre un proyecto. Los desarrolladores lo utilizan para facilitar el modelado simultáneo, almacenar los archivos de proyectos y hacer un seguimiento de los cambios **(19).**

Luego de analizadas las herramientas informáticas antes mencionadas, se selecciona como Herramienta CASE para dar soporte al desarrollo del sistema alasClínicas a Visual Paradigm Suite for UML 6.1 (VP-UML).

1.8.2 Lenguaje Unificado de Modelado UML

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

El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar. UML incluye conceptos semánticos, notación, y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. La especificación de UML no define un proceso estándar pero está pensado para ser útil en un proceso de desarrollo iterativo. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos **(20**).

Se propone utilizar UML como lenguaje de modelado para la descripción de la arquitectura de alasClínicas, pues es el lenguaje más utilizado para llevar a cabo esta tarea.

1.9 Métodos de evaluación de la arquitectura

El primer paso para la evaluación de una arquitectura es conocer qué se quiere evaluar. De esta forma, es posible establecer la base para la evaluación, puesto que la intención es saber qué se puede evaluar y qué no. Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar las decisiones arquitectónicas con respecto a su impacto sobre dichos atributos. Cuanto más temprano se encuentre un problema en un proyecto de software, mejores serán los resultados al momento de su despliegue evitando de forma económica mayores inconvenientes. La evaluación temprana no tiene que esperar a que la arquitectura esté totalmente especificada por lo que puede ser utilizada en cualquier etapa del proceso de creación de la arquitectura mientras que la evaluación tardía toma lugar no sólo cuando la arquitectura está terminada, también cuando la implementación está completa. La evaluación de una arquitectura no produce resultados cuantitativos, en cambio, ayuda a encontrar debilidades y fortalezas.

Existen múltiples métodos para realizar pruebas a la AS, cada uno con sus características específicas. Es necesario tener en cuenta las características fundamentales del software para escoger la ideal, teniendo en cuenta las fortalezas y debilidades de la misma. Entre los principales métodos se encuentran: SAAM (Software Architecture Analysis Method), ARD (Active Design Review), ATAM (Architecture Tradeoff Analysis Method) y ARID (Active Review Intermediate Designs).

ADR

El método de Revisión de Diseño Activo o ARD (Active Design Review) se emplea en la evaluación de diseños bien detallados de unidades de software como los componentes. Se centra en la calidad y el nivel de completamiento de la documentación, y en el nivel de conveniencia y suficiencia de los servicios que contiene el diseño propuesto **(21)**.

ATAM

ATAM pretende ser un método de identificación de riesgo, un medio de detectar áreas de riesgo potencial dentro de la arquitectura de un sistema intensivo de software complejo. Revela la forma en que una arquitectura satisface ciertos atributos de calidad, provee una visión de la forma que interactúan estos atributos con otros. ATAM se inspira en las áreas de estilos arquitectónicos, análisis de atributos de calidad y el método de evaluación SAAM.

ATAM se aplica cuando la arquitectura presenta un alcance definido y manejable, pues el equipo evaluador necesita conocer los objetivos del negocio, las decisiones arquitectónicas tomadas para lograr los atributos de calidad especificados, vistas de componentes y conectores

El análisis de este método producirá resultados en consonancia con el nivel de detalle de la especificación de la arquitectura. Además, no es necesario producir análisis detallados de cualquier atributo de calidad del sistema (como el tiempo medio de retardo o el tiempo de fallo) para tener éxito **(22)**. ATAM consta de nueve pasos, divididos en cuatro fases.

Tabla 2 Pasos del método de evaluación ATAM

Fase 1:Presentación

Paso 1. Presentación del ATAM	El equipo de evaluación presenta un panorama general de los pasos de ATAM y trata de establecer las expectativas de los resultados del proceso.
Paso 2. Presentación de las metas del negocio	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico. Se presenta brevemente el negocio y el contexto de la arquitectura.
Paso 3. Presentación de la arquitectura	El arquitecto presenta un panorama de la arquitectura, describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
Paso 4. Identificación de los enfoques arquitectónicos	El equipo de evaluación y el arquitecto deben detallar los planteamientos arquitectónicos descubiertos en el paso anterior. Estos elementos son detectados, pero no analizados.
Paso 5. Generación del Utility Tree	Se identifican, priorizan, definen y refinan los atributos de calidad más importantes en un formato de árbol de utilidad y que engloban la "utilidad" del sistema, especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
Paso 6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance. Se utilizan las preguntas similares a las presentadas en el paso 5.
Fase 3: Pruebas	
Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios y se le da prioridad a los escenarios sobre la base de su importancia relativa.
Paso 8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Fase 4: Reporte	
Paso 9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes. El equipo de evaluación recapitula los pasos de ATAM, resultados, y recomendaciones.

Al finalizar el desarrollo del método se obtienen los siguientes resultados:

- El documento de propuestas arquitectónicas.
- El conjunto de escenarios priorizados.
- El conjunto de preguntas basadas en los atributos.
- El árbol de utilidad.
- Los riesgos descubiertos.
- Los no riesgos documentados.
- Los puntos de sensibilidad y de trade-off encontrados.

ARID

El método ARID es un híbrido del método ARD y ATAM. ARID constituye un método conveniente para la evaluación de diseños parciales en las etapas tempranas del desarrollo usando técnicas de evaluación basadas en escenarios. Utiliza para la evaluación del diseño unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto. Define los roles de Arquitecto, Equipo de verificación y Stakeholders y comprende nueve pasos agrupados en dos fases. El método ARID evalúa el grado en que los atributos de calidad satisfacen cada uno de los escenarios definidos. 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 **(23)**.

ARID consta de 9 pasos recogidos en 2 fases.

Tabla 3 Pasos del método de evaluación ARID

Fase 1: Actividades previas	
Paso 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.
Paso 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.
Paso 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.
Paso 4. Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase.
Fase 2: Revisión	
Paso 5. Presentación del ARID	Se explica los pasos del ARID a los participantes.
Paso 6. Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. El objetivo es verificar que el diseño es conveniente.
Paso 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. Se proponen escenarios, los cuales son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
Paso 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.
Paso 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.

Por las características de los métodos de evaluación de arquitectura explicados anteriormente, se decide utilizar ATAM debido a que permite realizar la evaluación temprana de los diseños de la arquitectura, además es el más documentado.

1.10 Atributos de calidad en la arquitectura de software

Para determinar los atributos de calidad que debe lograr la AS con la tecnología J2EE fue necesario guiarse por los objetivos y restricciones arquitectónicas que el negocio impone y por los modelos de calidad. Mediante estos atributos será posible saber que la arquitectura seleccionada es correcta evaluando el resultado que han producido dichas decisiones sobre estos.

Entre los modelos de calidad más importantes propuestos se encuentran: McCall (1977), FURPS (1987), Dromey (1996) e ISO/IEC 9126, este último adaptado para arquitecturas de software propuesto por Losavio en el año 2003.

ISO/IEC 9126:

Este estándar ha sido desarrollado en un intento de identificar los atributos clave de calidad para un producto de software **(24) .**El mismo es una simplificación del Modelo de McCall **(25)**, e identifica seis características básicas de calidad que pueden estar presentes en cualquier producto de software, las mismas se relacionan a continuación.

Funcionalidad, Confiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad.

ISO/IEC 9126 adaptado para arquitecturas de software:

Losavio **(25)** en el año 2003 propone una adaptación del modelo ISO/IEC 9126 del año 2001 de calidad de software para efectos de la evaluación de AS. El modelo se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

La tabla que se muestra a continuación presenta los atributos de calidad planteados por Losavio **(24)** que poseen subcaracterísticas asociadas con elementos de tipo arquitectónicos.

Tabla 4 Atributos de calidad - Modelo ISO/IEC 9126 adaptado para arquitecturas de software.

Característica	Subcaracterísticas	Elementos de tipo arquitectónico

Funcionalidad	Adecuación	Refinamiento de los diagramas de secuencia
	Exactitud	Identificación de los componentes con las funciones responsables de los cálculos
	Interoperabilidad	Identificación de conectores de comunicación con sistemas externos
	Seguridad	Mecanismos o dispositivos que realizan explícitamente la tarea
Confiabilidad	Tolerancia a fallas	Existencia de mecanismos o dispositivos de software para manejar excepciones
	Recuperabilidad	Existencia de mecanismos o dispositivos de software para restablecer el nivel de desempeño y recuperar datos
Eficiencia	Desempeño	Componentes involucrados en un flujo de ejecución para una funcionalidad
	Utilización de recursos	Relación de los componentes en términos de espacio y tiempo
Mantenibilidad	Acoplamiento	Interacciones entre componentes
	Modularidad	Número de componentes que dependen de un componente
Portabilidad	Adaptabilidad	Presencia de mecanismos de adaptación
	Instalabilidad	Presencia de mecanismos de instalación
	Coexistencia	Presencia de mecanismos que faciliten la coexistencia
	Reemplazabilidad	Lista de componentes reemplazables para cada componente

La presente investigación utilizará los atributos de calidad del modelo ISO/EC 9126 adaptado para arquitecturas de software para la evaluación del diseño arquitectónico debido a la correspondencia de dichos atributos con las necesidades del sistema.

1.11 Técnicas de evaluación

Las técnicas de evaluación de la arquitectura se clasifican en cualitativas (escenarios, cuestionarios y listas de chequeo) y cuantitativas (métricas, normas, máximos y mínimos teóricos, simulaciones, prototipos). Estos tipos de técnicas posibilitan evaluar una arquitectura y establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface más un atributo de calidad específico.

Se plantean diferentes técnicas de evaluación tanto cualitativas como cuantitativas, entre estas se pueden encontrar: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia **(26)**. A continuación se muestra una breve descripción de la técnica empleada para la evaluación de la arquitectura del sistema alasClínicas:

Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con el propio sistema. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El 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. El contexto describe qué sucede en el sistema al momento del estímulo. La respuesta describe, a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad.

Dentro de la evaluación por escenarios se utiliza lo que se conoce como el Utility Tree que no es más que 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. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol. El Utility Tree contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado **(26)**. Cada

atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

1.12 Conclusiones parciales del capítulo

En este capítulo se realizó un estudio detallado sobre los temas relacionados con la AS, mostrándose conceptos de gran importancia para el diseño de la arquitectura como: los patrones y estilos arquitectónicos, patrones de diseño, lenguajes y herramientas de modelado. Además, se escogió UML como el lenguaje de modelado. Estos aspectos sirven de introducción a las funciones del capítulo 2 y los procedimientos de evaluación de la arquitectura del capítulo 3.

Capítulo 2: Descripción de la Arquitectura de Software.

En el presente capítulo se encontrará íntegramente la descripción de la AS con la tecnología J2EE que podría ser utilizada en el desarrollo de aplicaciones Web que utilicen esta tecnología, también se incluyen decisiones importantes acerca del estilo de codificación a seguir por los desarrolladores, los patrones arquitectónicos y los patrones de diseño propuestos para elevar la capacidad del diseño que se traduce en calidad del software.

2.1 Representación arquitectónica

La representación arquitectónica expuesta en este documento está guiada por el proceso de desarrollo RUP, que precisamente define la representación arquitectónica mediante 4+1 vistas: una vista de Casos de Uso, una vista Lógica, una vista de Despliegue, una vista de Implementación y una vista de Procesos. Señalar que en este documento no se representa la vista de procesos, debido a que en el sistema no existen procesos concurrentes significativos. Esas vistas están representadas mediante modelos de VP-UML y usa UML como lenguaje de modelado.

A fin de seguir con la filosofía del sistema OpenClinica, el sistema estará dividido en 4 módulos: Administrar Empresa, Gestionar Estudio, Enviar Datos y Extraer Datos.

El módulo **Administrar Empresa:** Permitirá gestionar los permisos y roles de los usuarios que accederán al sistema así como la gestión de centros.

El módulo **Gestionar Estudio:** Permitirá la creación de los EC, así como la confección del Cuaderno de Recogida de Datos (CRD) y la creación del cronograma general de ejecución de un EC. Además, se podrá realizar la validación de las variables de un CRD a través del establecimiento de reglas de validación y derivación; permitiendo la disminución y detección de los errores que pueden cometerse al entrar los datos recogidos de los pacientes en el ensayo.

El módulo **Enviar Datos:** Permitirá generar el Cronograma de Ejecución específico para cada paciente dentro de un ensayo, así como el llenado de los modelos del CRD. Además, permitirá realizar el monitoreo de los datos de los pacientes de un sitio y la creación de consultas asociadas a cada uno de los datos monitoreados.

El módulo **Extraer Datos:** Permitirá realizar reportes a partir de los datos entrados de los pacientes en el ensayo y realizar un reporte de las trazas de auditoría asociado fundamentalmente a las variables de los modelos y a las acciones en general que realiza determinado usuario sobre los elementos del sistema. Garantiza la salida de información en diversos formatos de uso extendido: SPSS, EXCEL y HTML.

2.2 Objetivos y restricciones arquitectónicas

El producto, por estar enmarcado en un entorno de salud y tener como objetivo la gestión informática de los EC, debe seguir los reglamentos y estándares internacionales para este tipo de sistemas.

Las restricciones del sistema arquitectónicamente significativas están dadas porque:
1. Se deben recopilar los datos de varios EC que se encuentran distribuidos geográficamente por todo el país.
2. Los datos que suministran la mayoría de los EC están en copia dura, o sea, documentos impresos.
3. Los datos con los cuales trabaja el CIM son de vital importancia y de carácter estratégico para la gestión de EC del país por lo cual es necesario garantizar su seguridad.
4. Debe existir un repositorio central para almacenar todos los datos recopilados.
5. La cantidad de información que se maneja es muy grande.

Además, el producto se ve afectado por los siguientes requisitos no funcionales, que influyen en el diseño de la arquitectura del sistema:

RNF de apariencia o interfaz externa

- Las páginas no tendrán muchas imágenes y poseerán pocos colores.
- Las páginas principales tendrán información que servirá de guía al usuario.
- Cada rol tendrá una interfaz diferente con las funciones que le corresponden.
- Se hará uso de simbología mediante íconos para indicar el estado de los elementos utilizados en el diseño. Además, los íconos contendrán funcionalidades específicas.
- Se hará uso de colores para especificar el estado de los modelos en el módulo Enviar Datos.

RNF de usabilidad

- Las personas que interactuarán con el software serán médicos, clínicos y especialistas de la salud ubicados en el CIM, Centro de Ingeniería Genética y Biotecnología (CIGB), Instituto Finlay, Centro Nacional Coordinador de Ensayos Clínicos (CENCEC) y todos los hospitales del país.
- La aplicación tendrá un ambiente sencillo y será fácil de manejar para los usuarios incluso aquellos que no han tenido mucha experiencia en el trabajo con computadoras o con sistemas informáticos.

RNF de Seguridad

- El acceso a cualquier manipulación del sistema, tanto entrada como análisis de datos debe estar sometido a un proceso de autenticación del usuario donde será especificado el rol, usuario y contraseña.
- Las contraseñas deberán tener más de 7 caracteres de longitud y tener una fortaleza media.
- Los usuarios estarán obligados a cambiar la contraseña cada 60 días como máximo, pueden cambiar todos los días si desean.
- Cada usuario tendrá asignado uno o varios roles en el sistema. Cada rol definido tendrá niveles de acceso al Software.
- Sólo podrán acceder a la aplicación, clientes a través de direcciones IP específicas bien controladas.
- Todo cambio o modificación en el sistema debe ser atribuible a un usuario particular según su autenticación.
- Paralelo a la base de datos primaria se debe mantener una base de datos que registre todas las modificaciones hechas a la base de datos original, ordenadas cronológicamente y con la especificación del usuario responsable de dicha modificación, de manera que siempre se realicen trazas a la información manejada.
- Se debe garantizar comunicaciones seguras entre los clientes y el servidor, encriptando todo el tráfico de información con la utilización de llaves negociadas, algoritmos y protocolos.

RNF de Extensibilidad

- Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

RNF de Disponibilidad

- Se garantizará el funcionamiento de la aplicación durante las 24 horas del día y los siete días de la semana con el menor tiempo posible de recuperación de fallos.
- El servidor de aplicación debe soportar un aumento de usuarios concurrentes por minuto de 1 a 400.

2.3 Plataforma tecnológica

Una de las buenas prácticas en el desarrollo de la AS es definir la plataforma tecnológica a utilizar la cual constituye un conjunto de herramientas de hardware y de software, los cuales deben ser restringidos para el diseño e implementación del sistema.

Sistema operativo

El sistema se desarrolla sobre el sistema operativo GNU Linux, Ubuntu. Este sistema operativo es código abierto y es libre en todas sus versiones, lo que lo hace muy aplicable para el desarrollo de software en Cuba. Es un sistema muy robusto, confiable y adaptable al proyecto a desarrollar.

Seguridad (Antivirus, Niveles de Acceso a código fuente, documentación)

Los niveles de acceso al código fuente y a la documentación se realizan desde un cliente, donde los mismos se conectan al servidor del proyecto mediante el Subversion por un protocolo seguro y bajo autenticación.

Gestión de recursos (Levantamiento tecnológico, asignación de recursos)

El proyecto está compuesto por 24 integrantes de los cuales 6 son profesores y un total de 21 PC, distribuidas en un laboratorio de las cuales una es el servidor.

Control de versiones

Para el control de versiones se utiliza el Subversion. Tiene una fuerte integración con el servidor Web Apache, lo que posibilita definir controles de acceso avanzados y navegación vía web para consultar el depósito de archivos.

Gestión documental

Para la gestión documental se utiliza el Subversion.

Seguimiento de errores

Para el seguimiento de errores se utiliza el Trac. Es una herramienta de gestión de proyectos, principalmente de software, que unifica un sistema wiki con un sistema de seguimiento (issue tracking) con claras ventajas a la hora de trabajar con un repositorio de Subversion por su fuerte integración con este último.

Herramientas de modelado

Como herramienta CASE para dar soporte al desarrollo del sistema a Visual Paradigm Suite for UML 6.1.

Compilación (Compilador, Maquina Virtual, Intérprete)

Como compilador de java se utiliza la Maquina Virtual de Java (jdk-6u3). Es un conjunto de programas y librerías que permiten desarrollar (compilar, ejecutar, generar documentación) programas en lenguaje Java. Se puede instalar el Java Development Kit (JDK) en sistemas tipo UNIX, Solaris y Windows, en arquitecturas x386, x86-64 e Intel Itanium lo que la hace multiplataforma.

Herramientas de desarrollo

Como Entorno de Desarrollo Integrado (IDE) se utiliza el Eclipse3.4, con plugin para conexión al servidor de Subversion.

Lenguajes de programación

Como lenguaje de programación se utiliza Java con sus diferentes objetos como son:

- Los servlets que permiten que las solicitudes de los usuarios estipulen el procesamiento (el acceso a las bases de datos a través de la conectividad de la base de datos de Java (JDBC), Java Database Connectivity), la comunicación con otros servlets gracias a la tecnología RMI (Java Remote Method Invocation) y el acceso a LDAP (Lightweight Directory Access Protocol, Protocolo Ligero de Acceso a Directorios).
- Las páginas JSP, que representan el código HTML donde el código Java es nombrado EJBs (Enterprise JavaBeans), que son componentes del servidor escritos en lenguaje Java y que se usan para acceder a sus métodos.
- El lenguaje en el que se basa J2EE es Java, un lenguaje orientado a objetos que alcanzó su madurez con la popularización de Internet y que es en cierta manera el heredero legítimo de C++. La expansión de este lenguaje entre la comunidad de programadores ha sido vertiginosa y se ha impuesto como el paradigma de los lenguajes de programación orientados a objetos.

Servidor de Aplicaciones.

Como servidor de aplicaciones Tomcat5.5. Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) es un Servidor Web con soporte de servlets y JSPs. Incluye el compilador Jasper, que compila JSPs convirtiéndolas en servlets. El motor de servlets de Tomcat a menudo se presenta en combinación con el servidor web Apache. Hoy en día es usado como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Sistema Gestor de Bases de Datos. (Servidor, Cliente, o Modelo propio de persistencia)

PostgreSQL 8.2.4: Dada las necesidades del producto (especificado en los requerimientos no funcionales), entre las que se destaca la gran cantidad de datos a almacenar (y la necesidad de estos, como mínimo, por 15 años dentro de la BD), hubo necesidad de seleccionar una solución que permitiera grandes almacenes de información pero que además no requiriera primariamente la utilización de muchos servidores (debido a las limitaciones del país, aprovechar toda la capacidad de almacenamiento que permitiera cada PC servidora). Además, la tecnología escogida debía ser totalmente libre y permitir una alta seguridad de la información manejada. La respuesta a las necesidades fue encontrada en el servidor de bases de datos PostgreSQL 8.2.4.

2.4 Vistas arquitectónicas

Siguiendo la metodología RUP tal y como se expuso en el Capítulo1, a continuación se desarrolla la descripción de la arquitectura a través de las 4+1 vistas propuestas por Kruchten.

2.4.1 Vista de casos de uso

La vista de casos de usos en el marco arquitectónico, representa los casos de usos arquitectónicamente significativos; seleccionados a partir de los casos de uso catalogados de críticos en cada uno de los módulos. Estos serían los casos de usos que modelan las principales funcionalidades del sistema de acuerdo con las exigencias del cliente.

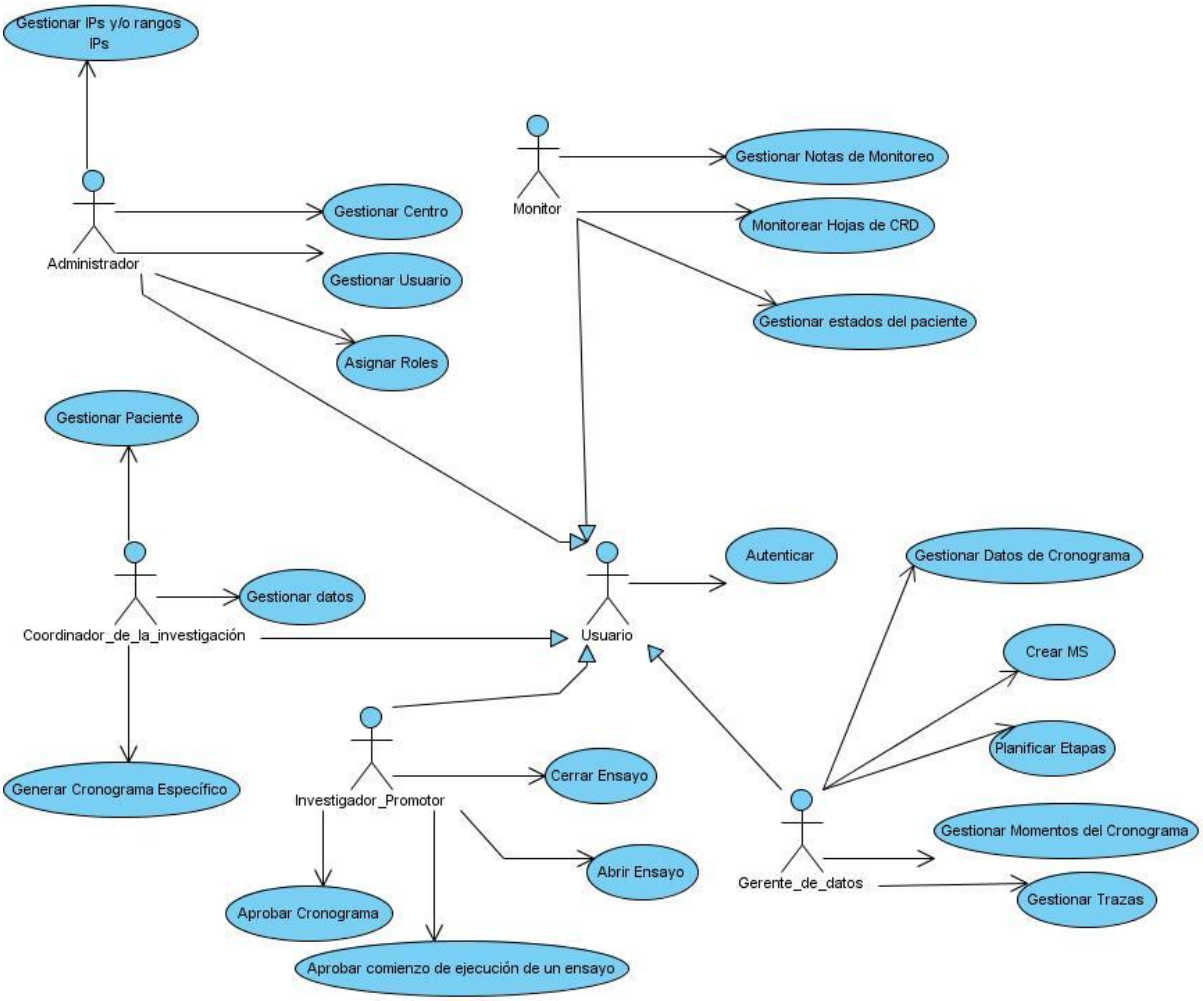


Fig. 1 Casos de Uso del Sistema

La descripción de los CU se encuentra en el artefacto Modelo de CU del Sistema en el Expediente de Proyecto del sistema alasClínicas.

2.4.2 Vista lógica

La vista lógica describe el diseño más importante de las clases y su organización en paquetes y subsistemas, y la organización de éstos en capas. También contiene algunas realizaciones de casos de uso. Ésta muestra cómo la funcionalidad es diseñada en el interior del sistema, en términos de la estructura estática y comportamiento dinámico del sistema.

Para el desarrollo del producto se escogió el patrón Modelo-Vista–Controlador. A continuación se presenta un diagrama con las capas diseñadas y sus relaciones, estas son: capa Vista, capa Controlador y la capa Modelo.

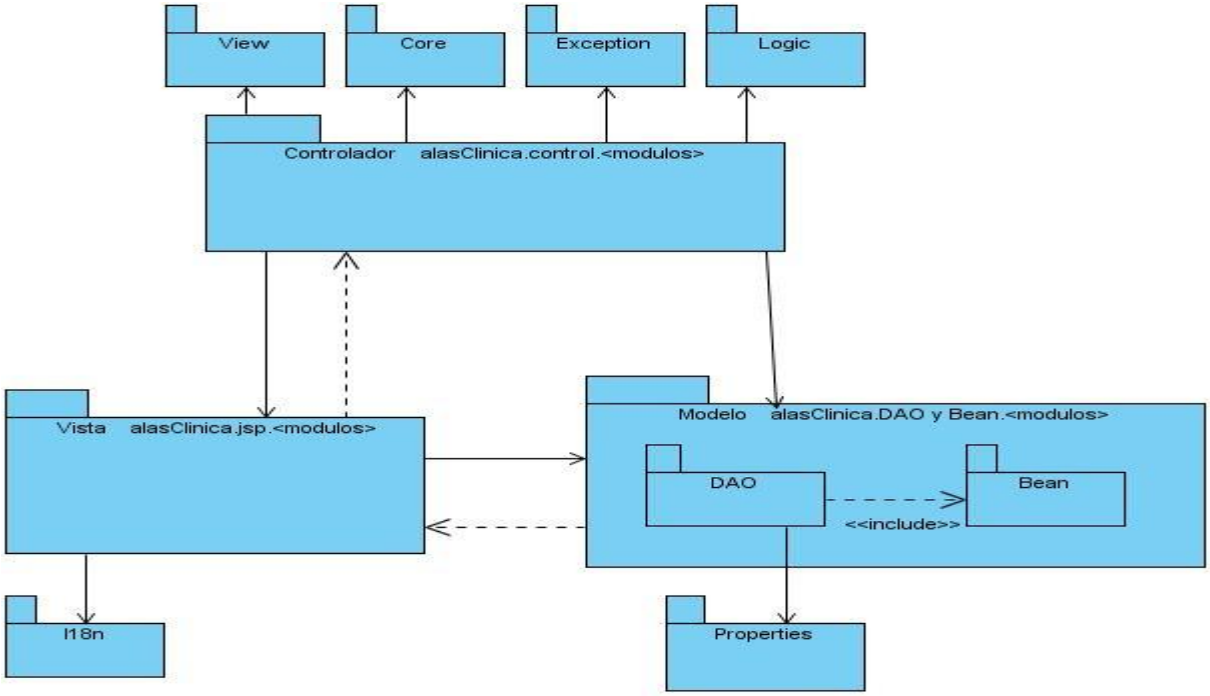


Fig. 2. Vista Lógica - Patrón Modelo Vista Controlador

Capa: Vista

La capa vista está compuesta por 6 paquetes, cada paquete está contenido dentro de la carpeta jsp. Cada uno contienen páginas con extensiones jsp encargadas de la presentación de la aplicación, las mismas se construyen a partir de una página servlet. Dicha capa utiliza el paquete i18n encargado de la internacionalización del idioma.

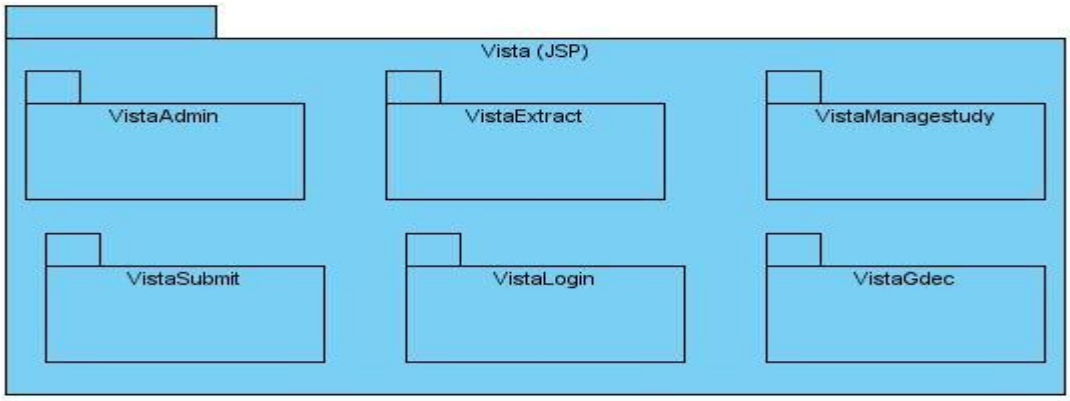


Fig.3. Vista Lógica - Capa Vista

Capa: Controlador

La capa controlador está compuesta por 6 paquetes. Cada uno contiene las páginas Servlet, encargadas de recibir petición de la vista, procesarla, entregárselas al modelo, y construir la vista según la respuesta. Esta capa utiliza los paquetes View, Core, Exception y Logic.

- View contiene las clases encargadas de la construcción de la vista.
- Core contiene las clases encargadas de la construcción de formularios y tablas, dentro del mismo se encuentra el paquete Form que contiene clases encargadas de las validaciones de los formularios.
- Exception es el paquete que contiene las clases encargadas del trabajo con las excepciones de errores.
- Logic es el paquete encargado de alguna lógica del negocio común que se utiliza en la capa controladora.

El paquete ControlCore contiene la clase encargada de la seguridad de la aplicación, como verificación de permisos y roles, tal es el caso de SecureController que es una clase abstracta donde se definen una serie de métodos que luego se deben redefinir, la misma hereda de la clase HttpServlet propia de Java; de SecureController heredan todos los Servlet que se encuentran en los restantes paquetes encargados de redefinir los métodos abstractos de dicha clase.

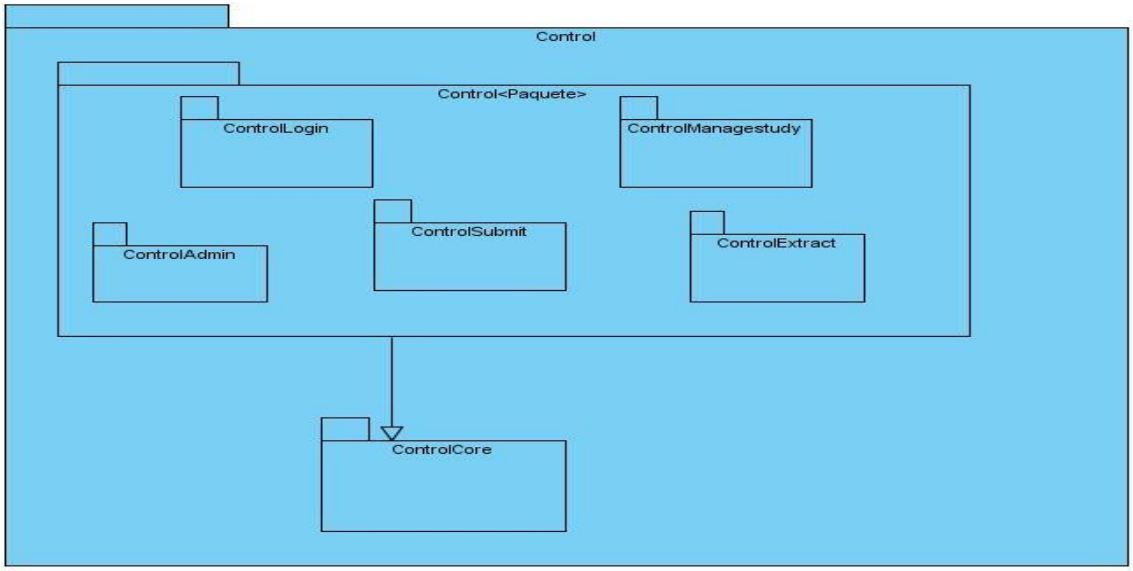


Fig.4. Vista Lógica - Capa Controlador

Capa: Modelo

La capa modelo está compuesta por 12 paquetes. 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 .La capa contiene 6 paquetes representando las clases DAO que heredan de clases del paquete DAOCore encargadas del acceso a la base de datos y los mismos hacen uso de los restantes 6 paquetes que son las clases Bean que heredan de clases del paquete BeanCore como representación de las entidades del negocio.

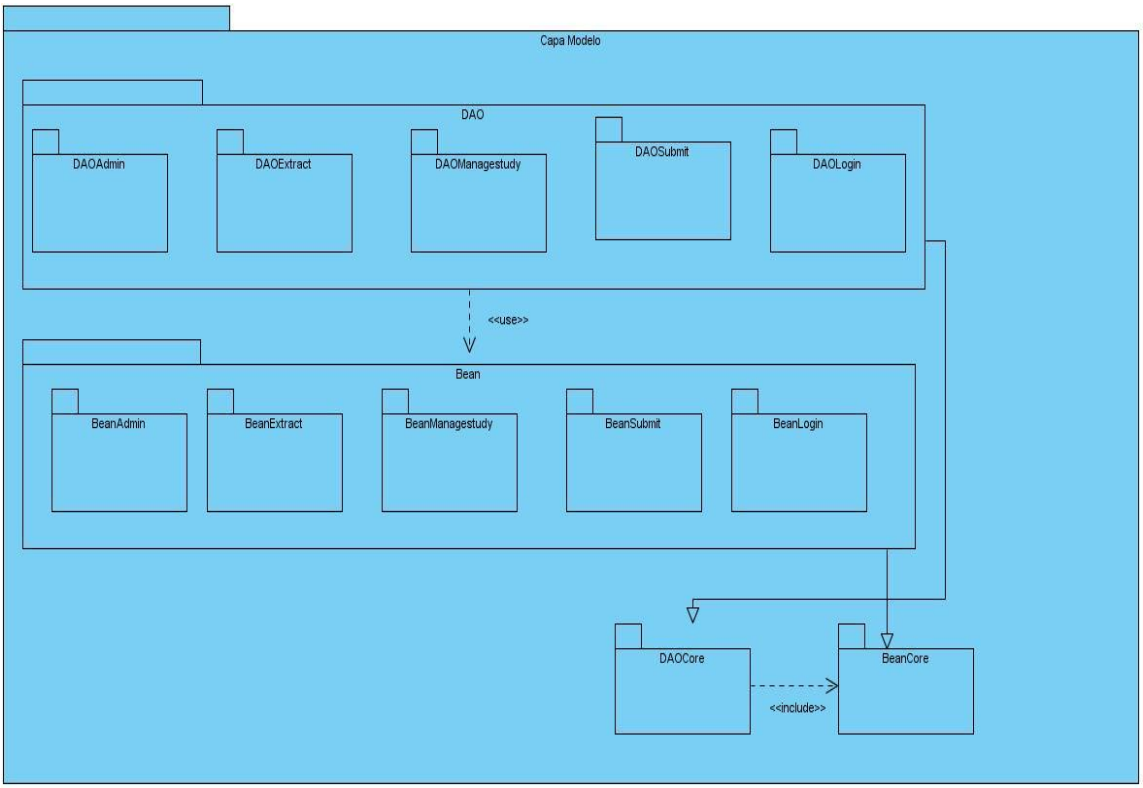


Fig.5. Vista Lógica - Capa Modelo

2.4.3 Vista de despliegue

Muestra la distribución física de los procesos del sistema (ordenadores, dispositivos) y sus conexiones. Esta vista es un grafo de nodos unidos por conexiones de comunicación, la misma es perfeccionada durante las iteraciones.

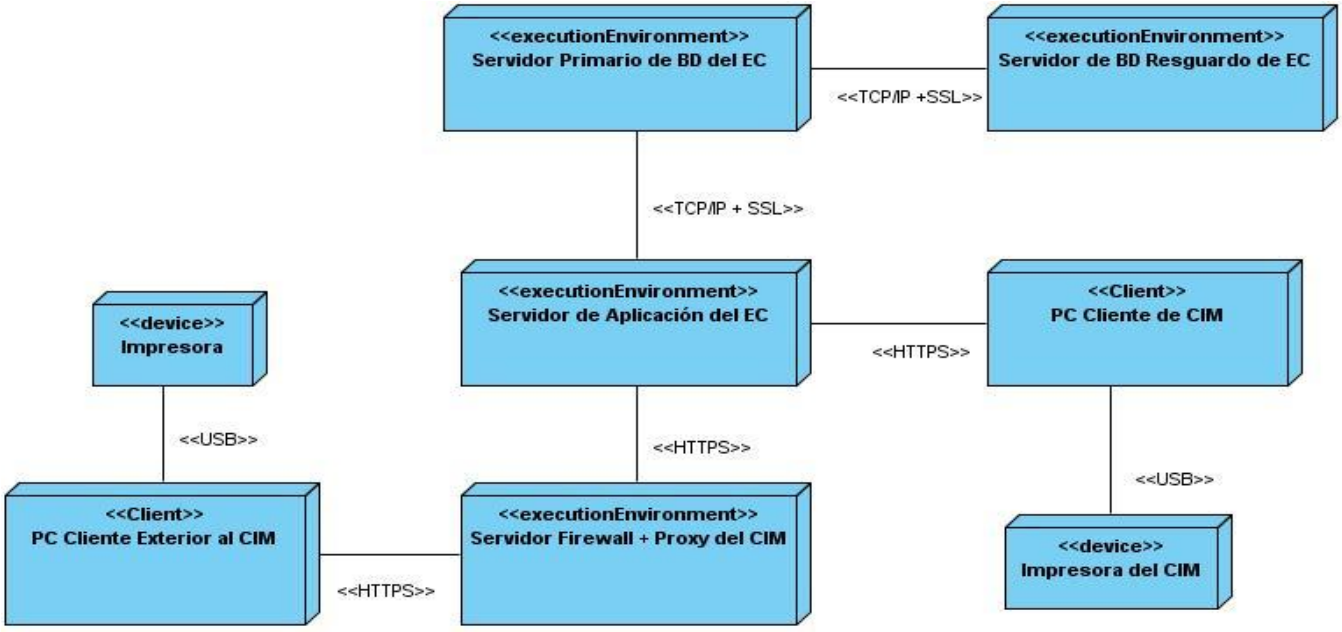


Fig. 6.Diagrama despliegue.

La aplicación estará distribuida de la siguiente forma:

En el Centro de Inmunología Molecular se encontrará la aplicación servidora, a la que se conectarán todas las PCs clientes del centro de forma directa, y también se conectarán a esa aplicación, PCs de otras partes del país con la debida autorización del centro y configuración en el servidor Firewall + Proxy. Existirán impresoras para la impresión de los modelos y otros datos de importancia; esto fue un requisito del cliente, pero en caso de faltar este dispositivo, no afectará en nada el funcionamiento de la aplicación. Habrá un servidor, que será la base de datos primaria, donde se encontrarán todos los datos del sistema, y se realizarán copias de resguardo de la información en otro servidor.

2.4.4 Vista de implementación

La vista de Implementación incluye la colección de componentes y subsistemas de implementación mediante el modelo de implementación (diagrama de componentes). Esta vista define además, ejecutables, bibliotecas, ficheros, subsistemas y dependencias entre ellos.

Se presenta la dependencia existente entre las capas en el siguiente diagrama y se describe básicamente la relación que existe internamente entre los componentes de cada vista.

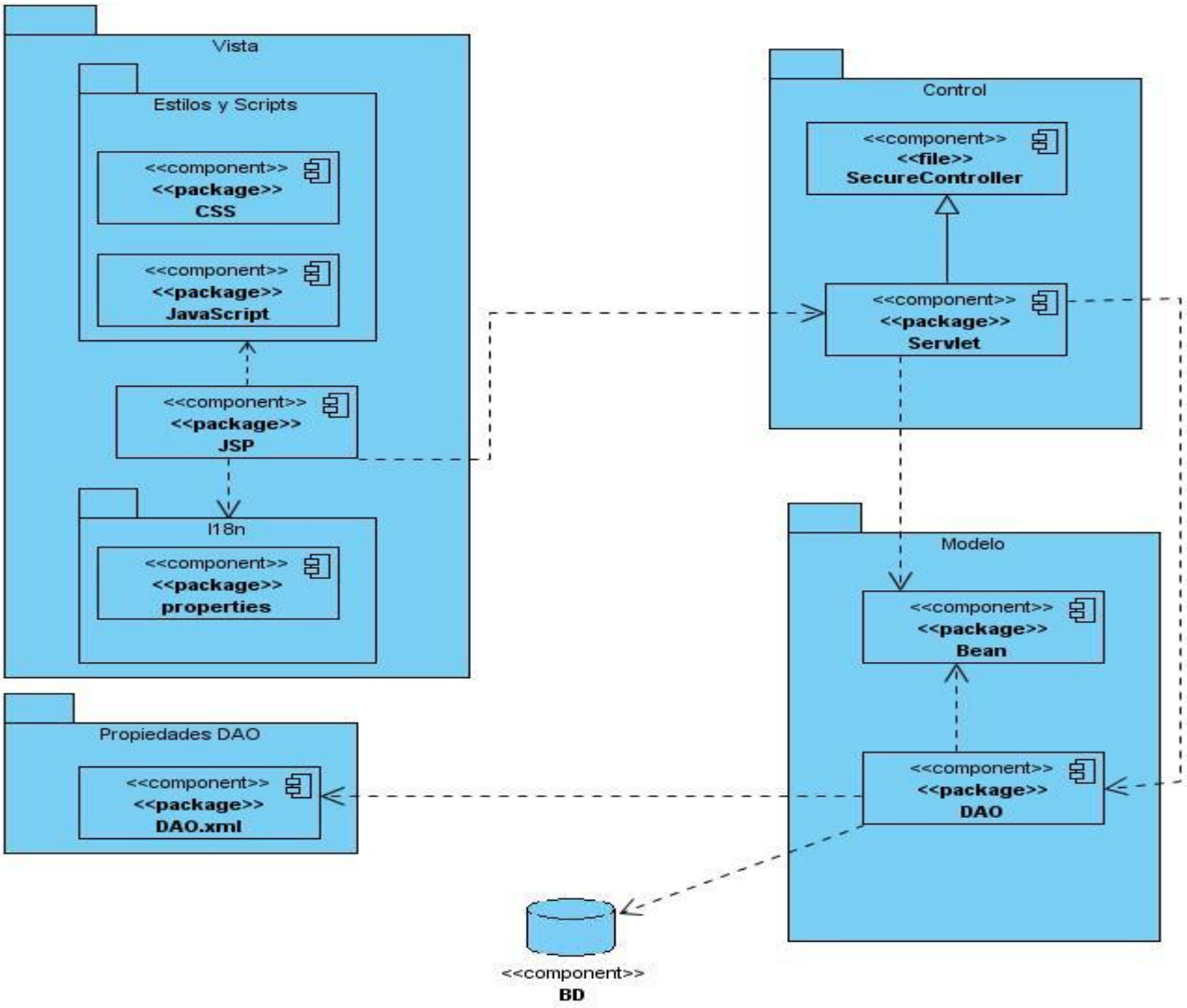


Fig.7. Vista de Implementación

En el paquete Vista se incluyen 2 paquetes y un componente:

- ✓ El paquete Estilos y Scripts contiene los componentes JavaScript y CSS:
  - o El componente Scripts contiene los ficheros *.js encargados de validar un conjunto de datos o conformar la vista.
  - o El componente CSS contiene los ficheros *.css contenedores de los estilos que se muestran en la vista.
- ✓ El paquete I18n contiene el componente properties encargado de la internacionalización del idioma.

- ✓ El componente JSP contiene las páginas *.jsp encargadas de crear la vista y además las páginas servidoras de las que éstas dependen.

En el paquete Control se incluye:

- ✓ El componente Servlet contiene todas las clases servlet encargadas del control de datos.
- ✓ El componente SecureController es un servlet del cual heredan todos los demás servlet en el sistema. En él se definen un conjunto de pasos que son comunes para atender cualquier petición de la vista, y se da la posibilidad que cada servlet que herede de él añada otros pasos que considere necesario a través de un método abstracto.

En el paquete Modelo se incluyen 2 componentes:

- ✓ El componente Bean contiene todas las clases Bean encargadas de la contención de datos.
- ✓ El componente DAO contiene todas las clases DAO encargadas de la comunicación con la Base de Datos.

En el paquete Propiedades DAO se incluye 1 componente:

- ✓ El componente DAO.xml contiene todos los ficheros *.xml contenedores de las consultas para obtener la información de la base de datos, estas consultas se pueden traducir mediante el uso de Digester (Biblioteca que se utiliza para procesar archivos XML a través de la búsqueda de reglas).

2.5 Lineamientos de diseño e implementación

En esta sección se incluyen todas las restricciones impuestas por el arquitecto de software para el correcto desarrollo y evolución de la solución, tanto desde el punto de vista de diseño, como de la implementación.

2.5.1 Patrones de diseño propuestos

Según lo expuesto en la Fundamentación Teórica y lo planteado en la vista lógica para el diseño de la aplicación se tuvieron en cuenta los siguientes patrones de diseño:

Patrón Objeto de Acceso a Datos, DAO (de sus siglas en inglés Data Access Object):

El DAO maneja la conexión con la fuente de datos para obtener y almacenar datos. Un ejemplo del uso de este patrón se evidencia en el Caso de Uso Mostrar Listado de Pacientes, donde StudySubjectDAO es el DAO encargado de obtener los datos de la base de datos y entregárselos a la clase StudySubjectBean

como entidad de negocio, evidenciándose como se separa el trabajo con el acceso a los datos y la lógica del negocio.

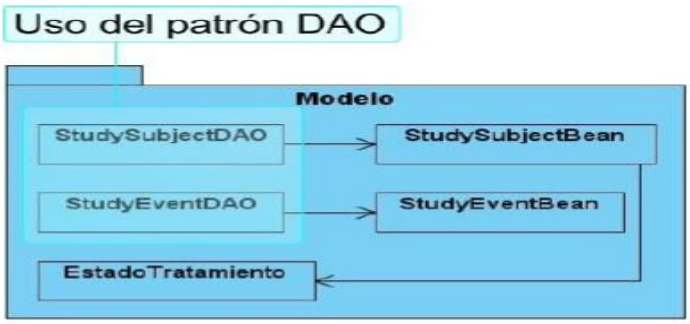


Fig. 8. Uso del Patrón DAO.

Método Plantilla (Template Method)

El sistema alasClínicas hace uso del mismo a través de la clase abstracta SecureController.java, todas las clases servlet que invocan cambios en el modelo o en la vista heredan de esta superclase y cada una se encarga de redefinir sus métodos.

Vista compuesta (Composite View)

Por ejemplo, la página JSP CronogramaList que incluye las otras páginas JSP que se muestran a continuación en color amarillo en la figura, usando la directiva include para construir la página final que muestra el contenido al usuario.

```
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>

<fmt:setBundle basename="org.akaza.openclinica.i18n.words" var="resword"/>
<fmt:setBundle basename="org.akaza.openclinica.i18n.format" var="resformat"/>
<fmt:setBundle basename="org.akaza.openclinica.i18n.workflow" var="resworkflow"/>
<fmt:setBundle basename="org.akaza.openclinica.i18n.notes" var="restext"/>

<jsp:include page="managestudy-header.jsp"/>
<jsp:include page="../include/breadcrumb.jsp"/>
<jsp:include page="../include/userbox.jsp"/>
<!-- move the alert message to the sidebar-->
<jsp:include page="../include/sideAlert.jsp"/>
```

Fig. 9. Patrón Vista Compuesta

Iterador (Iterator)

En este fragmento de código se usa el patrón iterador para iterar los objetos almacenados en el objeto alist de tipo ArrayList que almacena la consulta obtenida de la Base de Datos.

```
public ArrayList findByScheduled(String tipo, int studyId) {
    ArrayList answer = new ArrayList();
    this.setTypesExpected();

    HashMap variables = new HashMap();
    variables.put(new Integer(1), tipo);
    variables.put(new Integer(2), new Integer(studyId));
    String sql = digester.getQuery("findByScheduled");
    ArrayList alist = this.select(sql, variables);

    Iterator it = alist.iterator();
    while (it.hasNext()) {
        StudyEventDefinitionBean seb = (StudyEventDefinitionBean) this.getEntityFromHashMap((HashMap)
        answer.add(seb);
    }
    return answer;
}
```

Fig. 10. Patrón Iterador.

Fábrica Abstracta (Abstract Factory)

Cuando la fuente de almacenamiento está sujeta a cambios, se implementa el patrón Abstract Factory, permitiendo éste al DAO adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a los demás componentes del negocio.

En este caso, esta estrategia proporciona un objeto factoría abstracta de DAOs (SQLFactory) en el cual se implementa las diferentes fuentes de almacenamiento siendo éste utilizado por los DAOs.

```
public class SQLFactory {

    public final String DAO_USERACCOUNT = "useraccount";
    public final String DAO_STUDY = "study";
    public final String DAO_STUDYEVENTDEFNITION = "studyeventdefintion";
    public final String DAO_SUBJECT = "subject";
    public final String DAO_STUDYSUBJECT = "study_subject";
    public final String DAO_STUDYGROUP = "study_group";
    public final String DAO_STUDYGROUPCLASS = "study_group_class";
    public final String DAO_SUBJECTGROUPMAP = "subject_group_map";

     public void run(String dbName) {

        HashMap fileList = new HashMap();

        if ("oracle".equals(dbName)) {
            fileList.put(this.DAO_STUDYGROUP, "oracle_study_group_dao.xml");
            fileList.put(this.DAO_STUDYGROUPCLASS, "oracle_study_group_class_dao.xml");
            fileList.put(this.DAO_STUDYSUBJECT, "oracle_study_subject_dao.xml");
            fileList.put(this.DAO_SUBJECT, "oracle_subject_dao.xml");
            fileList.put(this.DAO_SUBJECTGROUPMAP, "oracle_subject_group_map_dao.xml");

        } else if ("postgres".equals(dbName)) {
            fileList.put(this.DAO_USERACCOUNT, "useraccount_dao.xml");
            fileList.put(this.DAO_STUDY, "study_dao.xml");
            fileList.put(this.DAO_STUDYEVENTDEFNITION, "studyeventdefinition_dao.xml");
            fileList.put(this.DAO_STUDYGROUP, "study_group_dao.xml");
            fileList.put(this.DAO_STUDYGROUPCLASS, "study_group_class_dao.xml");
            fileList.put(this.DAO_STUDYSUBJECT, "study_subject_dao.xml");
```

Fig. 11. Patrón Fábrica Abstracta.

Bajo acoplamiento

Ejemplo de uso de este patrón es mediante la clase ecCronogramaRow y la clase ecCronogramaBean, un cambio en esta clase implicaría el mínimo cambio en la forma en que la clase ecCronogramaRow genera a través del método generateRowsFromBeans () el arreglo de ecCronogramaBeans.

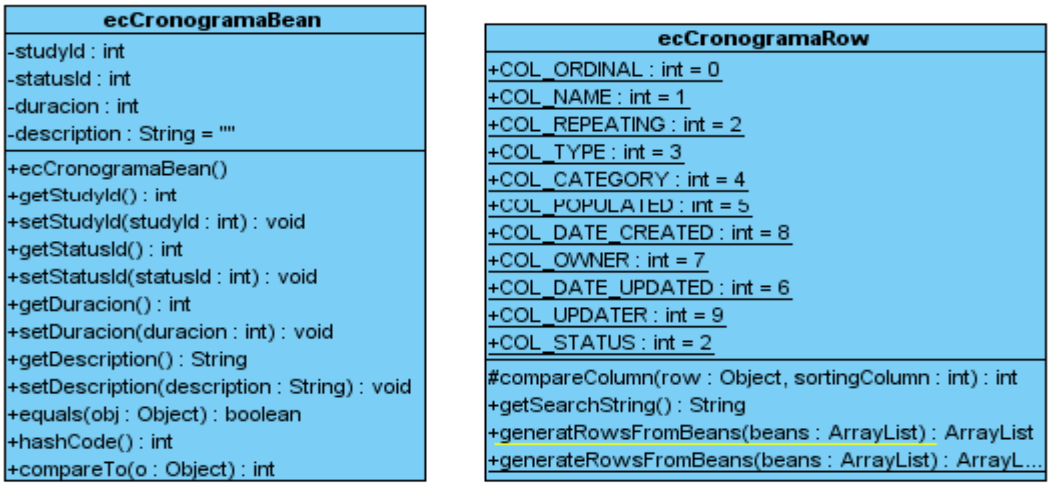


Fig. 12. Patrón Bajo Acoplamiento

Experto

Un ejemplo del uso de este patrón se evidencia en la clase EventCRFBean en el método getStage. En este método se utiliza la información de la base de datos y a través de la clase DataEntryStage se convierte el valor del estado de cada hoja de CRD, que es un entero, a un objeto de tipo DataEntryStage, y en este caso el responsable de hacer esta tarea es el experto en información EventCRFBean.

Alta cohesión

Esto se evidencia desde el punto de vista que cada una de las clases implementadas tiene responsabilidades específicas y no están sobrecargadas con demasiadas de ellas, ejemplos: EntityBeanTable, StudyEventDefinitionRow, StudyEventDefinitionBean.

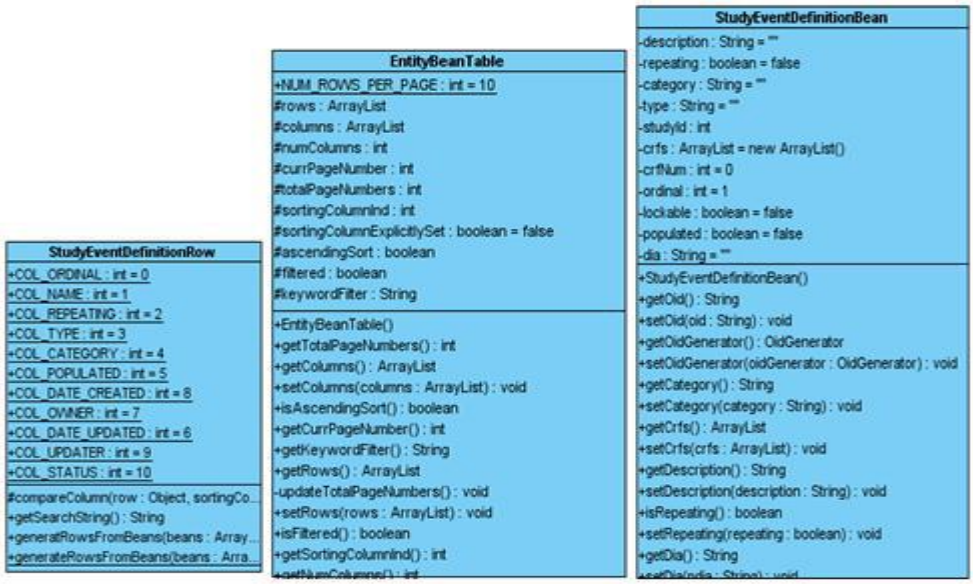


Fig. 13. Patrón Alta Cohesión

Creador

El uso de este patrón se puede encontrar en todos los diagramas de clases del diseño y diagramas de componentes realizados. Un ejemplo del mismo se aprecia entre las clases DAO y Bean, donde las clases DAO son las encargadas de la creación de objetos Bean.

Fig.14. Patrón Creador

Controlador

Las clases Servlet son clases controladoras que se encargan de gestionar todos los eventos del sistema. Ante cualquier petición gestionan la información relacionada a la misma a través de las clases del paquete modelo, procesan esta petición y redireccionan a una página JSP la respuesta a esta solicitud. El uso del

patrón Controlador se evidencia en todas las clases Servlet que conforman los diagramas de clases del diseño y diagramas de componentes.

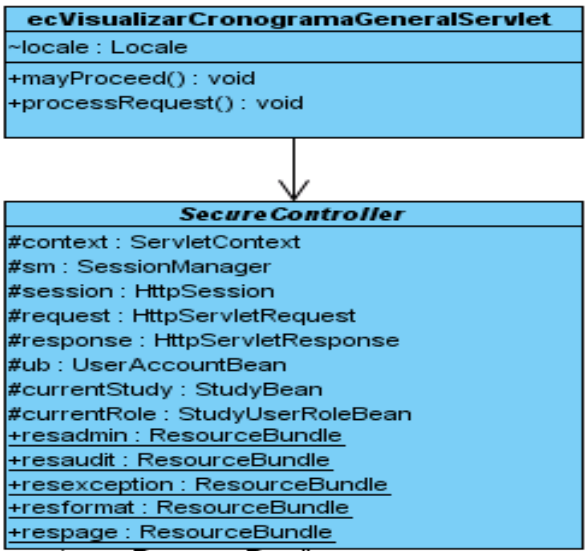


Fig.15. Patrón Controlador

2.5.2 Estándares de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código y contribuye a la documentación del código y al ahorro de tiempo y recursos en la comprensión de lo escrito. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación para asegurarse de que todos los desarrolladores del proyecto trabajen de forma coordinada de manera tal que sea comprensible para todos y que el código en consecuencia sea mantenible. Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento pues el mismo debe desarrollarse de manera que pueda ser reutilizado total o parcialmente en cualquier otro software, ya que cualquier otra persona debería ser capaz de leer el código , entender su funcionamiento o podría modificar el sistema de software para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento.

Las reglas que define el estilo de codificación adoptado en el presente trabajo se exponen íntegramente en el Anexo 1.

2.6 Conclusiones parciales del capítulo

En el capítulo en cuestión se realizó la descripción de la arquitectura a través de las vistas arquitectónicas definidas por RUP, así como el uso de los patrones estudiados y analizados en el capítulo anterior. Además se definieron los estándares de codificación a seguir para un mejor uso y entendimiento de los desarrolladores.

Capítulo 3: Evaluación de la arquitectura propuesta.

En el presente capítulo se pretende evaluar la arquitectura propuesta analizando los resultados obtenidos de acuerdo con las técnicas y métodos de evaluación de arquitecturas de software.

Esta evaluación se inicia desde el momento en que quedan bien definidos los requisitos que debe cumplir el sistema para satisfacer las necesidades del usuario. La arquitectura debe estar preparada para desarrollar los casos de usos críticos y a medida que avance el proyecto permita incluir los restantes. Con ello se busca que la propuesta sea funcional y cumpla con los atributos de calidad definidos en los diferentes modelos de evaluación; por lo que debe ser sometida a prueba, con el fin de que el sistema que la soporte sea robusto, seguro, de gran rendimiento y usabilidad.

Una arquitectura es adecuada cuando cumple dos criterios:

- El sistema resultante cumple con los atributos de calidad definidos.
- El sistema puede ser construido con los recursos disponibles para el mismo.

3.1 Aplicación del método de evaluación

En el capítulo 1 quedó expuesto el método de evaluación propuesto para la arquitectura siendo el mismo el ATAM, además de los atributos de calidad ISO/IEC 9126 adaptado para arquitecturas de software apoyado de las técnicas estudiadas con el instrumento de evaluación Utlily Tree.

A continuación se presenta la evaluación de la arquitectura propuesta según las fases del método de evaluación ATAM.

Fase 1: Presentación

Paso 1. Presentación del ATAM: El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde las preguntas propuestas.

Paso 2. Presentación de las metas del negocio: Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico. En este paso, el líder del proyecto presenta el sistema desde la perspectiva del negocio, donde quedan expuestos como principales funciones del sistema, los CU más significativos mostrados en el capítulo anterior.

Como restricciones del sistema, relevantes arquitectónicamente, quedaron identificadas:

1. Se deben recopilar los datos de varios EC que se encuentran distribuidos geográficamente por todo el país.
2. Los datos que suministran la mayoría de los EC están en copia dura, o sea, documentos impresos.
3. Los datos con los cuales trabaja el CIM son de vital importancia y de carácter estratégico para la gestión de EC del país por lo cual es necesario garantizar su seguridad.
4. Debe existir un único repositorio central para almacenar todos los datos recopilados.
5. La cantidad de información que se maneja es muy grande.

Y como metas de los atributos de calidad que dan forma a la arquitectura:

1. Lograr la habilidad del sistema de mantenerse operativo a lo largo del tiempo.
2. Lograr que el grado con el que el sistema cumpla sus funciones designadas dentro de ciertas restricciones como velocidad, exactitud o uso de memoria, sea satisfactorio.
3. Lograr la capacidad de someter al sistema a reparaciones y evoluciones, y que éstas se desarrollen de manera rápida y a bajo costo.
4. Lograr la habilidad de realizar cambios futuros al sistema sin mayores afectaciones a lo que ya está hecho.
5. Lograr la habilidad del sistema para ser ejecutado en diferentes ambientes computacionales.

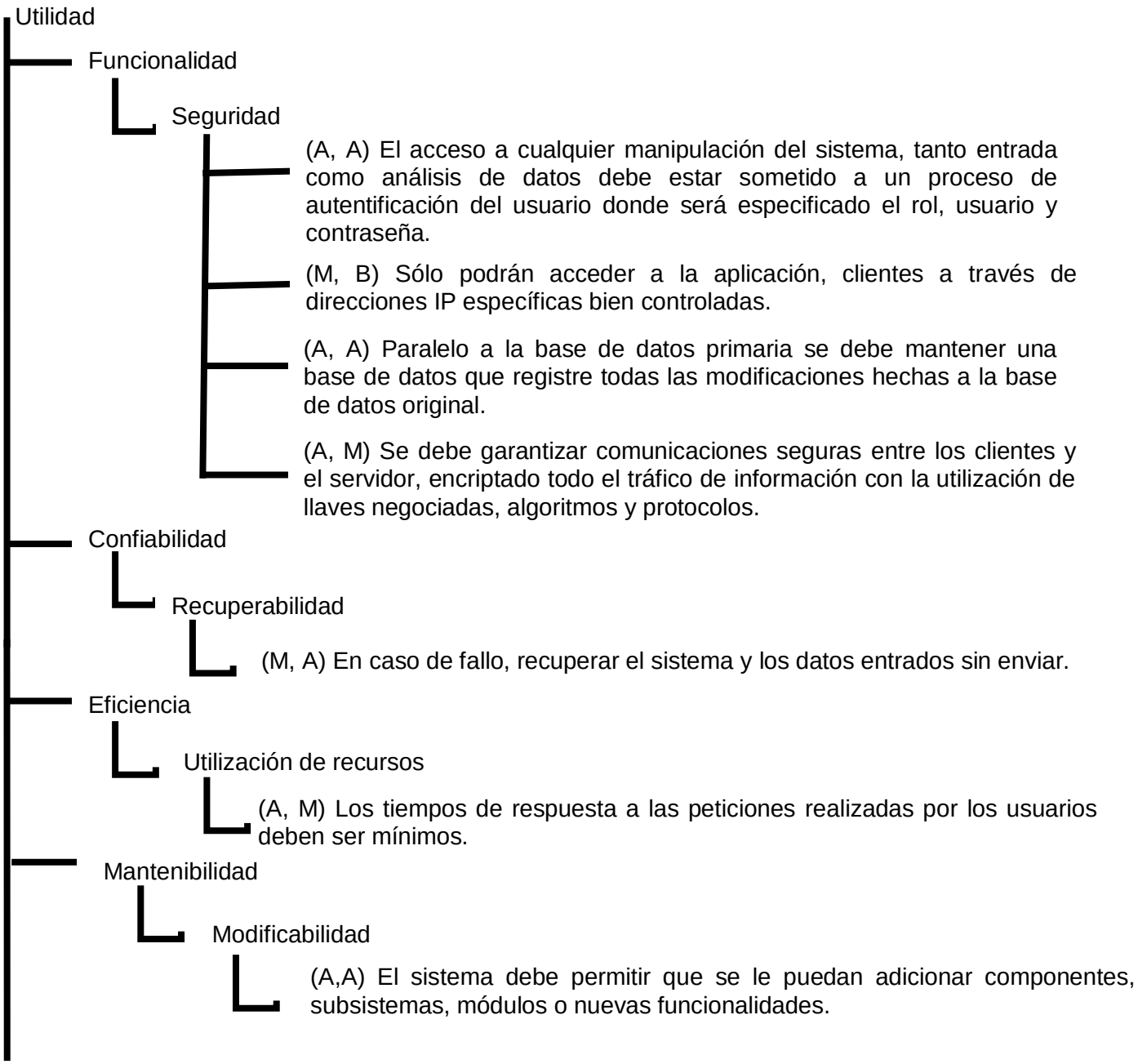
Paso 3. Presentación de la arquitectura: El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio. Este paso es desarrollado por los arquitectos e incluye toda la descripción arquitectónica en detalle, no se presenta en esta sección toda la información concerniente a la arquitectura que está siendo evaluada, ya que en el capítulo anterior se mostró completamente.

Fase 2: Investigación y análisis

Paso 4. Identificación de los enfoques arquitectónicos: Estos elementos son detectados, pero no analizados. En este momento quedan identificadas todas las propuestas arquitectónicas que serán analizadas más tarde, o sea, se especifica la tecnología que va a ser utilizada, el patrón arquitectónico y los patrones de diseño, estos elementos no se muestran a continuación ya que se encuentran detallados en la Fundamentación Teórica.

Paso 5. Generación del Árbol de Utilidad: Se obtienen los atributos de calidad que engloban la “utilidad” del sistema especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.

Al construir el árbol de utilidad, se define la prioridad del escenario como un par (X, Y) en el cual X define el esfuerzo de satisfacer el escenario, y la Y indica los riesgos que se corren al excluirlos del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio). El árbol de utilidad generado se toma como un plan para el resto de la evaluación que realiza el método. Indica además al equipo evaluador dónde ocupar su tiempo y dónde probar aproximaciones y riesgos arquitectónicos.



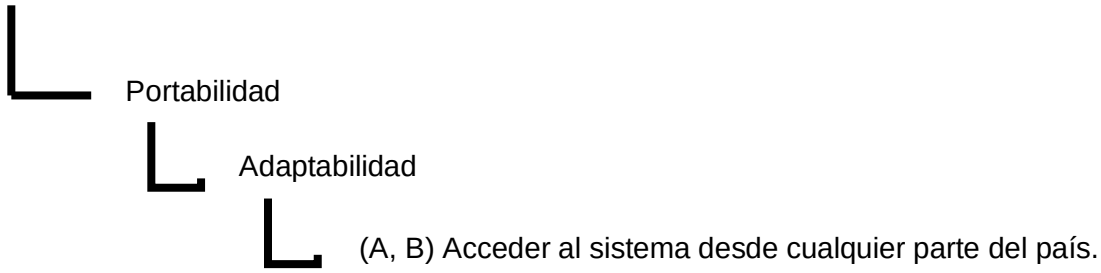


Fig. 26. Árbol de Utilidad

En este paso se obtienen los escenarios identificados en el paso 5 y las propuestas arquitectónicas que cumplen con estos escenarios, donde se documenta detalladamente la medida en la cual son adecuados el uno para el otro, y logra la meta del equipo de evaluación de estar convencidos que la propuesta instanciada en la arquitectura que se está evaluando es la apropiada para satisfacer los requerimientos de un atributo específico.

Tabla 6 Escenario#1.

Escenario #: 1	Escenario: El acceso a cualquier manipulación del sistema, tanto entrada como análisis de datos debe estar sometido a un proceso de autentificación del usuario donde será especificado el rol, usuario y contraseña.			
Atributo(s)	Funcionalidad-Seguridad			
Ambiente	Operación normal.			
Estímulo	Usuario que no cumple con los requisitos de autenticación intenta acceder a las funcionalidades del sistema.			
Respuesta	Sólo se muestran las funcionalidades a las que tiene acceso según su tipo de usuario.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
Clase SecureController	S1			N1
Explicación	Las funcionalidades del sistema deben mostrarse de acuerdo al usuario que este activo, garantizando de este modo que la aplicación sea utilizada en correspondencia con los privilegios permitidos a cada tipo de usuario según su rol, usuario y contraseña, lo cual se resuelve con la clase SecureController.			

Tabla 7 Escenario#2.

Escenario #: 2	Escenario: Sólo podrán acceder a la aplicación, clientes a través de direcciones IP específicas bien controladas.
Atributo(s)	Funcionalidad-Seguridad
Ambiente	Operación normal.
Estímulo	El usuario accede desde cualquier dirección IP al sistema.

Respuesta	Acceso denegado si no es la dirección IP correcta.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
Clase SecureController	S2			N2
Explicación	En la clase SecureController se implementa el método Seguridad que es el encargado de verificar que el usuario esté accediendo desde una dirección IP correcta.			

Tabla 8 Escenario#3.

Escenario #: 3	Escenario: Paralelo a la base de datos primaria se debe mantener una base de datos que registre todas las modificaciones hechas a la base de datos original.			
Atributo(s)	Funcionalidad-Seguridad			
Ambiente	Operación normal.			
Estímulo	Necesidad de salvar los datos en caso de algún fallo en la base de datos primaria.			
Respuesta	En caso de ocurrir un fallo se recuperan los datos almacenados en otra base de datos.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
En el despliegue se implementará un sistema de replicas entre las bases de datos.				N3
Explicación	La protección física de los datos es uno de los puntos más sensibles en la seguridad del sistema. Mediante la decisión arquitectónica de un sistema de replicas de bases de datos la distribución física de los datos estará garantizada en otra base de datos.			

Tabla 9 Escenario#4.

Escenario #: 4	Escenario: Se debe garantizar comunicaciones seguras entre los clientes y el servidor, encriptado todo el tráfico de información con la utilización de llaves negociadas, algoritmos y protocolos.			
Atributo(s)	Funcionalidad-Seguridad			
Ambiente	Operación normal.			
Estímulo	Envío de la información al CIM.			
Respuesta	Encripta la información y la envía al CIM de forma segura.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
Protocolo seguro HTTPS	S3			N4

Explicación	La seguridad de la información que es enviada a través de la red se garantiza con el uso de Protocolo HTTPS.			

Tabla 10 Escenario#5.

Escenario #: 5	Escenario: En caso de fallo, recuperar el sistema y los datos entrados sin enviar.			
Atributo(s)	Confiabilidad-Recuperabilidad			
Ambiente	Operación normal.			
Estímulo	Error del sistema operativo que reinicia el ordenador de la PC del cliente.			
Respuesta	No hay recuperación de los datos.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
No hay			R1	
Explicación	Al no recuperarse los datos del sistema luego de un fallo externo por falta de fluido eléctrico, error del sistema operativo o cualquier error ajeno a la aplicación en la máquina del cliente se produce el mayor de los errores ya que no existe ninguna decisión arquitectónica.			

Tabla 11 Escenario#6.

Escenario #: 6	Escenario: Los tiempos de respuesta a las peticiones realizadas por los usuarios deben ser mínimos.			
Atributo(s)	Eficiencia-Utilización de recursos			
Ambiente	Múltiples usuarios conectados concurrentemente.			
Estímulo	El usuario necesita realizar una determinada acción en el sistema.			
Respuesta	Muestra los datos rápidamente.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
SGBD PostgreSQL	S4			N5
Implementar procedimientos almacenados				N6
Explicación	Bajo una condición de stress como la conexión concurrente de múltiples usuarios a la aplicación haciendo diferentes o las mismas peticiones, la decisión arquitectónica que garantizan la respuesta satisfactoria del sistema son: SGBD PostgreSQL para manejar las conexiones concurrentes de muchos usuarios de forma eficiente. Implementar procedimientos almacenados permite que las respuestas de las consultas a la BD sean más rápidas.			

Tabla 12 Escenario#7.

Escenario #: 7	Escenario: El sistema debe permitir que se le puedan adicionar componentes, subsistemas, módulos o nuevas funcionalidades.			
Atributo(s)	Mantenibilidad-Acoplamiento			
Ambiente	Operación normal.			
Estímulo	Necesidad de adicionar componentes, subsistemas o nuevas funcionalidades al sistema.			
Respuesta	La arquitectura definida soporta la integración de nuevas funcionalidades al sistema.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
Patrón arquitectónico MVC	S5			N7
Patrón de diseño DAO	S6			N8
Hojas de Estilo CSS				N9
Explicación	Estas decisiones arquitectónicas garantizan el alto grado de mantenibilidad y modificabilidad en la arquitectura propuesta ya que: Por el patrón DAO la migración a un gestor de base de datos diferente sería más fácil pues oculta los detalles de implementación de la fuente de datos a sus clientes. Mediante el patrón MVC se alcanza una mayor cohesión entre los elementos que se especializan en sus funciones, y un bajo acoplamiento entre ellos, por tanto los cambios son fáciles de implementar. Al utilizar los estilos CSS se aumenta la capacidad del sistema de evolucionar fácilmente y responder a los cambios de forma rápida, ya que esta decisión tiene la ventaja de ofrecer a los desarrolladores la facilidad de cambiar el formato de las páginas con sólo cambiar el elemento que desea en el estilo.			

Tabla 13 Escenario#8.

Escenario #: 8	Escenario: Acceder al sistema desde cualquier parte del país.			
Atributo(s)	Portabilidad-Adaptabilidad			
Ambiente	Operación normal.			
Estímulo	El usuario desea utilizar el sistema desde cualquier provincia o municipio.			
Respuesta	La aplicación se encuentra disponible a través de la web.			
Decisiones arquitectónicas	Punto de sensibilidad	Punto de trade-off	Riesgo	No Riesgo
Tecnología Web	S7			N10
Explicación	La decisión arquitectónica de utilizar tecnología Web se basa en la necesidad de garantizar el acceso al sistema desde cualquier geografía del			

	país, sólo necesita estar conectado a Internet y tener instalado algún navegador Web, como ejemplo de la tecnología Web utilizada está J2EE.

Paso 6. Análisis de los enfoques arquitectónicos

Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. Se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance. Se utilizan las preguntas similares a las presentadas en el paso 5.

Fase 3: Pruebas

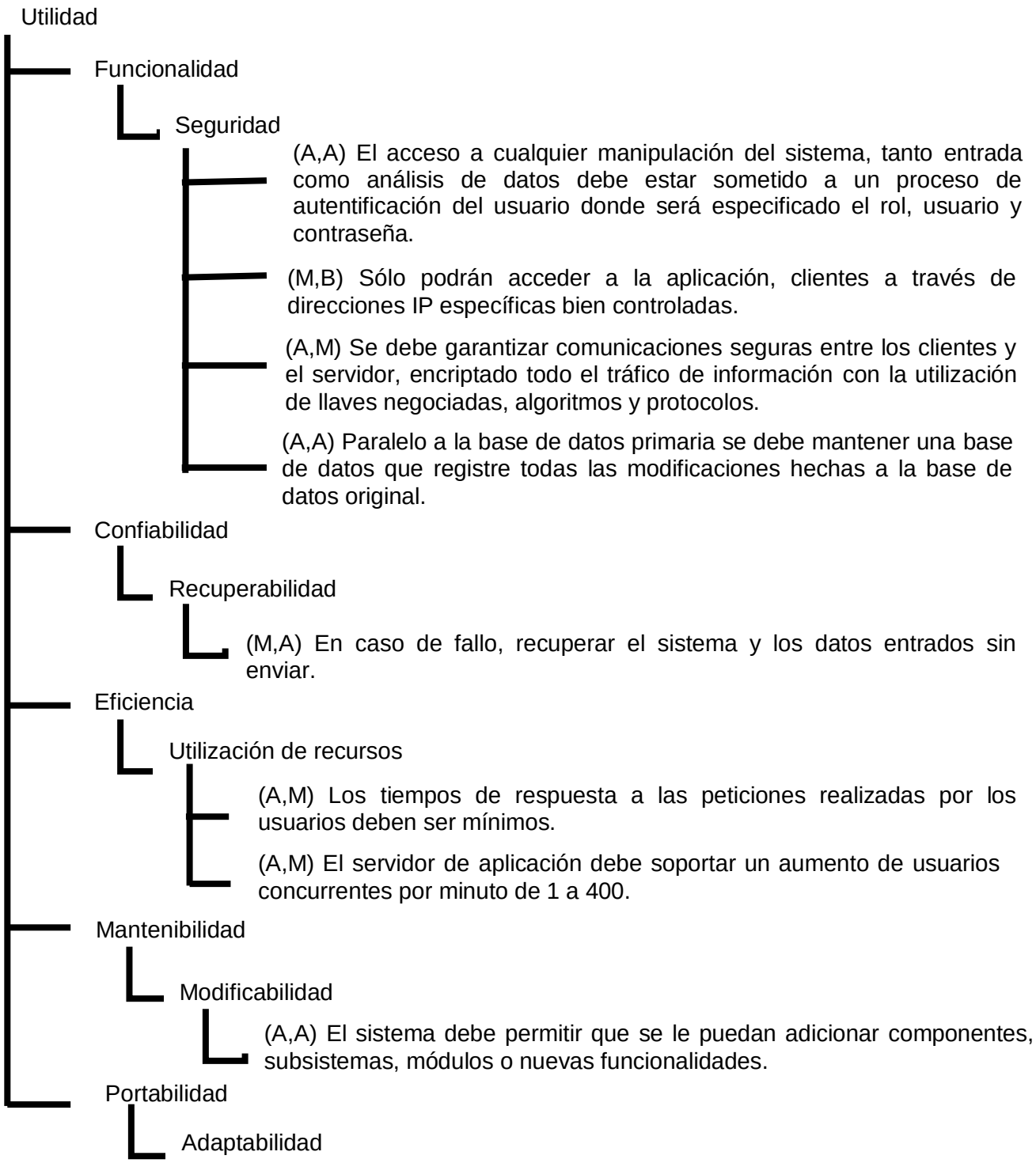
Paso 7. Lluvia de ideas y establecimiento de prioridad de escenarios: Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.

Mientras que la generación del árbol de utilidad es fundamental para entender cómo el arquitecto percibe y maneja las guías arquitectónicas de los atributos de calidad, el propósito de la lluvia de ideas de escenarios es tomarle el pulso a una gran comunidad de stakeholders.

Tabla 14 Establecimiento de prioridad de escenarios

Nro.	Escenario	Votos	Atributo de Calidad
9	Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.	3	Portabilidad
10	El servidor de aplicación debe soportar un aumento de usuarios concurrentes por minuto de 1 a 400.	5	Eficiencia

La lista de los escenarios priorizados resultantes de la lluvia de ideas, fue comparada con los generados por el árbol de utilidad, y al descubrir nuevos escenarios fueron adicionados al árbol obtenido en el paso 5 el cual recoge todos los escenarios identificados durante el proceso de evaluación.


Utilidad
Funcionalidad
Seguridad
(A,A) El acceso a cualquier manipulación del sistema, tanto entrada como análisis de datos debe estar sometido a un proceso de autentificación del usuario donde será especificado el rol, usuario y contraseña.
(M,B) Sólo podrán acceder a la aplicación, clientes a través de direcciones IP específicas bien controladas.
(A,M) Se debe garantizar comunicaciones seguras entre los clientes y el servidor, encriptado todo el tráfico de información con la utilización de llaves negociadas, algoritmos y protocolos.
(A,A) Paralelo a la base de datos primaria se debe mantener una base de datos que registre todas las modificaciones hechas a la base de datos original.
Confiabilidad
Recuperabilidad
(M,A) En caso de fallo, recuperar el sistema y los datos entrados sin enviar.
Eficiencia
Utilización de recursos
(A,M) Los tiempos de respuesta a las peticiones realizadas por los usuarios deben ser mínimos.
(A,M) El servidor de aplicación debe soportar un aumento de usuarios concurrentes por minuto de 1 a 400.
Mantenibilidad
Modificabilidad
(A,A) El sistema debe permitir que se le puedan adicionar componentes, subsistemas, módulos o nuevas funcionalidades.
Portabilidad
Adaptabilidad

(A,B) Acceder al sistema desde cualquier parte del país.

(A,B) Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

Fig. 27. Árbol de Utilidad

Paso 8. Análisis de los enfoques arquitectónicos: Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.

Fase 4: Reporte

Paso 9. Presentación de los resultados: Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Finalmente, haciendo uso del método de evaluación seleccionado ATAM se resume y presenta; las más importantes salidas son:

- El conjunto de escenarios priorizados
- El árbol de utilidad
- Los riesgos descubiertos: Anexo 2
- Los no riesgos descubiertos: Anexo 3
- Los puntos de sensibilidad y de trade-off encontrados: Anexo 4 y Anexo 5 respectivamente

3.2 Conclusiones parciales del capítulo

En este capítulo se realizó la evaluación de la arquitectura del sistema alasClínicas. Se analizaron brevemente los atributos de calidad de la arquitectura propuesta, teniendo en cuenta el procedimiento descrito en el capítulo 1 (método ATAM), arrojando como resultado una serie de riesgos y puntos sensibles, así como los escenarios identificados. Estos resultados serán tomados por el arquitecto del proyecto facilitándole la toma de decisiones.

Conclusiones

- Se definió la línea base de la arquitectura, para la misma se realizó un estudio de estilos y patrones que proporcionaran al sistema calidad, claridad y sencillez en la estructura y diseño de la solución.
- A través del diseño de las 4 + 1 vistas propuestas por RUP se logró desarrollar la propuesta de solución de la arquitectura del sistema.
- Se evaluó la arquitectura propuesta aplicando el método de evaluación ATAM, lo cual contribuyó al diseño de una arquitectura flexible, robusta y escalable.

Recomendaciones

Después de culminar el presente trabajo se plantean las siguientes recomendaciones:

- Evaluar la posibilidad de utilizar los frameworks Hibernate y Spring para refinar la descripción de la arquitectura propuesta.
- Aplicar otros métodos de evaluación de la arquitectura para aumentar la fortaleza del sistema.

Referencias Bibliográficas

1.http://www.s4research.es/que-es-openclinica-solutions-cro-barcelona-espana-open-source-ensayos-clinicos. [En línea] [Citado el: 25 de 10 de 2009.]

2. http://www.openclinica.org. [En línea] [Citado el: 25 de 10 de 2009.]

3. Clements, Paul. *A Survey of Architecture Description Languages. . Alemania : s.n.,* 1996.

4. Shaw,Mary y Garlan ,David. An introduction to software architecture. s.l. : CMU Software Engineering Institute Technical Report, 1994.

5. IEEE Std, 1471-2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000.

6. Montero Morales, Yagnieris y Carmenate Valero, Elier. *Selección de un Lenguaje de Descripción Arquitectónica y modelado de las decisiones arquitectónicas del proyecto ERP Cuba.* Ciudad Habana : Facultad 5,Universidad de las Ciencias Informáticas, 2009.

7. Wolf, Dewayne E. Perry and Alexander L. *Foundations for the Study of Software Architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, Octubre 1992.

8. Clements,Paul y Shaw Mary . *A field guide to Boxology : Preliminary classification of architectural styles for software systems.* 1997.

9. Shaw, Mary y Garlan, David. *An introduction to software architecture.* 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

10. http://isg3.pbworks.com. [En línea] [Citado el: 22 de 10 de 2009.]

11. Sarver, Toby. *Pattern Refactoring Workshop.* 2000.

12. Shaw, Mary. *Some Patterns for Software Architecture.* s.l. : Addison-Wesley, 1996.

13. Burbeck, Steve. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).* [En línea] [Citado el: 15 de 11 de 2009.]
http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

14. Gamma, Erich , Helm, Richard, Johnson , Ralph y Vlissides, John. *Design Patterns-Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994.

15. Jacobson Ivar ,Booch Grady y Rumbaugh James. *The Unified Software Development Process.* s.l. : Addison-Wesley, 1999.

16. Kruchten, Philippe B. *Architectural blueprints: The 4+1 view model of architecture.* s.l. : IEEE Software, 1995.

17. B. Collins-Sussman, B. W. Fitzpatrick y C. Michael Pilato. Control de versiones con Subversion. *Control de versiones con Subversion.* [En línea] 2004. [Citado el: 22 de 10 de 2009.] http://svnbook.red-bean.com/nightly/es/svn-book.pdf.

18. Zapata, Luis Giraldo y Liliana. *Herramientas de Desarrollo de Ingeniería de Software para Linux.* 2005.

19. González Hernández, Marlien y De Laosa Socárras, Jenely. *Sistema de Manejo de Datos de Ensayos Clínicos Cubano. Módulo Validación: Diseño del submódulo "Derivación de las variables del Cuaderno de Recogida de Datos". Tesis (Ingeniero en Ciencias Informáticas)* Ciudad Habana. Universidad de las Ciencias Informáticas, 2008.

20. Larman, C. *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* La Habana : Félix Valera, 2004.

21. Paul Clements, Rick Kazman y Mark Klein. *ATAM: Method for Architecture Evaluation.*
s.l. : Carnegie Mellon, Software Engineering Institute, 2000. CMU/SEI-2000-TR-004.

22. Clements, Rick Kazman Mark Klein y Paul. *ATAM: Method for Architecture Evaluation.* s.l. : Pittsburgh, 2007.

23. Gómez, Omar Salvador. *Evaluando Arquitecturas de Software. Parte 2.* México : Brainworx S.A, 2007.

24. Francisca Losavio, Ledis Chirinos, Nicole Lévy, Amar Ramdane-Cherif. Quality Characteristics for Software Architecture. *Quality Characteristics for Software Architecture.* [En línea] 2003. [Citado el: 18 de 02 de 2010.] http://www.jot.fm/issues/issue_2003_03/article2.

25. Rick Kazman, Paul Clements y Mark Klein. *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

26. E. Camacho, F. Cordeso, y G. Nuñez. *Arquitectura de Software.* 2004.

Bibliografía

1. B. Collins-Sussman, B. W. Fitzpatrick y C. Michael Pilato. Control de versiones con Subversion. *Control de versiones con Subversion.* [En línea] 2004. [Citado el: 22 de 10 de 2009.] http://svnbook.red-bean.com/nightly/es/svn-book.pdf.

2. Burbeck, Steve. Application programming in Smalltalk-80: How to use Model-View-Controller (MVC). *Application programming in Smalltalk-80: How to use Model-View-Controller (MVC).* [En línea] [Citado el: 15 de 11 de 2009.]
http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html.

3. Clements, Paul. *A Survey of Architecture Description Languages. . Alemania : s.n.,* 1996.

4. Clements,Paul y Shaw Mary . *A field guide to Boxology : Preliminary classification of architectural styles for software systems.* 1997.

5. Clements, Rick Kazman Mark Klein y Paul. *ATAM: Method for Architecture Evaluation.* s.l. : Pittsburgh, 2007.

6. E. Camacho, F. Cordeso, y G. Nuñez. *Arquitectura de Software.* 2004.

7. Francisca Losavio, Ledis Chirinos, Nicole Lévy, Amar Ramdane-Cherif. Quality Characteristics for Software Architecture. *Quality Characteristics for Software Architecture.* [En línea] 2003. [Citado el: 18 de 02 de 2010.] http://www.jot.fm/issues/issue_2003_03/article2.

8. Gamma, Erich , Helm, Richard, Johnson , Ralph y Vlissides, John. *Design Patterns-Elements of Reusable Object-Oriented Software.* s.l. : Addison Wesley, 1994.

9. Garlan, David y Allen, Robert. *The Wright Architectural Description Language.* s.l. : Carnegie Mellon University, 1996.

10. Gómez, Omar Salvador. *Evaluando Arquitecturas de Software. Parte 1.* México : Brainworx S.A, 2007.

11. Gómez, Omar Salvador. *Evaluando Arquitecturas de Software. Parte 2.* México : Brainworx S.A, 2007.

12. González Hernández, Marlien y De Laosa Socárras, Jenely. *Sistema de Manejo de Datos de Ensayos Clínicos Cubano. Módulo Validación: Diseño del submódulo "Derivación de las variables del*

Cuaderno de Recogida de Datos". Tesis (Ingeniero en Ciencias Informáticas) Ciudad Habana. Universidad de las Ciencias Informáticas, 2008.

13. http://isg3.pbworks.com. [En línea] [Citado el: 22 de 10 de 2009.]

14. http://www.s4research.es/que-es-openclinica-solutions-cro-barcelona-espana-open-source-ensayos-clinicos. [En línea] [Citado el: 25 de 10 de 2009.]

15. http://www.openclinica.org. [En línea] [Citado el: 25 de 10 de 2009.]

16. IEEE Std, 1471-2000. *Recommended Practice for Architectural Description of Software-Intensive Systems.* 2000.

17. Jacobson Ivar ,Booch Grady y Rumbaugh James. *The Unified Software Development Process.* s.l. : Addison-Wesley, 1999.

18. Kruchten, Philippe B. *Architectural blueprints: The 4+1 view model of architecture.* s.l. : IEEE Software, 1995.

19. Larman, C. *UML y Patrones: Introducción al análisis y diseño orientado a objetos.* La Habana : Félix Valera, 2004.

20. Len, Bass, Clement, Paul ,and Kazman, Rich. *Software Architecture in Practice.* s.l. : Addison-Wesley, 2003.

21. Montero Morales, Yagnieris y Carmenate Valero, Elier. *Selección de un Lenguaje de Descripción Arquitectónica y modelado de las decisiones arquitectónicas del proyecto ERP Cuba.* Ciudad Habana : Facultad 5,Universidad de las Ciencias Informáticas, 2009.

22. Paul Clements, Rick Kazman y Mark Klein. *ATAM: Method for Architecture Evaluation.* s.l. : Carnegie Mellon, Software Engineering Institute, 2000. CMU/SEI-2000-TR-004.

23. Reynoso, Carlos and Kicillof, Nicolás. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft*. [En línea] 2004. [Citado el: 02 de 12 de 2009.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

24. Rick Kazman, Paul Clements y Mark Klein. *Evaluating Software Architectures. Methods and case studies.* s.l. : Addison Wesley, 2001.

25. Sarver, Toby. *Pattern Refactoring Workshop.* 2000.

26. Shaw, Mary. *Some Patterns for Software Architecture.* s.l. : Addison-Wesley, 1996.

27. Shaw,Mary y Garlan, David. An introduction to software architecture. s.l. : CMU Software Engineering Institute Technical Report, 1994.

28. Shaw, Mary y Garlan, David. *An introduction to software architecture.* 1994. CMU/SEI-94-TR-21, ESC-TR-94-21.

29. Wolf, Dewayne E. Perry and Alexander L. *Foundations for the Study of Software Architecture.* s.l. : ACM SIGSOFT Software Engineering Notes, Octubre 1992.

30. Wolf, Dewayne E. Perry and Alexander L. *Foundations for the Study of Software .* 1992.

31. Zapata, Luis Giraldo y Liliana. *Herramientas de Desarrollo de Ingeniería de Software para Linux.* 2005.

Anexos

Anexo 1: Estándares de Codificación.

Generales.

1: Etiquetas de apertura JSP:

<% //código %> -- Etiquetas para especificar código java en una página JSP.

<%@ //código %> -- Etiquetas para obtener información de la página que se utilice.

2: Etiquetas del motor JSP:

<jsp: include //código /> --Permite la inclusión de un fichero.

<jsp: useBean //código /> --Permite la búsqueda o instanciación de un componente Java(JavaBean).

<jsp: setProperty //código /> --Para el establecimiento del valor de una propiedad de un JavaBean.

<jsp: setProperty //código /> --Para mostrar el valor de una propiedad de un JavaBean

3: Comentarios:

Para una sola línea se utilizará el formato "//" y varias líneas /* código */.

4: Indentación del código:

Debe ser a cuatro espacios sin caracteres de tabulación.

5: Llaves entre bloques de código:

Siempre utilizar llaves para encerrar bloques de código en las sentencias (if, for, while, do, function…). La llave de apertura debe estar debajo de la línea de la sentencia dada.

```
while (true)
{
    sentencia
}
```

? En el ejemplo anterior, puesto que se trata de una sola línea se pudieran haber obviado las llaves pero la presencia de estas brinda una mejor claridad y limpieza al código.

6: Estructuras de control: Deben tener un espacio entre la sentencia de la estructura y el signo de apertura del paréntesis para distinguirlas de las llamadas a funciones.

```
if ((condition1) || (condition2))
{
    //sentencia1;
}
```

7: Llamadas a funciones: Las funciones deben llamadas sin espacio entre el nombre de la función, el paréntesis que abre y el primer parámetro.

```
$variable = buscarNombre(String ci);
```

8: Espacios entre signos: Deben existir un espacio entre los signos y el resto de las sentencias (variables, valores…). Para el caso del uso de los signos coma "," o signo punto y coma ";" se debe dejar un espacio luego de su aparición.

```
int i;

for(i=5;i<=j;i++)
```

? INAPROPIADO: Existe poca claridad en el código al estar los signos, sentencias, variables y valores juntos.

```
int i;

for (i = 5; i <= j; i++)
```

? Permite comprender mejor el código expuesto.

Definición de nombres.

Todas las palabras utilizadas deberán estar en idioma español, sin usar tildes y para el caso de la letra ñ se usarán con dos n (ej, disenno).

General: Todo lo que sea clase o atributo que se refieran a nombres de tablas o campos respectivamente de la base de datos se le pondrá el mismo nombre y en caso de que en la base de datos aparezcan nombres de campos con alguna unión del tipo guión bajo (_) la letra que le sigue a este guión empezará con mayúscula

Ejemplos; Tabla: usuario_ensayo, Campo: nombre_usuario

Clase: UsuarioEnsayo, Atributo: nombreUsuario

1. Clases: Las clases controladoras, entidades (Bean) o DAO se nombrarán en el formato ecNombreClase, donde existirá el prefijo ec (Ensayos Clínicos) y cada nueva palabra siguiente comenzará con mayúscula.
2. Variables:

   Las variables que sean atributos de clase se representarán en el formato nombreVariable, donde la primera letra siempre se escribirá en minúscula y a cada nueva palabra comenzará con mayúscula, y se deberán escoger palabras descriptivas con respecto a la intención de la variable, pero a la vez simples.

Las variables temporales o auxiliares utilizadas en métodos o funciones aisladas deberán escribirse con la menor cantidad posible de caracteres (i, j, q), excepto si se trata de variables de retorno, que seguirán la especificación anterior.

3. Métodos de clases: Las métodos principales de las clases se representarán en el formato nombreMétodo, donde la primera letra siempre se escribirá en minúscula y a cada nueva palabra comenzará con mayúscula.
4. Funciones aisladas: Las funciones se representarán en el formato nombreFunción, donde la primera letra siempre se escribirá en minúscula y a cada nueva palabra comenzará con mayúscula.

Buenas prácticas.

Optimizar llamadas a funciones: Siempre que se implemente un algoritmo que requiera de un ciclo o algún tipo de solución que haga sucesivas llamadas a cierta función y que esta devuelva el mismo valor en cada una de estas iteraciones debe utilizarse una variable fuera del ciclo que recoja el valor de la misma y utilizarse luego en el algoritmo dado.

```
for (i = 0; i < lista.size(); $i++)
{
}
```

? INAPROPIADO: En cada iteración del ciclo se hacen llamadas sucesivas a la función size, la cual devolvería el mismo valor. Afecta en gran medida el rendimiento de la aplicación.

```
int aux = lista.size();
for (i = 0, i < aux; i++)
{
}
```

? Sólo se hace una primera llamada a la función size, se almacena en una variable aux y luego se utiliza la misma para cada iteración del ciclo.

Cadena de texto entre comillas: Siempre usar las comillas simples a menos que se necesite hacer interpolación de variables, en cuyo caso se usarán las dobles. El uso de comillas simples aumenta la velocidad de procesamiento de las páginas.

Anexo 2. Salidas del método de evaluación ATAM: Riesgos.

R1. Al no existir decisiones arquitectónicas para contrarrestar cualquier fallo en el sistema puede traer consigo graves problemas, ya que los datos no podrán ser recuperados sin ser enviados previamente a la BD.

Anexo 3. Salidas del método de evaluación ATAM: No Riesgos.

N1. En la clase SecureController se define el método abstracto mayProceed que es el encargado de verificar los permisos del usuario, las clases que heredan de SecureController redefinen este método propiamente para la clase en cuestión.

N2. En la clase SecureController se implementa el método Seguridad, el cual permite que no acceda al sistema desde direcciones IP que no tengan acceso.

N3. Para lograr que se mantenga una copia de la integridad de la BD primaria se debe implementar en el despliegue un sistema de replicas de BD garantizando que se mantengan todos los datos en caso de cualquier fallo.

N4. El uso del protocolo HTTPS para la transferencia de la información permite que se cree un canal cifrado, de este modo se consigue que la información sensible no sea interceptada por un atacante ya que lo único que obtendrá será un flujo de datos cifrados.

N5. El uso del SGBD PostgreSQL permite que se acceda rápido a la información por parte de los usuarios

N6. Al implementar procedimientos almacenados se logra que las respuestas de las consultas de las BD sean más rápidas a la petición de los usuarios.

N7. Al utilizar el patrón arquitectónico MVC se separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario; permitiendo desarrollar el modelo ,la vista y el controlador de forma independiente, así como realizar modificaciones en sus partes, sin afectar a las demás.

N8. Cuando se utiliza el patrón de diseño DAO se puede ocultar la implementación utilizada por si se diera la necesidad de cambiarla en el futuro, permitiendo así modificar o agregar nuevas funcionalidades en el sistema.

N9. Elegir el uso de las Hojas de Estilo CSS es una decisión que garantiza la fácil evolución del sistema ya que están definidos estilos por defecto en una archivo css y aunque se agreguen nuevas funcionalidades se pueden utilizar los mismos sin tener que redefinirlos de nuevo permitiéndola modificabilidad y mantenibilidad del sistema.

N10. Utilizar la tecnología Web garantiza que sea publicada la aplicación a través del servidor de aplicaciones Apache Tomcat en unos de los servidores de CIM.

Anexo 4. Salidas del método de evaluación ATAM: Puntos de Sensibilidad.

S1. Desarrollar la clase SecureController es un punto de sensibilidad para lograr que la autentificación del usuario sea especificando su rol, usuario y contraseña.

S2. Desarrollar la clase SecureController es un punto de sensibilidad para lograr que sólo puedan acceder a la aplicación, clientes a través de direcciones IP específicas bien controladas.

S3. El uso del protocolo seguro HTTPS constituye un punto de sensibilidad al encriptar todo el tráfico de información.

S4. Normalizar la BD constituye un punto de sensibilidad para lograr que los tiempos de respuesta a las peticiones realizadas por los usuarios sean mínimos.

S5. Elegir el patrón arquitectónico MVC constituye un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

S6. El patrón de diseño DAO establece un punto de sensibilidad para lograr un diseño con capacidad para la adición de nuevas funcionalidades y modificaciones futuras.

S7. La decisión de utilizar tecnología Web es un punto de sensibilidad para poder acceder al sistema desde cualquier parte del país.

Anexo 5. Salidas del método de evaluación ATAM: Puntos de Trade-off. No se detectaron puntos de trade-off ya que no existen decisiones que intervengan en la respuesta del sistema ante diferentes atributos

Glosario de Términos

ADL: Lenguaje de descripción de arquitectura cuyas siglas se derivan de su nombre en inglés Architecture Description Language y su función como su nombre lo indica es describir una AS.

Árbol de Utilidad: 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.

Contenedor de Servlets: Es un SHELL de ejecución que maneja e invoca Servlets por cuenta del usuario.

CVS (Concurrent Versions System): Es una aplicación informática que implementa un sistema de control de versiones.

GRASP (General Responsability Assignment Software Patterns): Son patrones generales de software para asignación de responsabilidades.

Herramientas CASE: Conjunto de aplicaciones informáticas orientadas al incremento de la productividad en el desarrollo de software, las siglas CASE vienen dadas por su nombre en inglés Computer Arded Software Engineering que se conoce como Ingeniería de Software Asistida por Computadoras.

JDBC (Java Database Connectivity): Es una API que permite la ejecución de operaciones sobre bases 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.

Stakeholders: Personas u organizaciones que están activamente implicadas en el negocio, ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto.

Servlets: Son objetos Java que se ejecutan en el servidor en respuesta a una solicitud de navegador. Pueden generar HTML o XML directamente, o llamar a JSP para producir la salida.