**Universidad de las Ciencias Informáticas**

**Facultad 6**

# Título: Propuesta de Arquitectura para el Simulador del Sistema Inmune J-IMMSIM 2.0

**Trabajo de Diploma**

Presentado para optar por el título de Ingeniero en Ciencias Informáticas

**Autores:** Diana González Delgado

Emily Dominguez Mejias

**Tutores:** Ing. Yunet González Mulet

Ing. Edgar Rojas Ricardo

**Consultante**: Lic. Milena Gómez Navarro

**Junio, 2008**

**Ciudad de La Habana**

## 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 firmamos la presente a los 15 días del mes de junio del año 2008.

Emily Dominguez Mejias

_____

Diana González Delgado

_____

Ing. Yunet González Mulet

_____

Ing. Edgar Rojas Ricardo

_____

## DATOS DEL CONTACTO

Ing. Yunet González Mulet

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

ygonzalezmu@uci.cu

Ing. Edgar Rojas Ricardo

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

erojas@uci.cu

## AGRADECIMIENTOS

*A la Revolución Cubana y a Fidel por habernos dado la oportunidad de estudiar y superarnos hasta convertirnos en profesionales con una alta cultura integral.*

*A nuestros padres por su apoyo en todo momento, todo lo que somos se lo debemos a ellos.*

*A nuestros familiares por su afecto.*

*A nuestros amigos por su fidelidad.*

*A nuestros compañeros de aula por enseñarnos tantas cosas, hemos pasado junto a ustedes momentos inolvidables.*

*A nuestros tutores Yunet y Edgar por su apoyo y esmero.*

*A René Lazo y a Yudel Quintana por darnos su ayuda desinteresadamente.*

*A aquellas personas que nos hacen felices.*

*Con todo nuestro cariño, muchas gracias.*

***Diana y Emily***

## DEDICATORIA

*A mis padres queridos Irene y Juan José por su cariño incondicional, por guiarme y forjar los valores que poseo. Por hacer posible que se cumplieran mis sueños.*

*A mi hermano Juan Manuel por su bondad y ser el motivo de mi superación.*

*A mis abuelos Miriam, Cary y Delgado por sus mimos y su infinito cariño y ternura.*

*A mi neno por su inmenso amor y hacerme sentir especial. Por acompañarme estos cinco años y hacerme tan feliz.*

*A mima y familiares de mi neno por acogerme cariñosamente y brindarme todo su afecto.*

*A mis santis Emily, Yadira, Karina y Yanet, por brindarme su linda amistad y apoyarme en todo momento.*

*A Ana Isabel por su generosidad y cariño.*

*A mis amigos de la Lenin por pasar junto a mí tantos momentos lindos.*

*A todas aquellas personas que quiero y me quieren.*

***Diana***

*A mis padres Amalia y Raymundo por sus sabios consejos, su devoción y su generosidad. Por educarme tan bien, darme tanto cariño y apoyo en todo momento.*

*A mi abuelita Lourdes por su ternura y comprensión. Por quererme muchísimo.*

*A mi coti por su amor, paciencia y ternura. Por brindarme en tan poco tiempo tantas cosas.*

*A Beba por su bondad, a René por sus enseñanzas y demás familiares por acogerme como una hija y brindarme todo su afecto.*

*A las santis Diana, Karina, Martha, Yadira y Yanet por estar conmigo todos estos años y compartir tantos momentos juntas. Por ser las amigas incondicionales.*

*A Rey, Baby, Israel y familiares por su hospitalidad y cariño.*

*A todos mis amigos, en especial a Luis y a Rebeca por su lealtad.*
*Y a aquellos que de una manera u otra me apoyaron y siempre estuvieron pendientes de mí.*

***Emily***

## Resumen

En el mundo de la Biología, las herramientas informáticas constituyen un gran avance pues apoyan a los biólogos en sus estudios, haciendo estos más ágiles y fructíferos. Dentro de estas herramientas los simuladores ocupan un lugar importante debido a que permiten observar comportamientos biológicos que resultan muy útiles en sus investigaciones. Específicamente los simuladores del Sistema Inmune (SI) han sido y son de gran utilidad para aquellos investigadores dedicados al estudio de dicho sistema. Para la creación de estas herramientas informáticas es vital una correcta estructuración de sus componentes, con el objetivo de simular lo más cercano a la realidad el funcionamiento de este sistema.

La Arquitectura de Software (AS) es la disciplina que se encarga del proceso de estructuración de un sistema informático. Los simuladores del SI que existen en el mundo no presentan una arquitectura documentada, por lo que no se puede reutilizar el conocimiento que le aplicaron los autores para conformar su estructura.

En Cuba se está desarrollando el simulador Java Immunology Simulator versión 1.0 (según sus siglas en inglés J-IMMSIM) que presenta una arquitectura concentrada, es decir, sus componentes estarán funcionando en una sola computadora. Esto no es lo más óptimo para dicho simulador por la gran cantidad de cálculo que deberá procesar. Se desarrollará una versión 2.0 del J-IMMSIM que elimine las limitaciones de la versión anterior, con este objetivo se presenta en este trabajo el diseño de una AS bien definido y robusto, asegurando que el producto final cumpla con los requerimientos especificados para este tipo de simulador.

PALABRAS CLAVES: Arquitectura de Software, Sistema Inmune, simulador y J-IMMSIM.

## Tabla de contenido:

**AGRADECIMIENTOS** ..... I
**DEDICATORIA** ..... II
**Resumen** ..... III
**Introducción** ..... 1

**Capítulo 1: Fundamentación teórica** ..... 3
**Introducción** ..... 3
**1.1 Arquitectura de Software** ..... 3
1.1.1 Surgimiento ..... 3
1.1.2 Definición ..... 4
1.1.3 Importancia de la Arquitectura de Software ..... 6
1.1.4 Clasificaciones de Arquitectura ..... 6
1.1.5 Arquitectura para simuladores ..... 7
**1.2 Estilos arquitectónicos** ..... 8
1.2.1 Definición de estilo ..... 8
1.2.2 Catálogo de estilos ..... 8
1.2.3 Ventajas de utilizar estilos: ..... 12
**1.3 Patrones** ..... 12
1.3.1 Definición de patrón ..... 13
1.3.2 Clasificación de Patrones ..... 13
1.3.2.1 Patrones de arquitectura ..... 13
1.3.2.2 Patrones de diseño ..... 16
1.3.3 Ventajas de los patrones: ..... 16
**1.4 Diferencias entre estilos y patrones** ..... 17
**1.5 Lenguajes de descripción de la arquitectura (ADL)** ..... 17
1.5.1 Lenguaje unificado de modelado (UML) ..... 18
**1.6 Metodologías, tecnologías y herramientas** ..... 18
1.6.1 Metodologías de desarrollo de software ..... 18
1.6.1.1 El rol de arquitecto de software según la metodología propuesta ..... 19
1.6.2 Tecnologías y herramientas ..... 20
1.6.2.1 Plataformas de desarrollo ..... 20

1.6.2.2 Patrones de JEE....21
1.6.2.3 Frameworks de JEE....24
1.6.3 Entorno Integrado de Desarrollo (IDE)....28
1.6.4 Entorno de ejecución....30
1.6.5 Herramientas CASE (Computer-Aided Software Engineering)....30
1.6.6 Gestor de base de datos....31
**Conclusiones:**....33

**Capítulo 2: Descripción de la arquitectura**....34
**Introducción**....34
**2.1 Características del J-IMMSIM 2.0**....34
**2.2 Organización del J-IMMSIM 2.0**....35
**2.3 Metas y restricciones**....36
2.3.1 Requisitos Funcionales....36
2.3.2 Requisitos no funcionales....38
**2.4 Vistas arquitectónicas**....40
2.4.1 Vista de Casos de Uso....40
2.4.2 Vista Lógica....44
2.4.3 Vista de Despliegue....46
2.4.4 Diagrama de componentes....50
2.4.5 Vista de Implementación....51
**Conclusiones**....52

**Capítulo 3: Evaluación de la arquitectura**....53
**Introducción**....53
**3.1 Evaluación de la AS**....53
**3.2 Importancia de evaluar la arquitectura**....53
**3.3 Etapas en las que se evalúa la arquitectura**....53
**3.4 Atributos de calidad**....54
**3.5 Técnicas de evaluación**....55
3.5.1 Evaluación basada en escenarios....56
3.5.2 Evaluación basada en simulación....57
3.5.3 Evaluación basada en modelos matemáticos....57

3.5.4 Evaluación basada en experiencia ..... 57
**3.6 Métodos de evaluación** ..... 58
3.6.1 Software Architecture Analysis Method (SAAM) ..... 58
3.6.2 Architecture Trade-off Analysis Method (ATAM) ..... 59
3.6.3 Active Reviews for Intermediate Designs (ARID) ..... 60
**3.7 Evaluando la arquitectura del J-IMMSIM 2.0** ..... 61
**Conclusiones** ..... 64

**Conclusiones generales** ..... 65
**Recomendaciones** ..... 66
**Referencias bibliográficas** ..... 67
**Anexos:** ..... 72
**Glosario de términos** ..... 73

## Índice de Figuras:

Figura 1 Estilo Arquitectura en Capas....15
Figura 2 Vista estilística en capas....28
Figura 3 Componentes del Sistema Inmune....34
Figura 4 Organización del J-IMMSIM 2.0....35
Figura 5 Vista de Casos de Uso....42
Figura 6 Vista Lógica....45
Figura 7 Vista de despliegue....49
Figura 8 Diagrama de componentes....50
Figura 9 Vista de implementación....52
Figura 10 Diagrama de casos de uso....72

# Índice de Tablas:

Tabla 1 Evaluación de los casos de uso ..... 41

Tabla 2 Breve descripción: Caso de uso Simular ..... 43

Tabla 3 Breve descripción: Caso de uso Gestionar Modelo ..... 43

Tabla 4 Evaluación en el escenario Gestionar Usuario ..... 62

Tabla 5 Evaluación en el escenario Capa de Negocio ..... 62

Tabla 6 Evaluación en el escenario BD ..... 62

Tabla 7 Evaluación en el escenario Capa de Acceso a Datos ..... 63

Tabla 8 Evaluación del escenario Capa de Presentación ..... 63

Tabla 9 Evaluación del escenario Capa de Acceso a Datos ..... 64

# Introducción

El Sistema Inmune es un complejo número de órganos y células que producen anticuerpos para proteger al cuerpo de sustancias extrañas como bacterias, virus y toxinas. Está formado por un conjunto de mecanismos que protegen a un organismo de infecciones por medio de la identificación y eliminación de agentes patógenos. Debido a que los patógenos abarcan desde virus hasta gusanos y parásitos intestinales, esta tarea es extremadamente compleja y las amenazas deben ser detectadas con absoluta especificidad distinguiendo los patógenos de las células y tejidos normales del organismo. A ello hay que sumar la capacidad evolutiva de los patógenos que les permite crear formas de evitar la detección por el sistema inmunológico e infectar al organismo huésped.

Durante años los biólogos han estado estudiando el comportamiento del SI, pero debido a la alta complejidad que este presenta, surgió la necesidad de crear una herramienta informática para agilizar estos estudios. Dicha herramienta consiste en la simulación de las interacciones de los entes biológicos, permitiéndoles a los biólogos guardar los resultados obtenidos para futuras investigaciones.

Actualmente en el mundo existen varios simuladores del SI, entre los principales están el IMMSIM++ y el C-IMMSIM. Estos simuladores no presentan una AS documentada, por lo que no se puede reutilizar el conocimiento que se aplicó a los mismos para estructurarlos.

En Cuba se está desarrollando el J-IMMSIM en su versión 1.0, el cual se implementa en Java (lenguaje de alto nivel). Este presenta una arquitectura concentrada, es decir, los tres componentes que lo conforman estarán funcionando en una computadora y como estos manejan grandes cantidades de cálculo de alta complejidad, una sola computadora no será suficiente para que el J-IMMSIM 1.0 trabaje con eficiencia y rapidez. Además no cuenta con una base de datos en la cual los biólogos podrían guardar los resultados de sus simulaciones y utilizarlos en posteriores estudios

Para erradicar las limitaciones antes expuestas, se implementará el J-IMMSIM versión 2.0, el cual necesitará de una estructura adecuada, de tal manera que sus componentes se comuniquen correctamente y que comprenda las propiedades visibles externamente de los mismos, así como las relaciones entre ellos, por lo cual se tiene como **problema científico**: ¿Cómo lograr la integración entre los componentes del J-IMMSIM 2.0?

En aras de brindar solución a la problemática anterior se define como **objeto de estudio**: La Arquitectura de Software.

**Campo de acción**: Arquitectura para simuladores.

**Objetivo general**: Diseñar una arquitectura para el J-IMMSIM 2.0.

**Objetivos específicos**:

- Conformar la línea base de la arquitectura.
- Describir la arquitectura del JIMMSIM 2.0.
- Evaluar la efectividad de la arquitectura diseñada.

**Tareas de la investigación**:

- Selección de los estilos y los patrones arquitectónicos a utilizar en el desarrollo de la arquitectura.
- Selección de las metodologías, tecnologías y herramientas a utilizar para el desarrollo del J-IMMSIM 2.0.
- Identificación de los casos de uso, clases del diseño y componentes arquitectónicamente significativos.
- Diseño de las vistas del sistema.
- Confección del documento descripción de la arquitectura.
- Selección de los métodos y técnicas de evaluación.
- Especificación de las ventajas y riesgos del diseño.

**Estructura del documento:**

**Capítulo 1. Fundamentación teórica**: Se describen los conceptos básicos de la AS, se analizan y seleccionan los estilos y los patrones arquitectónicos a utilizar en el desarrollo de la arquitectura. Se fundamenta desde el punto de vista teórico las metodologías, herramientas y tecnologías propuestas para el desarrollo del J-IMMSIM 2.0.

**Capítulo 2. Descripción de la arquitectura:** Se realiza la descripción del diseño arquitectónico del J-IMMSIM 2.0. Se identifican los casos de uso, clases del diseño y componentes arquitectónicamente significativos y se representan las características del sistema en las vistas arquitectónicas.

**Capítulo 3. Evaluación de la arquitectura:** Se analizan y seleccionan los métodos de evaluación y atributos de calidad más adecuados de acuerdo a las características del sistema. Se especifican las ventajas y riesgos del diseño y se muestran los resultados de la evaluación obtenidos después de aplicar tales métodos.

# Capítulo 1: Fundamentación teórica

## Introducción

En este capítulo se comparan los diferentes conceptos de AS definidos por varios autores. Se describen las características de los principales estilos y patrones arquitectónicos, argumentando el más adecuado para la organización del sistema. Se analizan y seleccionan las metodologías, herramientas y tecnologías a utilizar en el desarrollo del J-IMMSIM 2.0. Se explica el rol de arquitecto que propone la metodología utilizada.

## 1.1 Arquitectura de Software

En los inicios de la informática, la programación se consideraba un arte, debido a la dificultad que entrañaba el proceso de desarrollo de software. Con el tiempo fue aumentando la complejidad y el tamaño de los sistemas informáticos, por lo que los algoritmos y estructuras de datos no constituyen los principales problemas de diseño. Cuando los sistemas están construidos a partir de muchos componentes, la organización del sistema presenta un nuevo conjunto de problemas de diseño. Todo este proceso que define cómo va a estar estructurado el sistema se denomina Arquitectura de Software.

### 1.1.1 Surgimiento

La AS remonta sus antecedentes en la década de 1960, donde Edsger Dijkstra y otros científicos plantearon una serie de ideas que se acercaban a lo que es hoy la AS. En 1968 propusieron que se estableciera una estructuración correcta de los sistemas de software antes de programar.

El primer estudio en que aparece la expresión Arquitectura de Software en el sentido en que hoy se conoce, es sin duda el de Perry y Wolf; que ocurrió en 1992, aunque el trabajo se fue generando desde 1989, por lo que se puede decir que ellos fundaron la disciplina.

En la década de 1990 surgieron las contribuciones más importantes por parte de los miembros de la Universidad Carnegie Mellon: David Garlan, Mary Shaw, Paul Clements y Robert Allen, aportando valiosos conceptos a la AS.

En 1995 y 1996 surgieron dos textos fundamentales (el de la Banda de los Cuatro y la serie POSA) donde aparecían las primeras descripciones de los patrones. El primero de ellos promueve una expansión de la programación orientada a objetos, mientras que el segundo desenvuelve un marco ligeramente más ligado a la AS.

En el siglo XXI, la AS aparece estrechamente relacionada a la Ingeniería de Software, por esto se considera que todo lo que se ha hecho en ingeniería debe formularse de nuevo, integrando la AS en su conjunto. La AS se encuentra en una etapa de formación, constantemente están surgiendo nuevos aportes que desarrollan y amplían la disciplina. Hoy día existe un enorme repertorio de ideas, experiencias e instrumentos que ayudan a pensar de qué manera se pueden mejorar las prácticas. Existen varios conceptos de la AS desde el punto de vista de sus autores y cada uno de estos aporta ideas que van enriqueciendo la definición de la misma.

### 1.1.2 Definición

El significado de la expresión Arquitectura de Software resulta bastante claro, tal vez por ello no existe una definición única y aceptada de la misma. Es muy habitual citar varias de estas de manera simultánea, con el fin de proporcionar una visión de conjunto.

Bass y sus colegas definen este término de la siguiente forma:

*"La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes del software, las propiedades de esos componentes visibles externamente y las relaciones entre ellos."* [1]

La definición presentada anteriormente enfatiza el papel de los componentes del software en cualquier representación arquitectónica, que pueden ser tan simples como un módulo de programa y tan complicados como incluir bases de datos y las relaciones pueden ser tan sencillas como una llamada de procedimientos de un módulo a otro o tan complicadas como un protocolo de acceso a una base de datos. [1]

Otra de las definiciones es la que dan Mary Shaw y David Garlan, donde sugieren que la AS sea un nivel del diseño referido a las ediciones:

*"(…) más allá de las estructuras de los algoritmos y de datos del cómputo; diseñar y especificar la estructura del sistema total emerge como nueva clase de problema. Las ediciones estructurales incluyen la organización gruesa y la estructura global del control; los protocolos para la comunicación, la sincronización y el acceso a los datos; asignación de la funcionalidad para diseñar elementos; distribución física; composición de los elementos del diseño; escalamiento y funcionamiento; y selección entre alternativas del diseño*" [2]

La definición que se considera que describe mejor la AS con respecto a las anteriores es la de Clements:

*"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 la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema (...)*" [3]

A partir del año 2000 la IEEE publicó la definición oficial de AS:

"*La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución".* [3]

De los conceptos antes expuestos, se puede decir que tienden a identificar una noción de estructura Estos siempre plantean de una manera u otra la idea de AS como la forma de organizar un sistema, que se basa en la descripción de sus componentes y la forma en que se relacionan.

La AS establece los fundamentos para que los miembros de un proyecto trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema. Brinda a los desarrolladores mayor control en el sistema en el proceso de desarrollo desde sus inicios y promueve la temprana identificación y prevención de problemas. Como resultado, la arquitectura puede guiar el proyecto al éxito en vez de llevarlo al fracaso por falta de entendimiento.

Todo sistema de software, no importa el tamaño del mismo, tiene una arquitectura que puede variar en calidad. La arquitectura captura el conjunto de decisiones principales de diseño que se hacen sobre un sistema. Las decisiones de diseño son elecciones hechas pensando en cómo el sistema se desarrollará y cómo funcionará, incluyendo estructura, organización, funcionalidad, comportamiento, o más propiedades no funcionales como la usabilidad.
A pesar que la arquitectura es fundamentalmente una actividad centrada en el diseño, afecta a todo el ciclo de vida. La arquitectura abarca lo esencial en un sistema, la influencia de los requerimientos, el desarrollo e implementación, la planeación de su evolución y su adaptación, de ahí su importancia en el desarrollo de las aplicaciones.

### 1.1.3 Importancia de la Arquitectura de Software

La importancia de la AS radica en que esta abarca todo el proceso para dar distribución y orden a los elementos de un sistema informático, desde los requerimientos que debe satisfacer el mismo y las restricciones a las que está sujeto, hasta las propiedades no funcionales y las decisiones de diseño que gobiernan esta estructura.

La importancia de la AS se evidencia en los siguientes aspectos:

1. Comunicación mutua: Los desarrolladores de un software pueden utilizar la AS como base para crear un entendimiento mutuo y comunicarse entre sí. Esto es muy importante para tomar futuras decisiones y formar un consenso común respecto a estas.
2. Decisiones tempranas de diseño: Mediante la AS se toman decisiones tempranas del diseño sobre un sistema, lo cual tiene un peso importantísimo para la mitigación de riegos potenciales, evitando que ocurran futuros desastres a gran escala.
3. Representaciones constructivas: Una descripción arquitectónica proporciona diagramas que brindan una representación del sistema, indicando los componentes y las dependencias entre ellos, los cuales constituyen una guía para el desarrollo.
4. Reutilización: La AS promueve la reutilización a gran escala de una cantidad importante de componentes y frameworks. Esto reduce los costos de diseño y la cantidad de código se simplifica.
5. Evolución: La AS estima los posibles cambios y los costos de las modificaciones a las que se puede someter un sistema, permitiendo comprender el grado de mejoramiento que este puede alcanzar.

[3]

### 1.1.4 Clasificaciones de Arquitectura

Uno de los factores más importantes a considerar en el diseño de una AS es su grado de concentración o su grado de distribución.

Una **AS concentrada** es aquella en la que hay una o pocas aplicaciones que concentran cada una un número elevado de funcionalidades y funcionan en una o pocas computadoras con poco flujo de comunicación exterior. Entiéndase por aplicación el módulo de software que puede funcionar autónomamente en una sola computadora. [4]

Una **AS distribuida** tiene varias o muchas aplicaciones funcionando en varias computadoras con un flujo importante de comunicaciones. Esta se basa en una colección de computadoras unidas por una red y con un sistema que les permite compartir recursos de hardware, software y datos. [4]

### 1.1.5 Arquitectura para simuladores

La AS concentrada permite desarrollos simples de muy corto alcance. Aunque debido al incremento de potencia de la computación personal permite resultados casi profesionales, la AS concentrada se queda corta rápidamente en simuladores complejos.

Cuando se quieren implementar simuladores complejos como los simuladores a tiempo real resulta casi absurdo pensar en una AS concentrada, la distribuida ofrece múltiples beneficios y sin duda es la candidata perfecta. [4]

Entre las principales ventajas de AS distribuida están:

- Evita la sobrecarga de procesador con cálculos sobre los modelos matemáticos y generación de la escena.
- Permite una mayor reutilización del código.
- El uso de computadoras personales reduce el coste inicial de implantación.
- Las computadoras personales son altamente fiables, se reparan fácilmente y se sustituyen de forma inmediata.
- El uso del mismo tipo de computadora para tareas distintas permite un coste de mantenimiento más reducido.

En estos casos es conveniente proponer una AS propia que se adapte a las condiciones de lo que se quiere lograr con el simulador en materia de funcionalidad y que garantice un desempeño adecuado cuando esté terminado el producto. Es importante tener en cuenta el hecho de que pueda ser multiplataforma, lo que amplía las posibilidades de uso del simulador, sobre todo en la actualidad mundial donde cada vez se hace más popular el uso de otros sistemas operativos a parte de la plataforma de Windows, como son los de la plataforma GNU/Linux. El factor fundamental en la arquitectura para simuladores además de definir su grado de concentración, es que esta se organice de manera eficiente y se puedan aprovechar mejor los recursos con la adecuada selección de los estilos y patrones arquitectónicos.

## 1.2 Estilos arquitectónicos

Para conformar la organización del sistema se analizan un conjunto de estilos que catalogan las formas básicas posibles de estructuras de software. Estos brindan una mejor comprensión del sistema mediante el uso de estructuras convencionales.

### 1.2.1 Definición de estilo

Un estilo arquitectónico define a una familia de sistemas informáticos en términos de su organización estructural. Describe componentes y las relaciones entre ellos con las restricciones del sistema, la composición asociada y el diseño para su construcción. Este encapsula decisiones esenciales sobre los elementos arquitectónicos y enfatiza restricciones importantes de los elementos y sus posibles relaciones. [5]

Los estilos se agrupan en familias por la forma en que definen la organización de los componentes y sus relaciones. Existen varias taxonomías de estilos, a continuación se describen los más representativos y vigentes.

### 1.2.2 Catálogo de estilos

**Estilos de Flujo de Datos**

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

- ✓ **Tuberías y filtros**

El sistema tuberías y filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. [5]

**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. Subestilos

característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

✓ **Arquitecturas de Pizarra o Repositorio**

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. [5]

Todo modelo de este tipo consiste en las siguientes tres partes:

1. Fuentes de conocimiento, necesarias para resolver el problema.
2. Una pizarra que representa el estado actual de la resolución del problema.
3. Una estrategia, que regula el orden en que operan las fuentes.

**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.

✓ **Arquitecturas en Capas**

Este estilo se define 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. Al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, en la que cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados, lo que significa que si se minimiza las dependencias entre capas, resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema. [5]

Ventajas

- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Admite muy naturalmente optimizaciones y refinamientos.
- Proporciona amplia reutilización.

Desventajas

- Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de rendimiento pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.

- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

✓ **Arquitecturas Orientadas a Objetos**

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de datos abstractos. Los objetos representan una clase de componentes que ellos llaman managers (administradores), debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. [5]

✓ **Arquitecturas Basadas en Componentes**

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica, los componentes son las unidades de modelado, diseño e implementación. Las interfaces están separadas de las implementaciones y conjuntamente con sus interacciones son el centro de incumbencias en el diseño arquitectónico. Las funcionalidades y propiedades de los componentes puedan ser descubiertas y utilizadas en tiempo de ejecución. [5]

**Estilos de Código Móvil**

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

✓ **Arquitectura de Máquinas Virtuales**

Esta arquitectura se conoce como intérpretes basados en tablas o sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. El seudo-programa a su vez incluye el programa mismo y el análogo que hace el intérprete de su estado de ejecución (o registro de activación). La máquina de interpretación incluye tanto la definición del intérprete como el estado actual de su ejecución. De este modo, un intérprete posee por lo general cuatro componentes:

1. Una máquina de interpretación que lleva a cabo la tarea.
2. Una memoria que contiene el seudo-código a interpretar.
3. Una representación del estado de control de la máquina de interpretación.
4. Una representación del estado actual del programa que se simula. [5]

**Estilos Peer-to-Peer**

Esta familia, también llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados mediante broadcast. [5]

✓ **Arquitecturas Basadas en Eventos**

Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos), un componente puede anunciar mediante difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento. [5]

✓ **Arquitecturas Orientadas a Servicios**

Esta construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces. Es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa.

Características:

- Un servicio es una entidad de software que encapsula funcionalidad de negocio y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.
- Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran. Los servicios son las unidades de diseño e implementación. [5]

✓ **Arquitecturas Basadas en Recursos**

Esta define recursos identificables y métodos para acceder y manipular el estado de esos recursos.

Las URL (Uniform Resource Locator) identifican los recursos y HTTP (HyperText Transfer Protocol) es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. Constituye una lectura creativa de la lógica dinámica que rige el funcionamiento de la web (una especie de ingeniería inversa de alto nivel), al lado de una propuesta de nuevos rasgos y optimizaciones, o restricciones adicionales. [5]

**Estilo propuesto**

Como resultado del análisis de los estilos anteriormente explicados se seleccionó el estilo Arquitectura en Capas perteneciente a la familia Llamada y Retorno, pues una forma óptima en que se puede estructurar el J-IMMSIM 2.0 es dividiéndolo en capas, donde cada una agrupa funcionalidades concretas del sistema. Estas capas pueden ser tratadas de forma independiente permitiendo un bajo acoplamiento en el sistema, de manera que si se cambia la implementación en una de estas, las otras se verán afectadas en un menor grado.

### 1.2.3 Ventajas de utilizar estilos:

- Los estilos promueven reutilización de diseño. Las soluciones de rutina con propiedades bien entendidas se pueden aplicar otra vez a nuevos problemas con alguna confianza.
- El uso de estilos puede conducir a una significativa reutilización de código.
- Es más fácil entender la organización de un sistema si se utilizan estructuras convencionales, pues estas evocan una fuerte imagen respecto a cuales son sus piezas y cómo se vinculan recíprocamente.
- El uso de estilos estandarizados sustenta la interoperabilidad.

## 1.3 Patrones

Para un buen diseño de la AS se necesita aplicar aquellos patrones que brinden las mejores soluciones a los problemas que surjan en un contexto determinado. El uso de patrones mejoran la calidad del sistema y la selección de estos constituyen una decisión fundamental de diseño en el desarrollo de un software.

Grady Booch afirma lo antes planteado en la siguiente frase:

"*Una arquitectura orientada a objetos bien estructurada está llena de patrones. La calidad de un sistema orientado a objetos se mide por la atención que los diseñadores han prestado a las colaboraciones entre sus objetos. Los patrones conducen a arquitecturas más pequeñas, más simples y más comprensibles*". [6]

### 1.3.1 Definición de patrón

Un patrón es la solución a un problema en un contexto y codifica conocimiento específico acumulado por la experiencia en un dominio. Los patrones surgen de la experiencia de seres humanos al tratar de lograr ciertos objetivos y capturan la experiencia existente y probada para promover buenas prácticas.

Christopher Alexander plantea:

"*Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma*". [7]

Existen diversas clasificaciones de los patrones de acuerdo a las soluciones que brindan, por lo que es necesario estudiarlos para un correcto uso de estos.

### 1.3.2 Clasificación de Patrones

Existen diversas clasificaciones de patrones: de análisis, de arquitectura, de diseño, de organización o proceso, de programación y los llamados idiomas, entre otros, pues varios autores que se refieren al tema agregan nuevos tipos, por lo que no hay un límite de la variedad y ejemplares.

En el ámbito de la Ingeniería de Software actualmente los patrones pueden aplicarse a nivel de: análisis de requisito, el diseño de la arquitectura, el diseño detallado, la interacción con el usuario y el código. En el diseño de la arquitectura del J-IMMSIM 2.0 se considera factible seleccionar aquellos patrones arquitectónicos y de diseño que brinden las mejores soluciones en beneficio de la calidad del sistema, estos se describen a continuación.

#### 1.3.2.1 Patrones de arquitectura

Los patrones arquitectónicos son la descripción de un problema particular y recurrente del diseño, que aparece en contextos de diseño específico y presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes

que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí. [8]

Estos patrones especifican las propiedades estructurales de un sistema, por tanto, la selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo del J-IMMSIM 2.0.

**Principales patrones arquitectónicos:**

**Layers**: Consiste en estructurar aplicaciones que pueden ser descompuestas en grupos de subtareas, las cuales se clasifican de acuerdo a un nivel particular de abstracción.

**Pipes and Filters**: Provee una estructura para los sistemas que procesan un flujo de datos. Cada paso de procesamiento está encapsulado en un componente filtro (filter). El dato pasa a través de conexiones (pipes), entre filtros adyacentes.

**Blackboard**: Se aplica para problemas cuya solución utiliza estrategias no determinísticas. Varios subsistemas ensamblan su conocimiento para construir una posible solución parcial o aproximada.

**Broker**: Puede ser usado para estructurar sistemas de software distribuido con componentes desacoplados que interactúan por invocaciones a servicios remotos. Es responsable de coordinar la comunicación, como el reenvío de solicitudes, así como también la transmisión de resultados y excepciones.

**Model-View-Controler**: Divide una aplicación interactiva en tres componentes. El modelo (model) contiene la información central y los datos. Las vistas (view) despliegan información al usuario. Los controladores (controlers) capturan la entrada del usuario. Las vistas y los controladores constituyen la interfaz del usuario.

**Presentation-Abstraction-Control**: Define una estructura para sistemas de software interactivos de agentes de cooperación organizados de forma jerárquica. Cada agente es responsable de un aspecto específico de la funcionalidad de la aplicación y consiste de tres componentes: presentación, abstracción y control.

**Microkernel**: Se aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema. Separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.

**Reflection**: Provee un mecanismo para sistemas cuya estructura y comportamiento cambia dinámicamente. Soporta la modificación de aspectos fundamentales como estructuras y mecanismos de llamadas a funciones. [9]

### Patrón arquitectónico propuesto:

El patrón arquitectónico que se propone que se aplique es el de capas (layers), pues según las características del J-IMMSIM 2.0, este se puede estructurar específicamente en tres de estas: la capa de Presentación, la capa lógica y la capa de Acceso a Datos como muestra la Figura 1. Las capas inferiores proporcionan servicios a las superiores y cada una se tratará de forma independiente, pues encapsulan un aspecto concreto del sistema. Este patrón es importante porque simplifica la comprensión y la organización del desarrollo del sistema, reduciendo las dependencias de forma que las capas más bajas no son conscientes de ningún detalle o interfaz de las superiores.

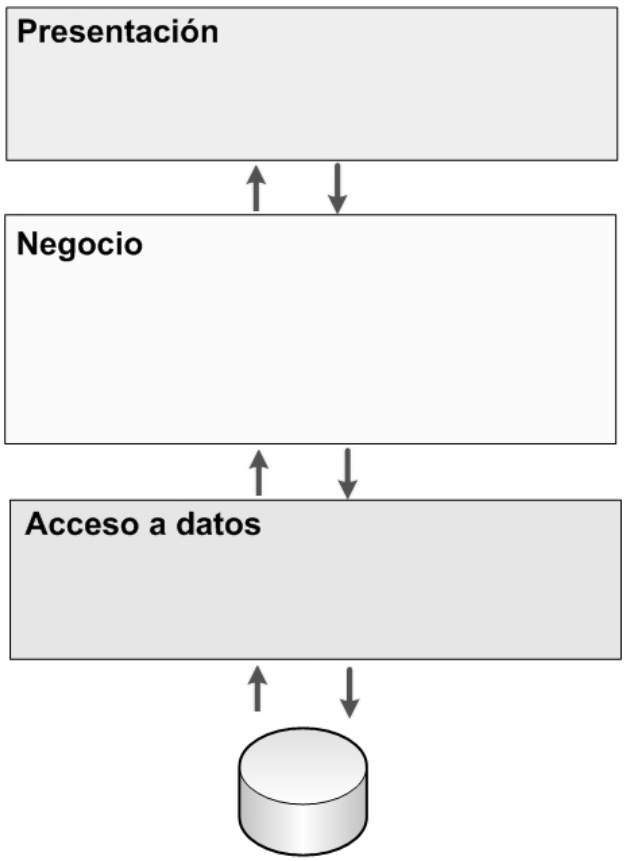


**Figura 1 Estilo Arquitectura en Capas**

#### 1.3.2.2 Patrones de diseño

Un patrón de diseño es una descripción de clases y objetos comunicándose entre sí, adaptada para resolver un problema de diseño general en un contexto particular. Estos identifican: clases, instancias, roles, colaboraciones y la distribución de responsabilidades (GRASP). [10]

**Patrones GRASP**

Los patrones GRASP (patrones generales de software para asignar responsabilidades) describen los principios fundamentales de la asignación de responsabilidades a objetos. [10] Deberán aplicarse los patrones GRASP que a continuación se mencionan, para una óptima asignación de responsabilidades a las clases del J-IMMSIM 2.0.

- Experto: Asigna responsabilidades a las clases que tienen la información necesaria para cumplir con la responsabilidad.
- Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental de este patrón es encontrar un creador que se conecte con el objeto producido en cualquier evento.
- Alta Cohesión: La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Este patrón se encarga de asignar una responsabilidad de modo que la cohesión siga siendo alta.
- Bajo Acoplamiento: El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Una clase con bajo acoplamiento no depende de muchas otras. Este patrón asigna una responsabilidad para mantener bajo acoplamiento.
- Controlador: Asigna la responsabilidad del manejo de un mensaje de los eventos de un sistema a las clases que representen un sistema global, una empresa u organización o algo en el mundo real que es activo. [10]

### 1.3.3 Ventajas de los patrones:

El uso de los patrones tiene grandes ventajas pues posibilita a los desarrolladores indistintamente de su experiencia y habilidad en el tema; un ahorro considerable en cuanto a tiempo y recursos. Los patrones son el resultado de la experiencia acumulada en el diseño de aplicaciones, de ahí que estos posibiliten elaborar diseños más simples, robustos, generales y factibles al cambio. Un buen diseño no

debe ser específico de una aplicación concreta, ya que debe basarse en soluciones que han funcionado bien en otras ocasiones, estas soluciones constituyen los patrones.

## 1.4 Diferencias entre estilos y patrones

Cuando se exponen los conceptos de estilos y patrones tienden a confundirse, por lo que es importante establecer las diferencias que existen entre ellos.

Los estilos definen una estructura organizacional concierne más bien a la teoría en el más alto nivel de abstracción, mientras que los patrones se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el refinamiento y el código.

Al contrario de los estilos arquitectónicos, los patrones son muchos y a su vez muy variados y es casi imposible revisar todos los patrones que existen a la hora de hacer una determinada aplicación, por eso se recomienda el uso de los patrones que estén asociados a cada uno de los estilos que se seleccionen para el desarrollo de la arquitectura. [5]

## 1.5 Lenguajes de descripción de la arquitectura (ADL)

Los lenguajes de descripción de la arquitectura (según sus siglas en inglés ADL) se utilizan para satisfacer requerimientos descriptivos de alto nivel de abstracción. Constituyen lenguajes descriptivos de modelado que se centran en la estructura de un sistema. Este proporciona un modelo explícito de componentes, conectores y sus respectivas configuraciones. [11]

Existe gran variedad de ADLs, los principales de estos que se mencionan a continuación:

**Acme**: Lenguaje de intercambio de ADL.

**Aesop**: ADL de propósito general, énfasis en estilos.

**Darwin**: ADL con énfasis en dinámica.

**Jacal**: Notación para la descripción y prototipado.

**MetaH**: ADL específico de dominio.

**Rapide**: ADL para simulación.

**UniCon**: ADL con énfasis en estilos.

**xADL**: Basado en XML.

**UML**: El Lenguaje Unificado de Modelado (UML) es un lenguaje semi-formal de modelado. Aunque este no es un ADL en el sentido usual de la expresión, constituye una herramienta de uso habitual en el modelado, aunque ya se piensa en el como un metalenguaje [11].

Se propone la utilización de UML para el diseño de la arquitectura pues permite especificar, visualizar, construir y documentar artefactos del sistema como las vistas arquitectónicas. Además constituye un lenguaje que la mayoría de los desarrolladores conocen. A continuación se describen sus principales características.

### 1.5.1 Lenguaje unificado de modelado (UML)

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Estos elementos se pueden representar mediante nueve tipos de diagramas, que son: de clases, de objetos, de casos de uso, de secuencia, de colaboración, de estados, de actividades, de componentes y de desarrollo. [10]

UML presenta características generales y razones por las que resulta interesante su aplicación para efectos de la representación de una AS. Permite el soporte para algunos de los conceptos asociados a las arquitecturas de software, como los componentes, los paquetes, las librerías y la colaboración. Además, admite la descripción de componentes en la AS en dos niveles; se puede especificar sólo el nombre del componente o especificar las clases o interfaces que implementan estos. [10]

## 1.6 Metodologías, tecnologías y herramientas

Para la construcción de un software se necesita contar con una metodología que guíe el proceso de desarrollo del mismo. Las herramientas y tecnologías brindan servicios imprescindibles para la implementación y funcionamiento del software. Por todo esto es necesario el estudio y selección de las mismas.

### 1.6.1 Metodologías de desarrollo de software

En el mundo existen varias metodologías con diferentes métodos y técnicas para guiar el desarrollo de software. Escoger la que más se ajuste es muy importante para organizar el proceso, por lo que es necesario comparar algunas de estas en aras de seleccionar la que resulte más factible para el desarrollo del J-IMMSIM 2.0.

**RUP**

RUP (Proceso Unificado de Rational) es un proceso de desarrollo de software que junto a UML constituyen la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos. Se caracteriza por ser iterativo e incremental, estar centrado en la arquitectura y guiado por casos de uso. [12]

A diferencia de otras metodologías, RUP es más apropiado para proyectos grandes, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. Agrupa las actividades en nueve flujos de trabajo principales, donde los seis primeros son conocidos como flujos de ingeniería y los tres últimos de apoyo. [13]

**OpenUP (Open Unified Process)**

El OpenUP es un proceso unificado que aplica propuestas iterativas e incrementales dentro del ciclo de vida. El uso del mismo se debe realizar cuando se tiene un equipo pequeño y se quiere evitar ser cargado excesivamente con metodologías formales e improductivas, por lo que propone no utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo este ser modificado, mejorado y extendido. Es un proceso ágil, ligero que promueve buenas prácticas para el desarrollo de software, haciéndolo pequeño, extensible si es necesario (híbrido, puede incluir partes de otros modelos). Aunque es un proceso de desarrollo de software simplificado, resulta bastante completo. [14]

### Metodología propuesta

Para guiar el desarrollo del J-IMMSIM 2.0 se utilizará la metodología OpenUP porque es la que más se corresponde a sus características, pues el equipo de trabajo es pequeño y además los artefactos que propone son precisamente los principales que se necesitan, obviando otros que no presentan gran importancia y hacen lento el proceso de desarrollo del software. Esta metodología define distintos roles donde cada uno se encarga de realizar las tareas asignadas por esta, entre estos roles están los arquitectos de software que son los encargados de la estructuración del sistema.

#### 1.6.1.1 El rol de arquitecto de software según la metodología propuesta

El arquitecto de software que define OpenUP debe ser una persona con madurez, visión y experiencia, capaz de comprender los problemas de manera rápida, de hacer juicios críticos en la ausencia de información completa. En concreto, el arquitecto debe poseer esta combinación de las siguientes calificaciones:

- Experiencia y dominio de la Ingeniería de Software, un profundo conocimiento de los requisitos para solucionar el problema y la participación activa en el desarrollo de software.
- Capacidad de liderazgo para motivar y mantener el impulso para el esfuerzo técnico a través de los integrantes del equipo y para tomar decisiones críticas bajo presión. Para ser eficaz, este rol debe tener la autoridad para tomar decisiones técnicas.
- Excelente capacidad de comunicación para ganarse la confianza, persuadir y motivar. El arquitecto debe ganarse el respeto de los miembros del equipo, del líder de proyecto, el cliente y de la comunidad de usuarios.

Para diseñar la arquitectura del J-IMMSIM 2.0 será necesario guiar el desarrollo del sistema, así como tomar las principales decisiones en el diseño global y la implementación del mismo. También se deberán identificar y documentar los aspectos arquitectónicamente significativos, como las vistas que describen los requisitos, el diseño, la implementación y el despliegue. De esto se encargará el arquitecto de software, que genera el documento descripción de la arquitectura como artefacto fundamental.

### 1.6.2 Tecnologías y herramientas

Las tecnologías y herramientas están evolucionando constantemente, por lo que brindan mejores prestaciones que facilitan a los desarrolladores la construcción de aplicaciones de mayor nivel. Resulta importante la correcta selección de cada una de estas, para explotar todos los servicios que ofrecen en beneficio del software.

#### 1.6.2.1 Plataformas de desarrollo

Una plataforma de desarrollo es el entorno de software en el cual se desenvuelve la programación de un grupo definido de aplicaciones. Comúnmente se encuentra relacionada directamente a un sistema operativo; sin embargo, también es posible encontrarla ligada a una familia de lenguajes de programación o APIs.[15]

La versión anterior (J-IMMSIM 1.0) se programó en el lenguaje Java, que es uno de los más elaborados y utilizados para la creación de software, por lo que se considera factible que el J-IMMSIM 2.0 utilice el mismo lenguaje para su implementación. Además con la evolución de Java se han desarrollado tres plataformas que constan de un gran conjunto de componentes que se pueden reutilizar y mecanismos para extenderlos, facilitando el trabajo a los desarrolladores. Dichas plataformas se refieren a continuación:

- **Java Plataform, Standard Edition, o JSE (antes J2SE):** Es una colección de APIs del lenguaje de programación Java útiles para muchos programas de la plataforma.
- **Java Plataform, Enterprise Edition, o JEE (antes J2EE):** Es para desarrollar y ejecutar software en lenguaje de programación Java con arquitectura de N niveles distribuida.
- **Java Plataform, Micro Edition, o JME (antes J2ME):** Es una colección de APIs de Java para el desarrollo de software para dispositivos de recursos limitados, como PDA (Asistente Digital Personal), teléfonos móviles y otros aparatos de consumo. [15]

**Plataforma propuesta**

Para facilitar el desarrollo del J-IMMSIM 2.0 se seleccionó la plataforma JEE que provee un conjunto de librerías, interfaces, patrones y frameworks que brindan una infraestructura para el desarrollo del sistema y arquitecturas de alto nivel. Esta abarca cada uno de los elementos como las transacciones, mensajerías, conexiones a base de datos, interfaces de usuarios, manejos de recursos, seguridad, entre otros. Además incluye todas las clases de la plataforma JSE.

Ventajas de JEE:

- Permite construir aplicaciones que se puedan ejecutar en varios tipos de plataformas o sistemas operativos.
- Permite reutilizar componentes y experiencias desarrolladas por otras aplicaciones.
- Es independiente del hardware, porque la Máquina Virtual de Java (MVJ, o JVM por sus siglas en inglés) hace posible que una aplicación desarrollada en Java se ejecute en cualquier sistema operativo que soporte esta máquina virtual, como por ejemplo Unix, GNU/Linux, Macintosh, Windows, entre otros.
- Es una plataforma de desarrollo madura, bien documentada y con amplio soporte de herramientas, frameworks, APIs y componentes, además presenta su propio conjunto de patrones de diseño que brindan diferentes soluciones. [15]

### 1.6.2.2 Patrones de JEE

Se propone que se apliquen los patrones de diseño que brinden las mejores soluciones a cada capa definida por el estilo y patrón arquitectónico seleccionado. La breve descripción de los principales patrones de diseño se expone a continuación:

**Capa de Presentación:**

| | |
|---|---|
| **Front Controller** | Plantea usar un controlador como el punto inicial de contacto para manejar las peticiones de los usuarios. |
| **View Helper** | Una vista (view) delega sus responsabilidades de procesamiento a sus clases de ayuda (helper). Un objeto helper encapsula la lógica de Acceso a Datos en beneficio de los componentes de la presentación. |
| **Composite view** | Define utilizar vistas compuestas por varias subvistas. [16] |

Se seleccionó el patrón **Front Controller** porque maneja de una forma centralizada las peticiones de los usuarios, evitando la duplicación de código que se puede generar en la capa de Presentación, de los usuarios acceder directamente sobre las vistas. Se explica a continuación con más detalles el patrón seleccionado.

**Front Controller**

Contexto**:** El mecanismo de manejo de peticiones de la capa de Presentación debe controlar y coordinar el procesamiento de todos los usuarios a través de varias peticiones. Dichos mecanismos de control se pueden manejar de una forma centralizada o descentralizada.

Problema**:** El sistema requiere un punto de acceso centralizado para que el manejo de peticiones de la capa de Presentación soporte la integración de los servicios del sistema, recuperación de contenidos, control de vistas y navegación. Cuando el usuario accede a la vista directamente sin pasar por un mecanismo centralizado, podrían ocurrir dos problemas:

- Se requiere que cada vista proporcione sus propios servicios del sistema, lo que normalmente resulta en duplicación de código.
- La vista de navegación se deja a los visores. Esto podría resultar en una mezcla de contenidos y navegación.

Además, el control distribuido es más difícil de mantener, ya que los cambios se tienen que realizar en numerosos lugares.

Solución: Usar un controlador como el punto inicial de contacto para manejar las peticiones, incluyendo la invocación de los servicios de seguridad como la autentificación y autorización, delegar el procesamiento de negocio, controlar la elección de una vista apropiada, el manejo de errores y el control de la selección de estrategias de creación de contenido.[15]

**Capa de Negocio**

| | |
|---|---|
| **Bussiness Delegate** | Un Business Delegate actúa como una abstracción de negocio del lado del cliente y por lo tanto oculta la implementación de los servicios del negocio. |
| **Transfer Object** | Define un Transfer Object para encapsular los datos del negocio. Se utiliza una única llamada a un método para enviar y recuperar el Transfer Object |
| **Service Locator** | Consiste en utilizar un objeto Service Locator para reducir la complejidad del código, proporcionando un punto de control y para fomentar la reutilización. [16] |

Se seleccionó el patrón **Bussiness Delegate** porque crea una capa de abstracción de negocio la cual sirve de punto de contacto entre la capa de Presentación y la de Negocio, evitando que estas interactúen directamente, pues de cambiar la implementación de los servicios de negocio se verán afectados los componentes de presentación. Se explica a continuación con más detalles el patrón seleccionado.

**Bussiness Delegate:**

Contexto: Un sistema multi-capa distribuido requiere invocación remota de métodos para enviar y recibir datos entre las capas. Los clientes están expuestos a la complejidad de tratar con componentes distribuidos.

Problema: Los componentes de la capa de Presentación interactúan directamente con servicios de negocio, exponiendo los detalles de la implementación de las APIs de una capa a otra. Como resultado, cuando cambia la implementación del servicio de negocio, también deberá cambiar el código expuesto de la capa de Presentación.

Solución: Utilizar un Bussines Delegate como una abstracción de negocio para reducir el acoplamiento entre los clientes de la capa de Presentación y los servicios de negocio, así como para ocultar los detalles de la implementación del mismo. Potencialmente, reduce el número de cambios que se deben hacer en el código de cliente de la capa de Presentación cuando cambie la API del servicio de negocio o su implementación subyacente. [15]

**Capa de Acceso a Datos**

| | |
|---|---|
| **Data Access Object (DAO)** | 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. |
| **Service Activator** | Se utiliza para recibir peticiones y mensajes asíncronos de los clientes. Cuando se recibe un mensaje, el Service Activator localiza e invoca a los métodos de los componentes de negocio necesarios para cumplir la petición de forma asíncrona. [16] |

Se seleccionó el patrón **Data Access Object** porque crea una capa de abstracción de acceso a datos la cual permite que los componentes de negocio interactúen directamente con esta y no con la fuente de datos, permitiendo la migración de gestor de base de datos sin que afecte la capa de Negocio. Se explica a continuación con más detalles el patrón seleccionado.

**Data Access Object**

Contexto: El acceso a los datos varía dependiendo de la fuente de los datos. El acceso al almacenamiento persistente, como una base de datos, varía en gran medida dependiendo del tipo de almacenamiento (bases de datos relacionales, bases de datos orientadas a objetos, ficheros planos, entre otros) y de la implementación.

Problema: Cuando los componentes de negocio necesitan acceder a una fuente de datos, pueden utilizar la API apropiada para conseguir la conectividad y manipularla. Pero introducir el código de conectividad y de acceso a datos dentro de estos componentes, genera un fuerte acoplamiento entre los mismos y la implementación de la fuente de datos. Dichas dependencias de código hace difícil y tedioso migrar la aplicación de un tipo de fuente de datos a otro, pues habría que cambiar la implementación de los componentes para manejar el nuevo tipo de fuente de datos.

Solución**:** Utilizar un DAO (Data Access Object) para abstraer, encapsular y manejar la conexión de todos los accesos a la fuente de datos para obtener y almacenar datos.

Los componentes de negocio que tratan con el DAO utilizan un interfaz simple y oculta los detalles de la implementación de la fuente de datos a sus clientes. Como la interfaz expuesta por el DAO no cambia cuando lo hace la implementación de la fuente de datos, este patrón permite adaptarse a diferentes esquemas de almacenamiento sin que esto afecte a sus clientes o componentes de negocio. [15]

### 1.6.2.3 Frameworks de JEE

Los frameworks (marcos de trabajo) constituyen implementaciones basadas en patrones de diseño, soluciones comunes normalmente empaquetadas para poder hacer más rápido el desarrollo haciendo uso de buenas prácticas.

Se propone que se utilicen los frameworks de la plataforma JEE para facilitar el desarrollo del J-IMMSIM 2.0 pues estos brindan una serie de funcionalidades abstrayendo a los desarrolladores de múltiples complejidades, permitiéndoles que se centren en la lógica del negocio. Se expone a continuación una breve descripción de los principales frameworks de la plataforma.

**Struts:** Este framework es recomendado para aplicaciones de gran tamaño y complejidad, que permite que el desarrollador se concentre en el diseño de estas como una serie simple de componentes del modelo y de la vista intercomunicados por un control centralizado. De esta manera puede obtenerse una aplicación más consistente y fácil de mantener.

**Java Server Faces (JSF):** Simplifica el desarrollo de la interfaz de usuario en una típica aplicación web y permite desarrollar sofisticados componentes de diseño de interfaz a la medida de sus necesidades. El estado de los componentes de la interfaz de usuario se guarda cuando el cliente solicita una nueva página y luego es restaurado al retornar de la petición. JSF se basa en el uso de Java Server Pages (JSP).

**Spring:** Facilita a los desarrolladores la implementación del software, promoviendo buenas prácticas de programación. Maneja objetos del negocio y además se integra fácilmente con otros frameworks. El framework Acegi está íntimamente ligado a Spring, el cual se encarga de manejar la seguridad de la aplicación.

**Hibernate**: Se encarga de gestionar la capa de Acceso a Datos de un software y realiza la transición de los datos de un modelo relacional a un modelo orientado a objetos. Además está basado en la implementación del patrón DAO, que crea una capa de abstracción que separa el negocio de la fuente de datos.

### Frameworks propuestos

Se propone que se apliquen los siguientes frameworks los cuales brindan facilidades y servicios para gestionar las funcionalidades de cada una de las capas en que esta organizado el J-IMMSIM 2.0 (Figura 2):

**Java Server Faces (JSF):** JSF se aplica a la capa de Presentación. Permite construir aplicaciones web que implementan una separación entre el negocio y la presentación, permitiéndole a cada miembro del equipo enfocarse en su parte del proceso de desarrollo.
Proporciona una rica arquitectura para manejar el estado de los componentes, procesar los datos, validar la entrada del usuario y manejar eventos. Finalmente, la tecnología JSF permite convertir y validar datos sobre componentes individuales y reportar cualquier error antes de que se actualicen los datos en el lado del servidor. [15]
Principales servicios que aporta JSF:

- Conversión de datos: Facilita la conversión de los datos entrados como texto por el usuario en los formularios web a distintos tipos como fechas, números y otros. Permite definir nuevas reglas de conversión.
- Validación y manejo de errores: Permite asociar reglas de validación a los campos y mostrar los mensajes de error apropiados.

- Desarrollo de componentes: Los programadores pueden desarrollar sofisticados componentes de diseño de interfaz a la medida de sus necesidades.
- Soporte de Herramientas: JSF está optimizado para ser desarrollado con herramientas automatizadas que agilicen el desarrollo de aplicaciones web.

**Spring**: Es un framework que tiene el objetivo de facilitar la construcción de aplicaciones Java. Se puede utilizar en cualquier tipo de aplicación, no solo en web. Spring es ligero por el mínimo impacto que tiene en las aplicaciones. [15]

Beneficios arquitectónicos:

- Spring puede organizar eficazmente la capa intermedia de cualquier aplicación ya sea web o escritorio.
- Facilita el desarrollo de buenas prácticas, reduce el costo de programación en una aplicación.
- Las aplicaciones con Spring son fáciles de testear.
- Permite la integración entre varios frameworks, constituyendo un eslabón central en la arquitectura de las aplicaciones.

El centro de Spring está basado en el principio de inyección de dependencias (IoC, Inversion of Control) Esta técnica hace externo la creación y el manejo de las dependencias de los componentes, logrando mayor limpieza y claridad en el código.

Con este mecanismo se obtienen los siguientes resultados:

- Reduce potencialmente el código de enlace entre los diferentes componentes de la aplicación.
- Externaliza las dependencias, lo cual permite la reconfiguración de las mismas sin necesidad de compilar todo el código.
- Permite el manejo de las dependencias en un solo lugar, facilitando la configuración de las mismas y disminuyendo el margen de errores.

[15]

**Acegi Security System**: Lleva a cabo la autenticación de los usuarios de la aplicación, el acceso a los recursos y el control de los permisos. Provee seguridad declarativa para las aplicaciones basadas en Spring, de esta manera libera al código de la aplicación de implementaciones de seguridad. En las aplicaciones web Acegi usa filtros que interceptan las solicitudes, utilizando un único mecanismo para declararlos e inyectarlos con sus dependencias mediante IoC. [15]

Ventajas que brinda Acegi:

- Provee configuración de protección del canal, permitiendo redireccionar hacia un canal adecuado (HTTP no seguro o HTTPS seguro) de acuerdo a la solicitud.
- Proporciona una librería de etiquetas que puede ser utilizada en JSP, para garantizar que el contenido protegido como enlaces y mensajes sean únicamente mostrados a usuarios que posean los permisos adecuados.
- Utiliza distintos métodos de almacenamiento de la información de autenticación.

**Hibernate**: Es un framework que constituye un motor de persistencia que implementa múltiples funcionalidades, utilizando ORM (mapeo objeto/relacional) que consiste en la técnica de realizar la transición de una representación de los datos de un modelo relacional a un modelo orientado a objetos y viceversa. [15]

Este framework es una clara implementación del patrón DAO, pues crea una capa separada que se ocupa del acceso a datos con total independencia del gestor y la base de datos, dando la oportunidad de trabajar con varios gestores y bases de datos dentro de la misma aplicación sin que esto cree ningún conflicto en los modelo de objetos. Soporta todos los sistemas gestores de bases de datos SQL y se integra de manera elegante y sin restricciones con los contenedores web más conocidos. [15]

En la siguiente figura se muestra cada uno de los frameworks propuestos que se aplicarán a las capas:

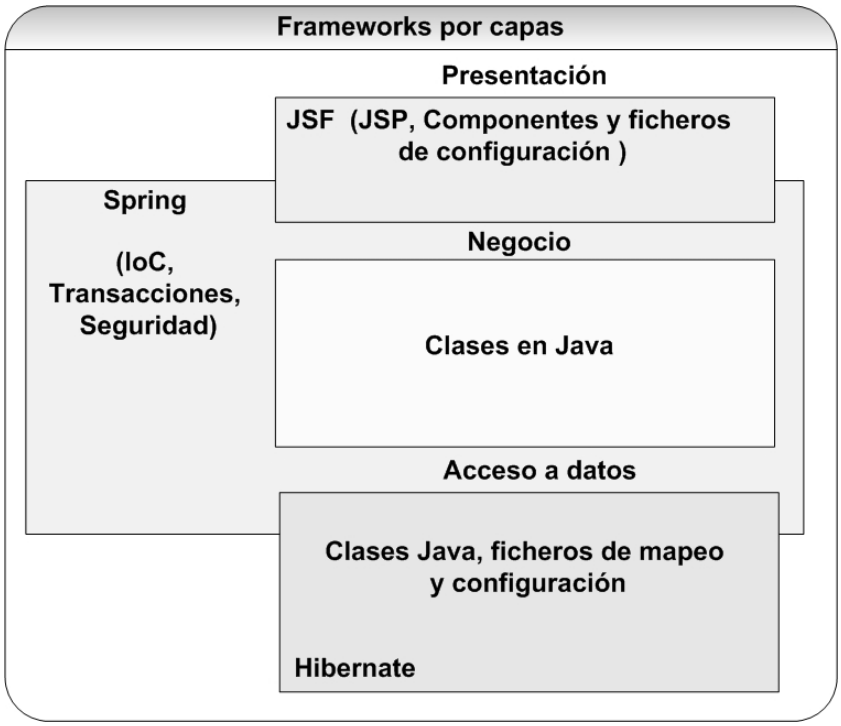


**Figura 2 Vista estilística en capas**

### 1.6.3 Entorno Integrado de Desarrollo (IDE)

Un entorno integrado de desarrollo es un programa compuesto por un conjunto de herramientas para un programador, permitiéndoles diseñar y construir aplicaciones así como dar soporte al desarrollo e implantación de las mismas.

Actualmente existe una gran competencia entre estos, mejorando para el beneficio de los programadores sus características, de acuerdo a estas se seleccionó el más factible para implementar el J-IMMSIM 2.0.

**NetBeans**

NetBeans se refiere a una plataforma para el desarrollo de aplicaciones de escritorio usando Java y a un entorno de desarrollo integrado (IDE) desarrollado usando la plataforma NetBeans.

La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases escritas para interactuar con las APIs de NetBeans. Las aplicaciones construidas a partir de módulos

pueden ser extendidas agregándole nuevos, debido a que estos pueden ser desarrollados independientemente. [17]

El NetBeans es un IDE para programadores pensado para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el IDE el cual es un producto libre y gratuito sin restricciones de uso. [17]

**Eclipse**

Eclipse está ganando prestigio entre los desarrolladores de Java, pues se puede utilizar para diseñar programas de alto nivel, componentes, sitios web y muchos otros elementos. Es basado en plugins y existen cientos de estos para programar en diversos lenguajes de programación como C++, Perl, PHP y XML. [18]

Ventajas

- Soporta la construcción de una variedad de herramientas para el desarrollo de aplicaciones.
- Corre en una gran cantidad de sistemas operativos incluyendo Windows y GNU/Linux.
- Mediante JDT (Java Development Tools) facilita la creación de aplicaciones programadas en Java. [18]

### Herramienta de desarrollo propuesta

Como herramienta de desarrollo se propone utilizar el Eclipse, porque el mismo tiene amplia documentación y soporte en la comunidad de Java. Es un potente editor con buen completamiento de código. Es basado en plugins que permiten agregar nuevas funcionalidades que se necesiten. Además es el más conveniente para el desarrollo de aplicaciones web y se integra fácilmente con el Tomcat que es el contenedor web que sustentará la aplicación.

Para agilizar el desarrollo con los frameworks propuestos, Eclipse brinda el plugin Exadel Studio 4.1 que tiene las siguientes ventajas:

- Agrega facilidades para el trabajo de configuración de Spring y JSF.
- Permite la generación de los ficheros de mapeo de Hibernate y de las clases persistentes a partir del esquema de la base de datos (cada fichero corresponde a una clase persistente y a su vez a una tabla en la base de datos).
- Agrega facilidades en el editor de las páginas JSP.

### 1.6.4 Entorno de ejecución

Se propone desarrollar el J-IMMSIM 2.0 como una aplicación web, pues es más ágil y sencillo en el despliegue y fácil de actualizar si esta se somete a cambios o reparaciones. En las aplicaciones de este tipo disminuyen los requerimientos de hardware en las computadoras de los usuarios y por tanto el costo de inversión. Estas están respaldadas por frameworks que apoyan el desarrollo en la plataforma. Las aplicaciones web, necesitan de un contenedor para su despliegue.

La aplicación web J-IMMSIM 2.0 será desplegada en el contenedor web Tomcat, pues se integra fácilmente con las APIs y frameworks que brinda la plataforma de desarrollo JEE antes propuesta y con el IDE Eclipse, el cual es precisamente otras de las herramientas seleccionadas a utilizar. La facilidad con que se integra el Tomcat con las tecnologías y herramientas anteriormente mencionadas, constituye grandes ventajas para el mejor funcionamiento de la aplicación. Además presenta un grupo de características descritas a continuación.

Tomcat está diseñado para ser un servidor web potente y flexible que pueda funcionar en la más amplia variedad de plataformas y entornos. Tomcat es un contenedor web sencillo de administrar y configurar, consumiendo menos recursos que un servidor de aplicaciones. Es gratis y de código abierto por lo que posee un gran número de usuarios y soporte en la comunidad mundial. Además es compatible con la mayoría de las APIs de JEE. En caso de aplicaciones con gran cantidad de usuarios, Tomcat permite lograr escalabilidad mediante clúster. Este se ha adaptado a una gran variedad de entornos a través de su diseño modular, el cual permite a los administradores de sitios web elegir qué características van a ser incluidas en el servidor seleccionando qué módulos se van a cargar, ya sea al compilar o al ejecutar el servidor. [19]

### 1.6.5 Herramientas CASE (Computer-Aided Software Engineering)

Las herramientas CASE apoyan las actividades, las técnicas y las metodologías propias de desarrollo de sistemas informáticos. Se establece una comparación entre las principales herramientas para apoyar las fases del ciclo de vida del desarrollo del J-IMMSIM 2.0.

**Rational Rose Enterprise Edition**

Rational Rose Enterprise Edition se encarga de la modelación de los diagramas correspondientes al desarrollo. Esta herramienta lleva a cabo tanto la automatización de los sistemas para la posterior generación de código (realización de los distintos diagramas y generación del código posterior), como para labores de ingeniería inversa (realización de los diagramas una vez conocido el código). [20]

Rational Rose Enterprise Edition es una forma de ayuda para la comprensión del sistema y de sus distintos componentes y lo mejor es que se puede aplicar ingeniería inversa a una multitud de códigos distintos, siempre que estén orientados a objetos. El Rational Rose Enterprise Edition es una potente herramienta CASE, pero presenta como desventajas que es un software propietario y que no soporta otras plataformas como GNU/Linux.

**Visual Paradigm**

Visual Paradigm es una herramienta profesional multiplataforma que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. Este utiliza UML como lenguaje de modelado que ayuda a una rápida construcción de aplicaciones con calidad y a un menor coste. Soporta múltiples usuarios trabajando sobre el mismo proyecto y la sincronización entre el código en Java y los modelos. Una de las principales desventajas es que utiliza muchos recursos computacionales, aunque responde a las tareas del usuario rápidamente. [21]

### Herramienta CASE propuesta

Para el desarrollo del J-IMMSIM 2.0 se definió utilizar el Visual Paradigm pues es una potente herramienta, multiplataforma y una de las más fáciles de usar para el modelado visual con UML. Facilita una excelente interoperabilidad con otras herramientas CASE y la mayoría de los principales IDE como Eclipse. Soporta los últimos estándares de Java y las notaciones UML, además proporciona generación de código e ingeniería inversa soportada para Java.

## 1.6.6 Gestor de base de datos

Para gestionar los datos del J-IMMSIM 2.0, es importante contar con un gestor de base de datos (BD), que es un tipo de software muy específico, dedicado a servir de interfaz entre la BD, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de manipulación de datos y de consulta.

Se establece una comparación entre algunos gestores de BD existentes con el propósito de escoger el que más se ajuste a las características del software, que maneje de una manera clara, sencilla y ordenada el conjunto de datos que posteriormente se convertirán en información relevante.

**Oracle**

Oracle se considera como uno de los gestores de BD más completos, destacando su soporte de transacciones, estabilidad, escalabilidad y el ser multiplataforma. [22]

Maneja BD relacionales que hacen uso de los recursos del sistema informático en cualquier hardware, para garantizar su aprovechamiento al máximo en ambientes cargados de información. Corre automáticamente en más de ochenta arquitecturas de hardware y software, sin tener la necesidad de cambiar una sola línea de código, esto se debe a que más del ochenta porciento de los códigos internos de Oracle son iguales a los establecidos en todas las plataformas de sistemas. [22]

La principal desventaja que este gestor presenta consiste en que es un software propietario, además su elevado precio (la BD más cara) hace que solo se vea en empresas muy grandes y multinacionales, presentando un alto costo de soporte técnico.

**PostgreSQL**

PostgreSQL es un gestor de bases de datos de alto nivel que ofrece muchas ventajas con respecto a otros sistemas de bases de datos. Tienen instalación ilimitada puesto que no tiene costo asociado a la licencia de software. El código fuente está disponible para todos sin costo. Permite extenderse o personalizarse con un mínimo esfuerzo. Es un software multiplataforma, se puede instalar en varios sistemas operativos. Facilita la migración de los datos procedentes de otros gestores de BD. [23]

### Gestor de BD propuesto

Se propone que se utilice el PostgreSQL para manejar los datos del J-IMMSIM 2.0, pues es multiplataforma y no tiene costo asociado a la licencia de software. Este permite que mientras algún usuario esté haciendo alguna modificación en una tabla otro realice cambios en la misma sin que se bloquee el gestor. Es un gestor que brinda gran consistencia y seguridad en los datos.

Ventajas adicionales:

- Por su arquitectura de diseño, escala muy bien al aumentar el número de CPUs y la cantidad de RAM.
- Soporta transacciones y claves ajenas (con comprobaciones de integridad referencial).
- Tiene mejor soporte para triggers y procedimientos en el servidor.

Inconvenientes:

- Consume bastantes recursos y carga más el sistema.
- Límite del tamaño de cada fila de las tablas a 8k (hasta la versión 7.1)

## Conclusiones:

- La propuesta de arquitectura es para el diseño de una aplicación web, la cual será distribuida, siendo esta la más óptima para los simuladores.
- Se organizará el sistema de acuerdo al estilo Arquitectura en Capas de la familia de Llamada y Retorno. Se aplicará el patrón arquitectónico Layers (capas).
- El rol de arquitecto estará guiado por la metodología OpenUP que utiliza UML como lenguaje de modelado
- Se utilizará JEE como plataforma de desarrollo del J-IMMSIM 2.0 y se consideró como IDE más adecuado para su implementación el Eclipse.
- Se propone que se apliquen los patrones de diseño Front Controller para la capa de Presentación, el Bussiness Delegate para la capa de Negocio y el Data Access Object para la de Acceso a Datos.
- Se utilizarán los siguientes frameworks de la plataforma JEE:
  1. Java Server Faces, para gestionar la capa de Presentación de la aplicación.
  2. Hibernate, para manejar la capa de Acceso a Datos.
  3. Spring, como gestor de los objetos de la aplicación, manejador de transacciones y como enlace entre los frameworks de la capa de Presentación y de Acceso a Datos con la capa de Negocio de la aplicación.
  4. Acegi, para gestionar la seguridad de la aplicación.
- Se utilizará el Tomcat como contenedor web.
- Se propone modelar en la herramienta CASE Visual Paradigm.
- Para el almacenamiento y gestión de los datos se utilizará PostgreSQL como gestor de BD.

# Capítulo 2: Descripción de la arquitectura

## Introducción

En este capítulo se representan las características del sistema mediante las vistas que propone la metodología OpenUP. Se describen los requisitos funcionales que responden a los casos de usos arquitectónicamente significativos y las propiedades o cualidades que el J-IMMSIM 2.0 debe tener.

## 2.1 Características del J-IMMSIM 2.0

El J-IMMSIM 2.0 está conformado por tres componentes que representan los órganos fundamentales que componen el Sistema Inmune: los Nodos Linfáticos, el Timo y la Médula Ósea (Figura 3).

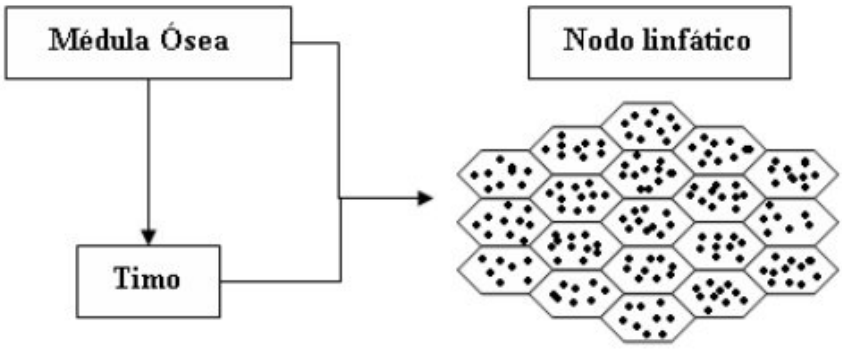


**Figura 3 Componentes del Sistema Inmune**

**Médula Ósea**

Es el componente donde se generan todas las células en un estado determinado, que intervendrán posteriormente en los procesos e interacciones que se llevan a cabo en el sistema. Algunas de ellas maduran en la propia Médula Ósea y otras lo hacen en el Timo.

**Timo**

Es el componente donde las células que no maduraron en la Médula Ósea entran en un proceso de maduración formado por una selección positiva y una negativa.

**Nodo Linfático**

Constituye el espacio de simulación. Dentro de él se encuentra un conjunto de células y moléculas que participarán en los procesos e interacciones del sistema que tienen lugar dentro del mismo.

## 2.2 Organización del J-IMMSIM 2.0

La organización del J-IMMSIM 2.0 quedará de la siguiente la manera:

Los usuarios accederán a la aplicación desde las PC (Personal Computer) clientes, donde le harán las peticiones al sistema. Las funcionalidades serán llevadas a cabo por los servidores, distribuyendo el cálculo por otras PC, para luego obtener un resultado final que será mostrado al usuario; si estos desean pueden guardar el resultado de las simulaciones en la base de datos. La Figura 4 muestra lo antes explicado.

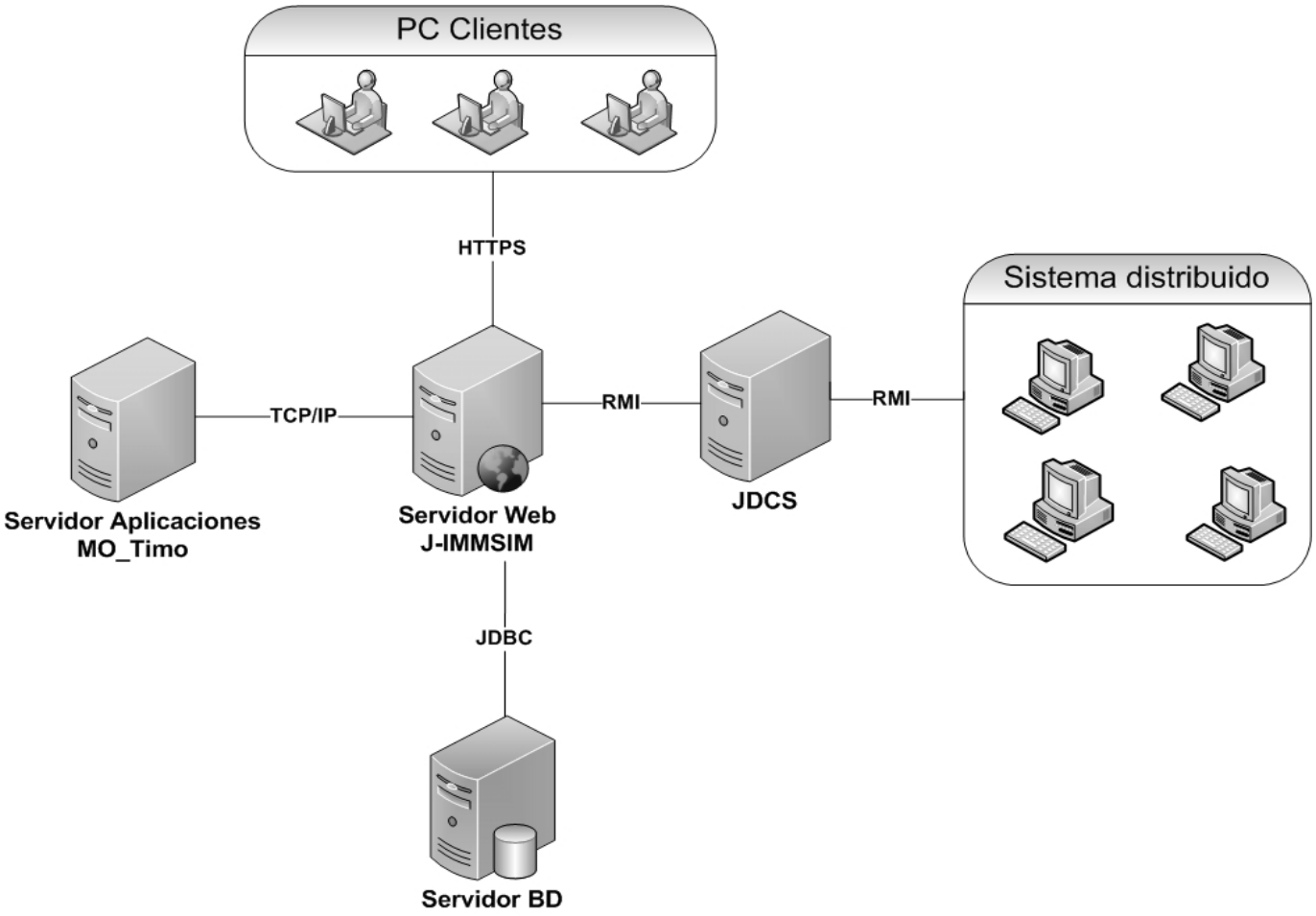


**Figura 4 Organización del J-IMMSIM 2.0**

## 2.3 Metas y restricciones

Las metas y restricciones se pueden observar a través de los requisitos que son capacidades o propiedades que tienen que ser alcanzados o poseídos por un sistema para satisfacer un contrato, estándar, u otro documento impuesto formalmente. Los requisitos se pueden clasificar en: funcionales y no funcionales.

### 2.3.1 Requisitos Funcionales

A continuación se describen los requisitos funcionales que corresponden a las capacidades o condiciones que el J-IMMSIM 2.0 debe cumplir.

R.1 Gestionar Modelo

- R.1.1 Definir nuevo modelo.
- R.1.2 Importar modelo.
- R.1.3 Exportar modelo.

R.2 Gestionar Célula

- R.2.1 Adicionar célula.
- R.2.2 Eliminar célula.
- R.2.3 Modificar célula.

R.3 Gestionar Interacción

- R.3.1 Adicionar interacción
- R.3. 2 Eliminar interacción

R.4 Gestionar Moléculas simples

- R.4.1 Adicionar moléculas simples.
- R.4.2 Eliminar moléculas simples.
- R.4.3 Modificar moléculas simples.

R.5 Gestionar Antígeno

- R.51 Adicionar antígeno.
- R.5.2 Eliminar antígeno.
- R:5.2 Modificar antígeno

R.6 Gestionar Tratamiento

- R.6.1 Adicionar tratamiento.
- R.6.2 Eliminar tratamiento.

R.7 Gestionar Anticuerpos

- R.7.1 Adicionar anticuerpos.
- R.7.2 Eliminar anticuerpos.

R.8 Gestionar Usuarios

- R.8.1 Adicionar usuario.
- R.8.2 Eliminar usuario.
- R.8.3 Modificar usuario.

R.9 Autenticar usuario

R.10 Simular el Sistema Inmune

R.11 Gestionar Receptor

- R.11.1 Adicionar receptor
- R.11.2 Eliminar receptor

R.12 Gestionar Etiqueta

- R.12.1 Adicionar etiqueta.
- R.12.2 Eliminar etiqueta.

R.13 Editar Preferencias de simulación

R.14 Gestionar Complejo Celular

- R.14.1 Adicionar complejo celular.
- R.14.2 Eliminar complejo celular.
- R.14.3 Modifica complejo celular

R.15 Gestionar Inmuno Complejo

- R.15.1 Adicionar inmuno complejo.
- R.15.2 Eliminar inmuno complejo.
- R.15.3 Modifica inmuno complejo

### 2.3.2 Requisitos no funcionales

A continuación se describen los requisitos no funcionales que son propiedades o cualidades que el J-IMMSIM 2.0 debe tener, clasificados en varias categorías.

**Software**

- Sistema operativo Windows98 o superior, GNU/Linux y sus distribuciones.
- Máquina Virtual de Java versión 1.5 o superior.
- MATLAB versión 7.0 como asistente matemático.
- JDCS (Java Distributed Computing System) para la distribución del cálculo.
- Tomcat 6.x para el servidor web.
- Navegador web Mozilla Firefox, Internet Explorer, Maxton, entre otros.

**Hardware**

- Computadoras Personales con al menos 256 MB de memoria RAM y procesador al menos de 1.0 GHz.
- Servidor de base de datos con más de 160 GB de espacio en el disco duro, 1 GB de memoria RAM y procesador al menos de 2.0GHz.
- Servidor web (J-IMMSIM) con mínimo de memoria RAM de 2 GB, procesador tecnología Pentium 4 (P4) de 2.0 GHz y disco duro de al menos 40 GB.
- Servidor de objetos (JDCS) con al menos 2 GB de memoria RAM, procesador P4 de 2.0GHz y disco duro de al menos 40 GB.
- Servidor de aplicaciones (MO_Timo) con al menos 2 GB de memoria RAM, procesador de 2.0GHz y disco duro de al menos 40 GB.

**Restricciones en el diseño y la implementación**

- Java como lenguaje de programación.
- Eclipse como herramienta de desarrollo.
- Visual Paradigm 3.0 como herramienta CASE para el modelado de los artefactos.
- PostgreSQL como gestor de base de datos.
- Tomcat 6.x como contenedor web.
- Librería JFreeChart para la graficación

**Requerimientos de apariencia o interfaz externa**

La apariencia será amigable y fácil de usar para el usuario, no dependiendo de alguna preparación, ni conocimientos profundos informáticos. Los botones y los campos van a estar organizados por tema para mayor comprensión.

**Requerimientos de Seguridad**

- Confidencialidad: La información manejada por el sistema estará protegida de acceso no autorizado. La aplicación contará con un sistema de autenticación de usuario otorgándole los permisos a cada uno según el nivel de acceso al sistema que tengan permitido, evitando que personal ajeno acceda a la información.
- Integridad: La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes. Se contará con un potente gestor de base de datos el cual proporciona la máxima protección a los datos almacenados. Se garantizarán las comunicaciones seguras entre los clientes y el servidor, encriptando el canal por donde fluye la información usando llaves negociadas, algoritmos y protocolos.
- Disponibilidad: A los usuarios autorizados se les garantiza el acceso a la información que ellos crearon y almacenaron en la base de datos con los procedimientos almacenados.

**Requerimientos de Usabilidad**

- Facilidad de aprendizaje: El software será diseñado para que sea utilizado por aquellos usuarios que tengan conocimientos biológicos, debido a que la información que se gestiona requiere de estos, aunque no limita que cualquier otro usuario pueda utilizarlo si tiene los permisos para acceder al sistema. El usuario que utilice el software puede aprender a usarlo en un corto período de tiempo, ya que desde el punto de vista técnico este no es complicado.
- Comprensibilidad: El software presentará interfaces con un buen diseño, pues para desarrollar las mismas se utilizará el framework JSF, que brinda sofisticados componentes de diseño de interfaz.

**Soporte**

- Mantenibilidad: El software se puede someter a las reparaciones en caso de necesitarla, las cuales se llevarán a cabo en el servidor web, lo que reduce el costo de soporte.
- Instalación: Una vez terminada la aplicación se instalará en el CIM para realizar pruebas.
- Nivel de Apoyo: El software presentará un manual de usuario con un nivel alto de explicación detallada. Si es necesario se dará una capacitación a los futuros usuarios para mayor comprensión.
- Extensibilidad: El sistema permitirá agregar nuevos CPUs para ampliar el espacio de almacenamiento de los datos.

**Rendimiento**

Para que el sistema sea capaz de responder lo más rápido posible a las peticiones se hace uso de la plataforma de cálculo distribuido y así este no se tornará lento, posibilitando agilidad en la obtención de los resultados.

**Portabilidad**

El sistema debe funcionar en varios sistemas operativos ya que este es multiplataforma porque está implementado en Java, para esto se requiere la Máquina Virtual de Java 1.5 o superior.

La mayoría de estos requisitos antes mencionados serán representados por las vistas arquitectónicas que se muestran a continuación.

## 2.4 Vistas arquitectónicas

Las vistas arquitectónicas son vistas abstractas, aportando el más alto nivel de comprensión. El objetivo principal de las mismas es describir los aspectos fundamentales del sistema, proporcionándole un lenguaje común a los desarrolladores muy importante a la hora de tomar futuras decisiones. Para describir la arquitectura del J-IMMSIM 2.0 se modelaron las vistas propuestas por la metodología OpenUP.

### 2.4.1 Vista de Casos de Uso

Esta vista representa un subconjunto del artefacto diagrama de casos de uso y lista los casos de usos o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema.

Los casos de uso significativos desde el punto de vista arquitectónico, son aquellos que hacen uso de una gran cantidad de componentes del sistema. [24]

Del diagrama de casos de uso del sistema (Figura 10), se identifican los casos de uso arquitectónicamente significativos aplicando la teoría a la práctica. En la tabla 1, se muestra la relación de los casos de uso/requisitos funcionales con los componentes del J-IMMSIM 2.0, evaluando la significancia para la arquitectura de cada caso de uso.

| **Casos de Uso** | **Significancia en la Arquitectura** | | | | | |
|---|---|---|---|---|---|---|
| | **Formulario** | **Server Pages** | **Entidades** | **Tablas** | **Client Pages** | **Total** |
| R.1 Gestionar Modelo | 4 | 3 | 1 | 1 | 4 | 13 |
| R.2 Gestionar Célula | 3 | 1 | 1 | 1 | 3 | 9 |
| R.3 Gestionar Interacción | 3 | 1 | 1 | 1 | 3 | 9 |
| R.4 Gestionar Moléculas simples | 3 | 1 | 1 | 1 | 3 | 9 |
| R.5 Gestionar Antígeno | 3 | 1 | 1 | 1 | 3 | 9 |
| R.6 Gestionar Tratamiento | 2 | 1 | 1 | 1 | 2 | 7 |
| R.7 Gestionar Anticuerpos | 2 | 1 | 1 | 1 | 2 | 7 |
| R.10 Simular el Sistema Inmune | 1 | 2 | 5 | 5 | 1 | 14 |
| R.11 Gestionar Receptor | 2 | 1 | 1 | 1 | 2 | 7 |
| R.12 Gestionar Etiqueta | 2 | 1 | 1 | 1 | 2 | 7 |
| R.13 Editar Preferencias | 1 | 1 | 1 | 1 | 1 | 5 |
| R.14 Gestionar Complejo Celular | 3 | 1 | 1 | 1 | 3 | 9 |
| R.15 Gestionar Inmuno Complejo | 3 | 1 | 1 | 1 | 3 | 9 |
| **TOTAL** | | | | | | **114** |

**Tabla 1 Evaluación de los casos de uso**

Para la selección de los casos de uso arquitectónicamente significativos se calculó la media aritmética y se escogieron aquellos casos de uso mayores que esta.

**Media Aritmética**: La media aritmética o promedio es el valor resultante que se obtiene al dividir la sumatoria de un conjunto de datos sobre el número total de datos. Solo es aplicable para el tratamiento de datos cuantitativos. [25]

**Cálculo:**

Media=TOTAL /cantidad de casos de usos.

Media=114/13.

Media=9

Según la media, aquellos casos de uso > 9 cantidad de componentes, son arquitectónicamente significativos.

La Figura 5 muestra los casos de usos arquitectónicamente significativos después de haber aplicado el método.

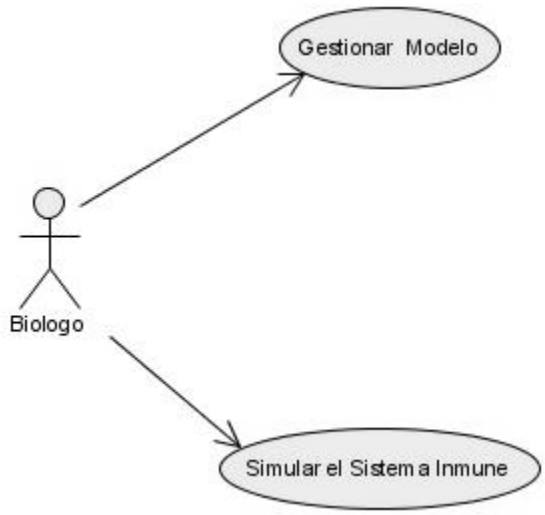


**Figura 5 Vista de Casos de Uso**

**Breve descripción de los casos de uso arquitectónicamente significativos**

| | |
|---|---|
| Caso de Uso | Simular el Sistema Inmune |
| Actores | Biólogo |
| Propósito | Simular las reacciones del Sistema Inmune bajo determinadas condiciones. |
| Descripción | El caso de uso se inicia cuando el biólogo desea simular una respuesta inmunitaria y selecciona la opción simular. |
| ¿Por qué arquitectónicamente significativo? | El objetivo principal del sistema es dar a conocer la dinámica (cantidad) de los entes biológicos (células, moléculas simples y sus complejos, anticuerpos y antígenos) después de interactuar. |

**Tabla 2 Breve descripción: Caso de uso Simular**

| | |
|---|---|
| Caso de Uso | Gestionar Modelo |
| Actores | Biólogo |
| Propósito | Crear, exportar e importar el modelo con los entes biológicos introducidas por el biólogo. |
| Descripción | El caso de uso se inicia cuando el biólogo selecciona la opción de crear un nuevo modelo, importarlo o exportarlo. |
| ¿Por qué arquitectónicamente significativo? | Este caso de uso es importante porque es el que gestiona los datos de los entes biológicos para su futura simulación. Sin él las simulaciones no tendrían lugar. |

**Tabla 3 Breve descripción: Caso de uso Gestionar Modelo**

### 2.4.2 Vista Lógica

La vista lógica representa los elementos del diseño más importantes para la arquitectura del sistema. Describe las clases agrupas por sus funcionalidades y ordenadas según el patrón arquitectónico Capas, la misma se estructura en paquetes y subsistemas para mayor comprensión y organización.

Los paquetes más generales que responden al patrón arquitectónico Capas son: Presentación (contiene las clases interfaces), Negocio (contiene las clases controladoras agrupadas en paquetes por sus funcionalidades) y Acceso a Datos (contiene aquellas clases de acceso a los datos y las clases persistentes de la base de datos) (Figura 6).

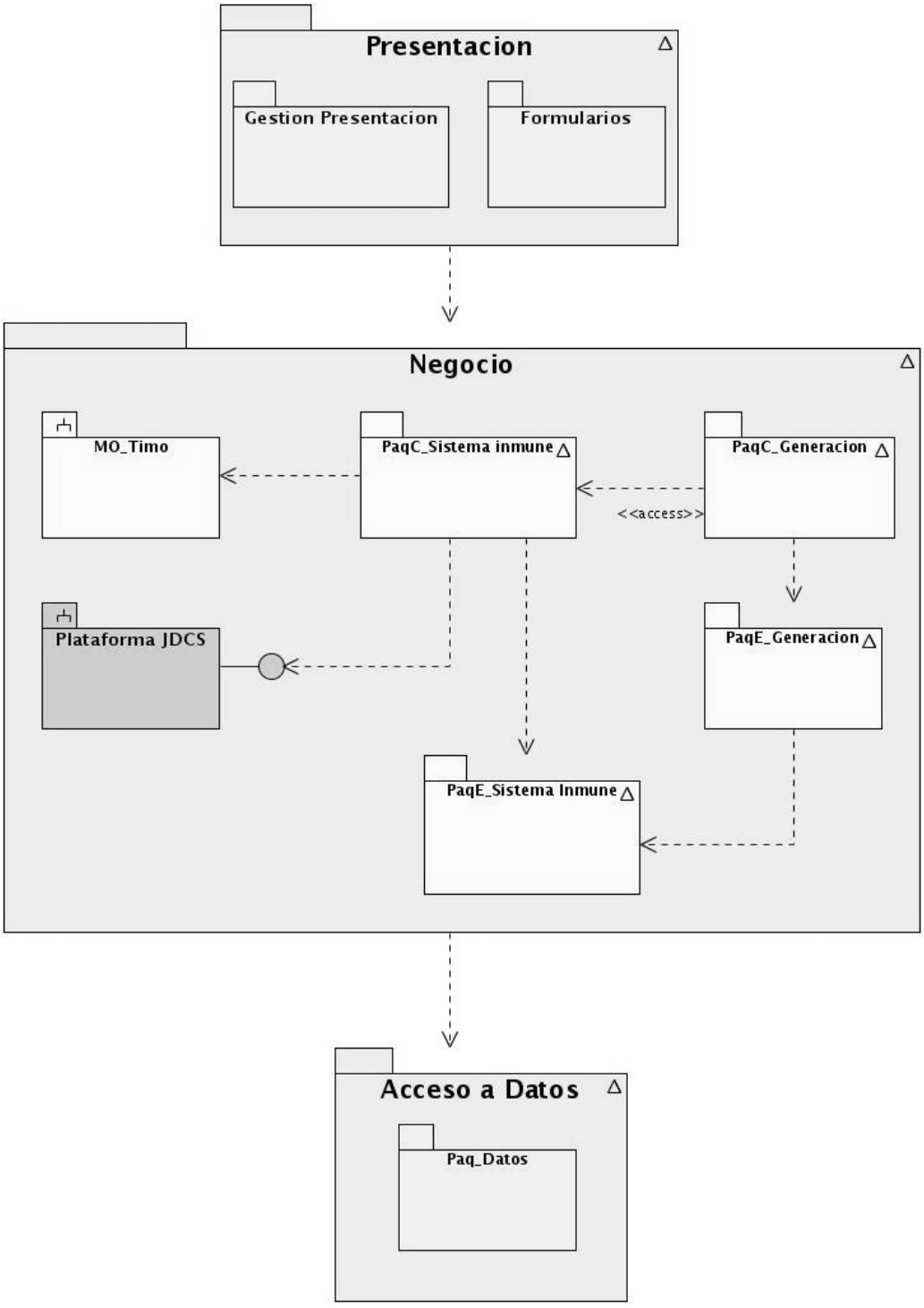


**Figura 6 Vista Lógica**

Descripción de los paquetes de la vista lógica (Figura 6).

➢ **Presentación**

- **Paquete Formularios**: Contiene las clases que generan las páginas interfaces de usuario.
- **Paquete Gestión Presentación**: Contiene las clases con las funcionalidades que gestionan las peticiones de las páginas interfaces de usuario.

➢ **Negocio**

- **Paquete PaqC_SistemaInmune**: Contiene las clases controladoras encargadas de la simulación que incluyen las interacciones entre los entes biológicos, el proceso de difusión (que consiste en balancear la cantidad de entes biológicos en los nodos linfáticos), así como la obtención del resultado final de la simulación.
- **Paquete PaqC_Generación:** Contiene las clases controladoras encargadas de la gestión del modelo, es decir, los entes biológicos que participan en el proceso de simulación que abarca la creación, eliminación y modificación de los mismos.
- **Paquete PaqE_Generación**: Contiene las clases que se encargan de la generación del modelo, es decir, las clases .java correspondientes a cada ente biológico.
- **PaqE_SistemaInmune:** Contiene las clases entidades que corresponden a los entes biológicos.
- **Subsistema Plataforma JDCS:** Es un software independiente que se encarga de la distribución del cálculo. Gestiona las peticiones solicitadas por el paquete PaqC_SistemaInmune y las envía a las diferentes PCs designadas al cálculo.
- **Subsistema MO_Timo**: Es un subsistema que calcula y devuelve la cantidad de células a incorporar al SI, simulando la Médula Ósea y el Timo biológico.

➢ **Acceso a datos**

- **Paquete Datos:** Contiene las clases que se encargan de las funcionalidades relacionadas con el acceso a los datos, mayormente a través de procedimientos almacenados, así como las interfaces correspondientes a los métodos que brinda DAO.

### 2.4.3 Vista de Despliegue

La vista de despliegue suministra una base para la comprensión de la distribución física de un sistema (computadoras, dispositivos) y sus conexiones. Es un grafo de nodos unidos por conexiones de comunicación. Se muestra a continuación algunos conceptos fundamentales para la comprensión de esta vista.

**Servidores**

Un **servidor** es un tipo de software que realiza ciertas tareas en nombre de los usuarios. El término servidor también se utiliza para referirse al ordenador físico en el cual funciona ese software, una computadora cuyo propósito es proveer datos de modo que otras computadoras puedan utilizar los mismos. [26]

Los **servidores web** sirven contenido estático a un navegador, carga un archivo y lo sirve a través de la red al navegador de un usuario. Este intercambio es mediado por el navegador comunicándose mediante un protocolo. Se pueden utilizar varias tecnologías en el servidor para aumentar su potencia, más allá de su capacidad de entregar páginas HTML. [26]

Los **servidores de aplicación** designados a veces como un tipo de middleware (software que conecta dos aplicaciones), ocupan una gran parte del territorio entre los servidores de BD y el usuario, que muy a menudo los conectan. [26]

El **servidor de BD** es donde se almacenan grandes colecciones de datos llevando a cabo todo el procesamiento, donde el cliente envía las consultas y el servidor la interpreta devolviendo una respuesta. [26]

Los **Servidores de Objetos** proporcionan objetos distribuidos que pueden acceder a otras computadoras de una red distante, los cuales son tratados por el programador como si estuvieran en su computadora local. [1]

**Protocolos de comunicación**

Un protocolo de comunicación es un conjunto de reglas establecidas entre dos dispositivos para permitir la comunicación entre ambos.

**HTTPS Hypertext Transfer Protocol Secure** (*Protocolo seguro de transferencia de hipertexto*): Es un protocolo de red basado en el protocolo HTTP, destinado a la transferencia segura de datos de hipertexto.

El sistema HTTPS utiliza las SSL para crear un canal cifrado (cuyo nivel de cifrado depende del servidor remoto y del navegador utilizado por el cliente) más apropiado para el tráfico de información sensible que el protocolo HTTP. [27]

**TCP/IP:** Es la base del Internet que sirve para enlazar computadoras.

El protocolo IP gestiona un sistema de direcciones que se asigna a cada computadora de la red y divide cada mensaje en bloques pequeños (paquetes IP). Coloca una cabecera a estos paquetes en la que van las direcciones del emisor, del receptor y la identificación y el tamaño del paquete. [27]

El protocolo TCP ocupa la capa superior al protocolo IP y se ocupa de gestionar los envíos entre computadoras para que sean seguros, es decir, que no se pierda información. Para ello añade a las cabeceras de los paquetes IP una serie de datos como son el tamaño del paquete, el número de orden del paquete, el número total de paquetes.

**JDBC (Java Database Connectivity):** Se utiliza para conectar un programa del usuario con una base de datos, sin importar qué software de administración o manejo de base de datos se utilice para controlarlo. Permite la ejecución de operaciones sobre BD desde Java independientemente del sistema operativo donde se ejecute o de la BD a la cual se accede. [15]

**RMI (Remote Method Invocation)**: La invocación de métodos remotos permite que un objeto que se ejecuta en una computadora puede invocar métodos de un objeto que se encuentra en ejecución bajo el control de otra, la comunicación se produce como si todo estuviese en una computadora local. [15]

**Nodos:**

Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento.

En la siguiente figura se muestra la distribución física del J-IMMMSIM 2.0 y a continuación de esta la descripción de los nodos con los requerimientos computaciones mínimos que debe tener cada uno:

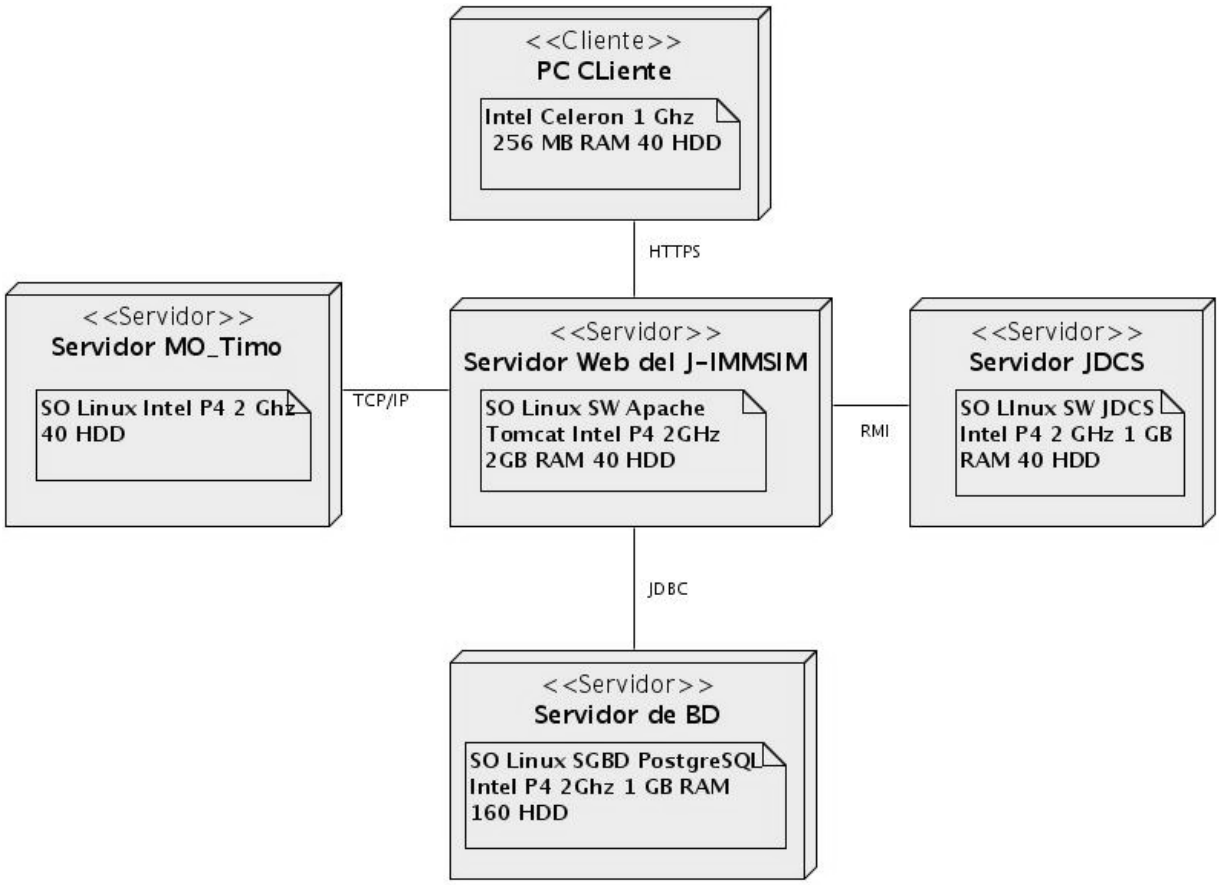


**Figura 7 Vista de despliegue**

**Nodo PC Cliente:** Son aquellas computadoras donde los usuarios accederán a la aplicación y se realizarán las peticiones al servidor mediante el protocolo HTTPS.

**Nodo Servidor del J-IMMSIM:** Es un servidor web encargado de un grupo de funcionalidades como la gestión de los usuarios y sus peticiones al sistema, la gestión de los entes biológicos, así como sus interacciones. Realiza la difusión por los nodos linfáticos para una próxima iteración. Calcula las cantidades parciales de los entes biológicos de cada nodo para dar el resultado final de la simulación.

**Nodo Servidor MO_Timo:** Es un servidor de aplicaciones que se encarga de la generación de las células a petición del servidor del J-IMMSIM, mediante el cálculo por las ecuaciones estocásticas, cuyos resultados implican el balance de las células por cada nodo linfático, este servidor pone en funcionamiento el asistente de cálculo MATLAB cada vez que sea necesario.

**Nodo Servidor de BD:** Es donde se encuentra la base de datos para gestionar la información de las simulaciones a petición de los usuarios. Utiliza JDBC como protocolo de comunicación entre este y el servidor del J-IMMSIM.

**Nodo Servidor JDCS:** Representa un servidor de objetos que recibe las peticiones del cliente, las divide en estaciones de trabajo y se las envía a las PCs clientes por RMI. Se encarga de la distribución del cálculo por las PCs que representan cada nodo linfático, permitiendo una mayor agilidad en la obtención de los resultados.

### 2.4.4 Diagrama de componentes

Un diagrama de componentes muestra un conjunto de componentes y sus relaciones. Los diagramas de componentes se utilizan para describir la vista de implementación estática de un sistema. Estos se relacionan con los diagramas de clases, ya que un componente normalmente se corresponde con una o más clases, interfaces o colaboraciones.

La Figura 8 muestra el diagrama de componentes del J-IMMSIM 2.0 y seguido la descripción de cada uno de estos.

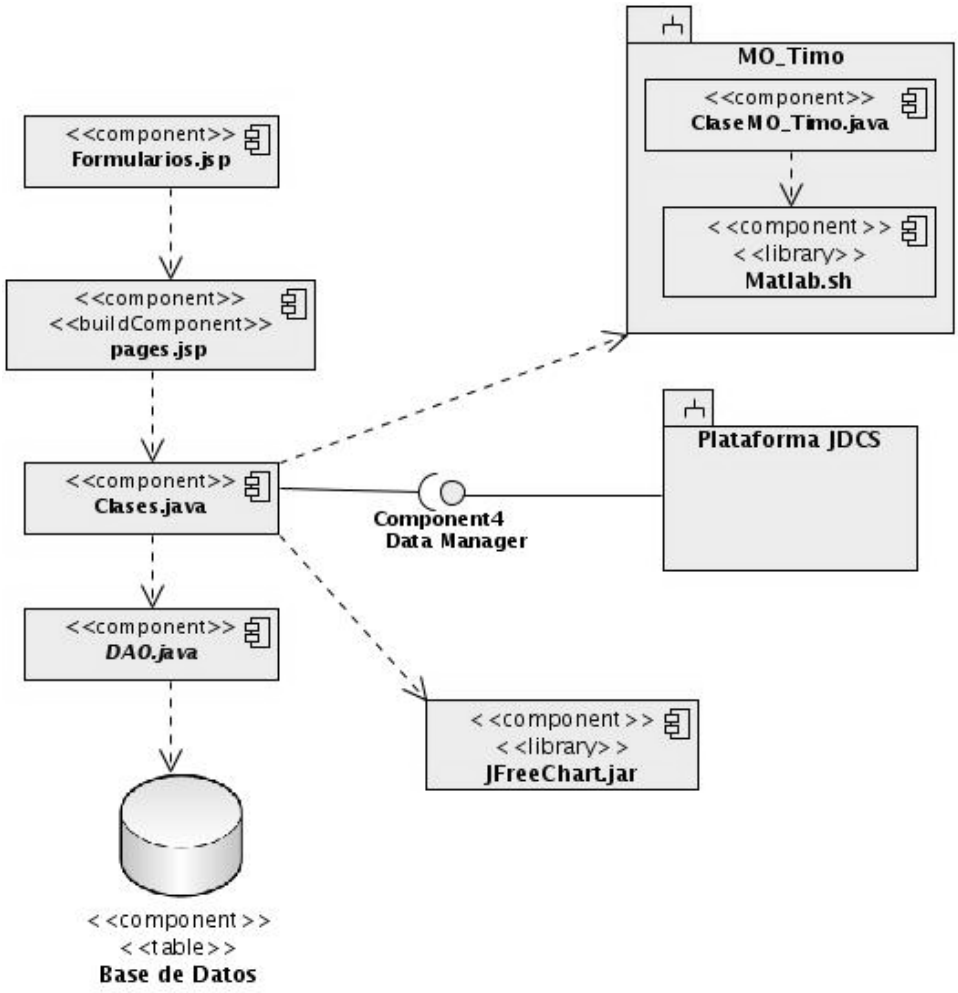


**Figura 8 Diagrama de componentes**

Los componentes que muestra la Figura 8 se describen a continuación:

- **Componentes pages.jsp**: Son las páginas interfaces donde los usuarios introducen la información.
- **Componentes Formularios.jsp**: Son las páginas que manejan la lógica del procesamiento de la capa de Presentación.
- **Clases.java**: Son las principales clases que implementan las funcionalidades fundamentales del simulador.
- **Componente Daos.java**: Son las clases que se encargan de todo lo concerniente a las funcionalidades para la manipulación de la información.
- **Componente Clases MO_Timo.java**: Representa un conjunto de clases encargadas de generar las células a petición de las clases en el paquete de Negocio.
- **Subsistema Plataforma JDCS**: Contiene los componentes de la plataforma JDCS que distribuye el cálculo a las PCs clientes mediante el protocolo de comunicación RMI.
- **Componente MATLAB**: Representa la herramienta de cálculo que utiliza el subsistema MO_Timo para calcular las ecuaciones diferenciales estocásticas.
- **Componente Base Datos**: Representa la base de datos física.
- **JFreeChart.jar**: Es una librería utilizada para la graficación de los resultados de las simulaciones.

### 2.4.5 Vista de Implementación

La vista de implementación muestra las relaciones físicas entre los componentes hardware y software en el sistema final, es decir, la configuración de los elementos de procesamiento en tiempo de ejecución y los componentes software (procesos y objetos que se ejecutan en ellos). Esta describe la descomposición del software en capas y subsistemas de implementación.

En la Figura 9 se representa la unión del diagrama de despliegue y el diagrama de componentes que conforman la vista de implementación del J-IMMSIM 2.0, donde cada componente está distribuido por los diferentes nodos físicos del despliegue.

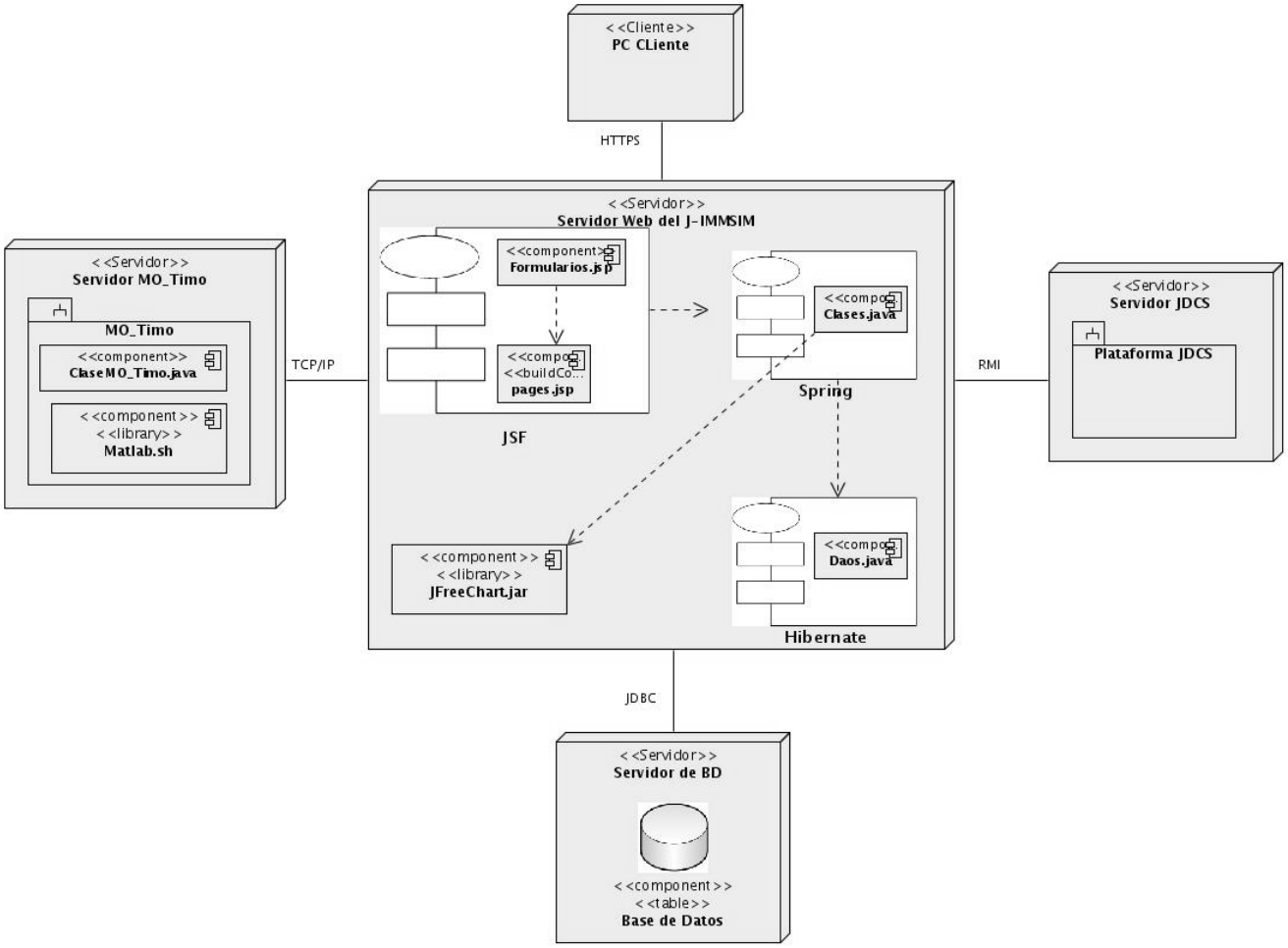


**Figura 9 Vista de implementación**

## Conclusiones

- Se identificaron los casos de uso, clases del diseño y componentes arquitectónicamente significativos para el sistema.
- Se diseñaron las vistas del sistema y se describieron los componentes que las integran.

# Capítulo 3: Evaluación de la arquitectura

## Introducción

En este capítulo se abordan los diferentes métodos, técnicas y atributos de calidad que se utilizan para evaluar la arquitectura. Se argumenta la importancia de evaluarla y se explican los resultados obtenidos.

## 3.1 Evaluación de la AS

La evaluación de una AS es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. [28]

El primer paso para la evaluación de una arquitectura es conocer qué es lo que se quiere evaluar. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. [29]

La evaluación tiene como objetivo saber si la arquitectura puede habilitar los requerimientos no funcionales y los atributos de calidad para asegurar que el sistema cumple con las necesidades de los clientes, por esto es muy importante evaluarla.

## 3.2 Importancia de evaluar la arquitectura

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

Una evaluación de la arquitectura no te dice si es buena o mala, solo identifica dónde está el riesgo, las fortalezas y debilidades. Después se pueden tomar algunas decisiones definiendo si puede continuar el proyecto dada las debilidades identificadas, si hay que reforzar la arquitectura o si hay que comenzarla de nuevo. Es importante saber en qué etapa evaluar la arquitectura para poder determinar con mayor exactitud los posibles riegos. [28]

## 3.3 Etapas en las que se evalúa la arquitectura

Una buena regla para decidir cuándo hay que realizar una evaluación de la arquitectura es cuando el equipo de desarrollo comience a tomar decisiones que dependan de la misma y el costo de deshacer dichas decisiones sea mayor al costo de realizar la evaluación.

Es posible realizar la evaluación de la arquitectura en cualquier momento, pero existen dos variantes que agrupan distintas etapas: temprano y tarde. [28]

Evaluación temprana: Establece que no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo. Bosch propone que es posible tomar decisiones sobre la arquitectura a cualquier nivel, puesto que se pueden aparecer distintos tipos de cambios arquitectónicos, producto de una evaluación en función de los atributos de calidad esperados. [28]

Evaluación tardía: Consiste en realizar la evaluación de la arquitectura cuando esta se encuentra establecida y la implementación completada. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado. Se considera muy útil la evaluación del sistema en este punto, puesto que puede observarse el cumplimiento de los atributos de calidad (que se muestran a continuación) asociados al sistema y cómo será su comportamiento general. [28]

## 3.4 Atributos de calidad

La AS se rige por cualidades para su evaluación, estos son los atributos de calidad, aunque a veces son demasiados imprecisos. Estos se clasifican en dos tipos: observables en vía ejecución y no observables en vía ejecución.

Observables en vía ejecución: Aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución.

No observables en vía ejecución: Aquellos atributos que se establecen durante el desarrollo del sistema.

Como el J-IMMSIM 2.0 no se ha implementado, para la evaluación de su arquitectura solamente se pueden medir los atributos de calidad no observables en vía de ejecución.

**Atributos de calidad no observables en vía de ejecución**

- Configurabilidad: Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
- Integrabilidad: Es la medida en que trabajan correctamente los componentes del sistema que fueron desarrollados separadamente al ser integrados.
- Integridad: Es la ausencia de alteraciones inapropiadas de la información.

- Interoperabilidad*:* Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de integrabilidad.
- Modificabilidad: Es la habilidad de realizar cambios futuros al sistema.
- Mantenibilidad: Es la capacidad de someter a un sistema a reparaciones y evolución. Capacidad de modificar el sistema de manera rápida y a bajo costo.
- Portabilidad: Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos.
- Reusabilidad: Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones.
- Escalabilidad: Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental.
- Capacidad de prueba: Es la medida de la facilidad con la que el software, al ser sometido a una serie de pruebas, puede demostrar sus fallas. Es la probabilidad de que, asumiendo que tiene al menos una falla, el software fallará en su próxima ejecución de prueba.

[28]

Para lograr la evaluación de la AS de un sistema es necesario medir los atributos de calidad, para esto existen diversas técnicas por lo que resulta importante estudiar las opciones que estas ofrecen para llevar a cabo esta tarea.

## 3.5 Técnicas de evaluación

Las técnicas utilizadas para la evaluación de atributos de calidad demandan grandes esfuerzos por parte del ingeniero de software para crear especificaciones y predicciones. Estas técnicas requieren información del sistema a desarrollar que no está disponible durante el diseño arquitectónico, sino al principio del diseño detallado del sistema.

Las técnicas existentes en la actualidad para evaluar arquitecturas permiten hacer una evaluación cuantitativa sobre los atributos de calidad a nivel arquitectónico, pero se tienen pocos medios para predecir el máximo (o mínimo) teórico para las arquitecturas de software. Sin embargo, debido al costo de realizar este tipo de evaluación, en muchos casos los arquitectos de software evalúan cualitativamente, para decidir entre las alternativas de diseño existen diferentes técnicas de evaluación: basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia. [28]

### 3.5.1 Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con este.

Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. 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.

Entre las ventajas de su uso están:

1. Son simples de crear y entender.
2. Son poco costosos y no requieren mucho entrenamiento.
3. Son efectivos.

Actualmente las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree y Profile.

**Utility Tree**

Utility Tree es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. [29]

**Perfiles (Profiles)**

Un perfil (profile) es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de estos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Los perfiles tienen asociados dos formas de especificación: perfiles completos y perfiles seleccionados. [28]

### 3.5.2 Evaluación basada en simulación

Se establece que la evaluación basada en simulación utiliza una implementación de alto nivel de la AS. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad. [28]

### 3.5.3 Evaluación basada en modelos matemáticos

Se establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro. [28]

### 3.5.4 Evaluación basada en experiencia

Se establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño. Aunque todas estas experiencias se basan en factores subjetivos como la intuición y la experiencia, la mayoría puede ser justificada por una línea lógica de razonamiento y pueden ser la base de otros enfoques de evaluación [28].

Existen dos tipos de evaluación basada en experiencia, la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño y la realizada por equipos externos de evaluación de arquitecturas.

#### Técnica de evaluación propuesta

De las técnicas antes estudiadas se utilizará la basada en escenario, pues los escenarios permiten de una manera más fácil concretar y entender los atributos de calidad. Además la técnica es poco costosa, no requiere de mucho entendimiento pues es fácil de aplicar y efectiva. Se usará como instrumento de evaluación perfiles pues permite detallar con más precisión el requerimiento para un atributo de calidad

Los métodos de evaluación sirven para identificar los atributos de calidad con los que cumple el software, los cuales serán evaluados con las técnicas. Por esta razón, resulta conveniente estudiar los métodos de evaluación de AS definidos hasta el momento.

## 3.6 Métodos de evaluación

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. [29]

A continuación se analizan algunos de los métodos:

### 3.6.1 Software Architecture Analysis Method (SAAM)

El Método de Análisis de Arquitecturas de Software (Software Architecture Analysis Method, SAAM) es el primero que fue ampliamente promulgado y documentado. El método fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad. [29]

El método de evaluación SAAM se enfoca en la enumeración de un conjunto de escenarios que representan los cambios probables a los que estará sometido el sistema en el futuro. Como entrada principal, es necesaria alguna forma de descripción de la arquitectura a ser evaluada. [29]

**Metodología SAAM:**

El método de evaluación SAAM comprende 6 pasos:

Paso 1. Desarrollo de escenarios.

Paso 2.Descripción de la arquitectura.

Paso 3.Clasificación y asignación de prioridad de los escenarios.

Paso 4. Evaluación individual de los escenarios indirectos.

Paso 5. Evaluación de la interacción entre escenarios.

Paso 6.Creación de la evaluación global.

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad.

Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

### 3.6.2 Architecture Trade-off Analysis Method (ATAM)

El Método de Análisis de Acuerdos de Arquitectura (Architecture Trade-off Analysis Method, ATAM) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos. [29]

El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados, estos elementos representan los medios empleados por la arquitectura para alcanzar los atributos de calidad, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas, entre otros [29].

**Metodología ATAM:**

El método de evaluación ATAM comprende 9 pasos, agrupados en 4 fases:

**Fase 1: Presentación**

1. Presentación del ATAM.
2. Presentación de las metas del negocio.
3. Presentación de la arquitectura.

**Fase 2: Investigación y análisis**

4. Identificación de los enfoques arquitectónicos.
5. Generación del Utility Tree.
6. Análisis de los enfoques arquitectónicos.

**Fase 3: Pruebas**

7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Análisis de los enfoques arquitectónicos.

**Fase 4: Reporte**

9. Presentación de los resultados

### 3.6.3 Active Reviews for Intermediate Designs (ARID)

El método de Análisis de Diseños Intermedios es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. En ocasiones, es necesario saber si un diseño propuesto es conveniente, desde el punto de vista de otras partes de la arquitectura. Según los autores, ARID es un híbrido entre Active Design Review (ADR) y Architecture Trade-Off Method (ATAM), descrito anteriormente. ADR es utilizado para la evaluación de diseños detallados de 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. [29]

Kazman propone que tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares, dado que ninguno por sí solo es conveniente. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura. [29]

De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una AS.

**Metodología ARID:**

El método de evaluación ARID comprende 9 pasos, agrupados en 2 fases:

**Fase 1: Actividades Previas**

1. Identificación de los encargados de la revisión.
2. Preparar el informe de diseño.
3. Preparar los escenarios base.
4. Preparar los materiales.

**Fase 2: Revisión**

5. Presentación del ARID.
6. Presentación del diseño.

7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen.

### Método propuesto

Un método de evaluación no es mejor que otro, sino que evalúa mejor, en ciertas condiciones, un atributo de calidad dado. Por lo que se concluye que en dependencia de las condiciones y lo que se desea evaluar, será el método de evaluación empleado.

El desarrollo del J-IMMSIM 2.0 se encuentra en una etapa prematura, por lo que el método que se utilizará es el ARID, el cual es más conveniente para realizar la evaluación de diseños parciales en etapas tempranas del desarrollo. Este método se basa en escenarios, técnica propuesta anteriormente. Este método es fácil de usar, aplicarlo es poco costoso y de gran beneficio. Este evalúa mejor la factibilidad de la arquitectura.

Con las técnicas, instrumentos, atributos de calidad y métodos seleccionados se evaluará el diseño de la AS del J-IMMSIM 2.0.

## 3.7 Evaluando la arquitectura del J-IMMSIM 2.0

Si las decisiones arquitectónicas determinan los atributos de calidad del sistema, entonces es posible evaluar dichas decisiones con respecto a su impacto sobre estos atributos.

Muchos de estos atributos como los que ya se vieron, no pueden ser medidos directamente. Por esto, un análisis de la arquitectura debe ser ejecutado para determinar cuán satisfactoria es para el propósito del sistema.

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

| **Atributo de calidad** | **Perfil** | **Escenario** |
|---|---|---|
| Configurabilidad | Configuración | Gestionar Usuario |
| **Relación atributo-escenario:** | | |
| La configurabilidad se refiere a la posibilidad que tiene un usuario a realizar cambios en el sistema. Mediante el framework Acegi se gestiona la autenticación de los usuarios, lleva a cabo el control de los permisos, garantizando un correcto acceso a los recursos.<br>El administrador el es único con permisos para realizarle cambios a las tabla Usuarios, siendo capaz de agregar, modificar o eliminar los mismos. Por otra parte, ningún usuario que no desempeñe este rol, puede realizarle otro cambio al sistema. | | |

**Tabla 4 Evaluación en el escenario Gestionar Usuario**

| **Atributo de calidad** | **Perfil** | **Escenario** |
|---|---|---|
| Mantenibilidad | Mantenimiento | Despliegue de la capa de Negocio |
| **Relación atributo-escenario:** | | |
| El mantenimiento del sistema es menos costoso siendo este una aplicación web, porque los cambios o reparaciones se realizarían solo en el servidor web y no en cada una de las PC clientes como sería en caso de tener un servidor de aplicaciones. | | |

**Tabla 5 Evaluación en el escenario Capa de Negocio**

| **Atributo de calidad** | **Perfil** | **Escenario** |
|---|---|---|
| Escalabilidad | Ampliación | Fuente de Datos (BD) |
| **Relación atributo-escenario:** | | |
| Mediante el gestor de BD PostgreSQL se pueden agregar nuevas CPUs, formando un clúster en caso que no se pueda almacenar más información porque no se cuenta con más capacidad de almacenamiento. Esto permite ampliar el espacio requerido para guardar información. . | | |

**Tabla 6 Evaluación en el escenario BD**

| Atributo de calidad | Perfil | Escenario |
|---|---|---|
| Mantenibilidad | Mantenimiento | Acceso a Datos. |
| **Relación atributo-escenario:** | | |
| Con la utilización del framework Hibernate que implementa el patrón DAO, permite crear una capa de abstracción de datos, asegurando la migración del gestor de BD en el momento que sea requerido, sin alterar la capa de Negocio. Esto posibilita que la arquitectura sea menos rígida en su diseño de datos. | | |

**Tabla 7 Evaluación en el escenario Capa de Acceso a Datos**

| Atributo de calidad | Perfil | Escenario |
|---|---|---|
| Portabilidad | Portabilidad | Capa Negocio |
| **Relación atributo-escenario:** | | |
| El J-IMMSIM 2.0 está implementado en Java, por lo que es multiplataforma, debido a que la Máquina Virtual de Java se pude ejecutar en varios sistemas operativos. Esto constituye una ventaja porque no restringe el despliegue del sistema. | | |

**Tabla 8 Evaluación del escenario Capa de Presentación**

| **Atributo de calidad** | **Perfil** | **Escenario** |
|---|---|---|
| Integridad | Integridad | Capa de Acceso a Datos |
| **Relación atributo-escenario** | | |
| Se utiliza el framework Acegi que configura la protección del canal por donde fluyen los datos, mediante el protocolo HTTPS se cifra dicho canal, asegurando que la información no sea alterada.<br>Se utiliza como gestor de base de datos PostgreSQL, el cual asegura mayor confiabilidad e integridad en los datos almacenados, además consta de un fuerte sistema de encriptación de contraseña, así como el nombre del usuario.<br>El patrón DAO implementa operaciones de gestión y obtención de datos, el cual asegura el acceso de los usuarios a la información que estos almacenaron ya sea con procedimientos almacenados o consultas. | | |

**Tabla 9 Evaluación del escenario Capa de Acceso a Datos**

Los sistemas informáticos están bajo constantes cambios y ninguno de ellos tiene completa seguridad. Por eso es necesaria una constante evaluación del diseño arquitectónico donde se pueden prever los principales riesgos en cada escenario.

En el diseño se observaron algunos atributos de calidad y su relación con escenarios del sistema. Algunos no se pudieron evaluar, porque es imposible observar determinadas características que el sistema presentará. Como consecuencia de esto, existen muchos riesgos que se recomienda mitigarlos en el desarrollo del sistema, con el refinamiento de la arquitectura y posterior evaluación de los atributos de calidad observables en tiempo de ejecución.

## Conclusiones

En este capítulo se hizo una evaluación de la arquitectura, aplicando el método ARID y la técnica basada en escenario con el instrumento de evaluación perfiles.

Se verificó que los requerimientos no funcionales están presentes en la arquitectura y que satisfacen los atributos de calidad evaluados, por lo que esta se considera robusta y factible. La arquitectura le brinda al sistema seguridad e integridad a los datos, además permite que sea mantenible, portable y escalable. Por lo anterior explicado, la arquitectura diseñada se puede aplicar para estructurar el J-IMMSIM 2.0..

## Conclusiones generales

- Se seleccionaron los estilos y patrones arquitectónicos, así como las tecnologías y herramientas para conformar la línea base de la arquitectura.
- Se identificaron los casos de usos del sistema, clases del diseño y componentes de implementación arquitectónicamente significativos para elaborar las vistas de sistema, confeccionando el documento descripción de la arquitectura donde quedaron plasmados todos los aspectos para el desarrollo J-IMMSIM 2.0.
- Se escogieron los métodos y las técnicas necesarias que propiciaron la evaluación de la efectividad de la arquitectura, especificando las ventajas y limitaciones del diseño arquitectónico.

## Recomendaciones

Luego de haber concluido el presente trabajo de diploma se recomienda:

- Desarrollar la aplicación J-IMMSIM 2.0 con la arquitectura propuesta.
- A medida que se esté implementado el J-IMMSIM 2.0, se debe refinar la arquitectura (si es necesario y está justificado).

## Referencias bibliográficas


1. **Pressman, Roges S.** *Ingeniería de Software un Enfoque Práctico.* 2005
2. **David Garlan y Mary Shaw**. *An Introduction to Software Architecture*. Software Engineering Institute, Carnegie Mellon University. 1994.
3. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* 2004.
4. Universidad Politécnica de Madrid. [En línea] [Citado el: 10 de diciembre de 2007.] www.gig.etsii.upm.es/jmcabanellas/Doctorado_0405/Doctor_Tecno_Simul_0405_guia_6.pdf.
5. **Carlos Reynoso, Nicolás Kicillof**. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.
6. **Grady Booch**. *Object-Oriented Design with Applications*. The Benjamin/Cummings Publishing Company, Inc. 1991.
7. **Christopher Alexander**. *A pattern language*. Oxford University Press, 1977.
8. **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. (GoF-Group of Four).** *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley. 2000
9. **Buschmann, Frank- Meunier , Regine – Rohnert, Hans- Sommerlad, Peter- Stal, Michael.** *Pattern-Oriented Software Architecture.* 2001.
10. **Larman, Craig.** *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development*. tercera edición. 2005.
11. **Carlos Reynoso-Nicolás Kicillof.** *Lenguajes de Descripción de Arquitectura*. [En línea] [Citado el: 10 de enero de 2008.] http://www.microsoft.com/spanish/msdn/default.mspx.
12. **Mendoza Sánchez, María A.** *Metodologías De Desarrollo De Software.* 2004 [En línea] [Citado el: 16 diciembre 2007] http://www.informatizate.net
13. **Rieco, Daniel – Romero, Daniel.** *Un Workflow que Automatice los Procesos de Negocios del Proceso Unificado Rational*. 2004.
14. **Lyons, Brian.** *The Open Unified Process: A Brilliant, Collaborative March into Open Source*. [En línea] [Citado el: 16 de diciembre de 2007.] www.numbersix.com.

15. Java en Castellano. [En línea] [Citado el: 28 de abril de 2008.] http://www.programacion.net/java

16. **Alur, Deepak – Malks, Dan- Crupi, John.** *JEE PATTERNS Best Practices and Design Strategies*. Segunda edición 2003

17. NetBeans [Citado el: 16 diciembre 2007][En linea] http://www.netbeans.org/index_es.html

18. **García Puebla, Iván.** *Introducción a la plataforma Eclipse*. 2005 [En línea]] [Citado el: 12 enero 2007] http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=56&Itemid=41

19. *The Apache Software Fundation*. 2007 [En Linea] [Citado el: 30 enero 2008] http://tomcat.apache.org/

20. *Rational, Rational Suite* [En línea] [Citado el: 16 diciembre 2007] http://www.rational.com.ar/herramientas/rationalsuite.html

21. *Why Visual Paradigm for UML?* . [En línea] [Citado el: 12 de enero de 2007.] http://www.visual-paradigm.com/product/vpuml.

22. **C.Costilla**. *Características Objeto-Relacionales del Sistema de Gestión de Bases de Datos Oracle*.2006**.**

23. Ventajas de PostgreSQL. [En línea] 2005. [Citado el: 12 de enero de 2008.] http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html.

24. **Booch, Grady - Jacobson, Ivar.** *The Unified Software Development Process***.** 2000.

25. **Quesada Ibarguen, Víctor Manuel - Vergara Schmalbach, Juan Carlos.** *Estadística básica con aplicaciones en Ms Excel.* 2005.

26. MasAdelante. Tipos de servidores [En línea]. [Citado el: 12 de enero de 2008.]. www.masadelante.com.

27. **Tanenbaum, Andrew S**. *Computer Networks*. Upper Saddle River. 2003.

28. **Bosch, J.** *Design & Use of Software Architectures*. s.l. : Addison-Wesley, 2000.

29. **Kazman, R., Clements, P., Klein, M**. *Evaluating Software Architectures.Methods and case studies.* Addison Wesley. 2001.

## Bibliografía


1. Documento sobre la Arquitectura de Software para los componentes a emplear por el Sistema de Información para la Salud. Softel., Junio / 2006.
2. Ayuda del OpenUp
3. Pressman, Roges S. *Ingeniería de Software un Enfoque Práctico.* 2005
4. David Garlan y Mary Shaw. An Introduction to Software Architecture. Software Engineering Institute, Carnegie Mellon University. 1993.
5. Reynoso, Carlos Billy. *Introducción a la Arquitectura de Software.* 2004.
6. Universidad Politécnica de Madrid. [En línea] [Citado el: 10 de diciembre de 2007.] www.gig.etsii.upm.es/jmcabanellas/Doctorado_0405/Doctor_Tecno_Simul_0405_guia_6.pdf.
7. Carlos Reynoso, Nicolás Kicillof. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004.
8. Grady Booch. Object-Oriented Design with Applications. The Benjamin/Cummings Publishing Company, Inc. 1991.
9. Christopher Alexander. A pattern language. Oxford University Press, 1977.
10. Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. (GoF-Group of Four). *Design Patterns. Elements of Reusable Object-Oriented Software.* Addison Wesley.
11. Buschmann, Frank- Meunier , Regine – Rohnert, Hans- Sommerlad, Peter- Stal, Michael. *Pattern-Oriented Software Architecture.* 2001.
12. Larman, Craig. *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development*. tercera edición. 2005.
13. Carlos Reynoso-Nicolás Kicillof. *Lenguajes de Descripción de Arquitectura*. [En línea] [Citado el: 10 de enero de 2008.] http://www.microsoft.com/spanish/msdn/default.mspx.
14. Mendoza Sánchez, María A. *Metodologías De Desarrollo De Software.* 2004 [En línea] [Citado el: 16 diciembre 2007] http://www.informatizate.net

15. Rieco, Daniel – Romero, Daniel. *Un Workflow que Automatice los Procesos de Negocios del Proceso Unificado Rational*. 2004.

16. Lyons, Brian. *The Open Unified Process: A Brilliant, Collaborative March into Open Source*. [En línea] [Citado el: 16 de diciembre de 2007.] www.numbersix.com.

17. Java en Castellano. [En línea] [Citado el: 28 de abril de 2008.] http://www.programacion.net/java

18. Alur, Deepak – Malks, Dan- Crupi, John. *JEE PATTERNS Best Practices and Design Strategies*. Segunda edición 2003

19. NetBeans [Citado el: 16 diciembre 2007][En linea] http://www.netbeans.org/index_es.html

20. García Puebla, Iván. *Introducción a la plataforma Eclipse*. 2005 [En línea]] [Citado el: 12 enero 2007] http://www.gui.uva.es/~laertes/nuke/index.php?option=com_content&task=view&id=56&Itemid=41

21. *The Apache Software Fundation*. 2007 [En Linea] [Citado el: 30 enero 2008] http://tomcat.apache.org/

22. *Rational, Rational Suite* [En línea] [Citado el: 16 diciembre 2007] http://www.rational.com.ar/herramientas/rationalsuite.html

23. *Why Visual Paradigm for UML?* . [En línea] [Citado el: 12 de enero de 2007.] http://www.visual-paradigm.com/product/vpuml.

24. C.Costilla. *Características Objeto-Relacionales del Sistema de Gestión de Bases de Datos Oracle*.2006.

25. Ventajas de PostgreSQL. [En línea] 2005. [Citado el: 12 de enero de 2008.] http://soporte.tiendalinux.com/portal/Portfolio/postgresql_ventajas_html.

26. Booch, Grady - Jacobson, Ivar. *The Unified Software Development Process*. 2000.

27. Quesada Ibarguen, Víctor Manuel - Vergara Schmalbach, Juan Carlos. *Estadística básica con aplicaciones en Ms Excel.* 2005.

28. MasAdelante. Tipos de servidores [En línea]. [Citado el: 12 de enero de 2008.]. www.masadelante.com.

29. Tanenbaum, Andrew S. *Computer Networks*. Upper Saddle River. 2003.

30. Bosch, J. *Design & Use of Software Architectures*. s.l. : Addison-Wesley, 2000.

31. Kazman, R., Clements, P., Klein, M. *Evaluating Software Architectures.Methods and case studies.* Addison Wesley. 2001.

32. Guevara Vargas, Yusmilia – Pérez Díaz, Bismark. Sistemas de manejo de Ensayos Clínicos: Diseño de la Arquitectura. Ciudad de La Habana. Tesis. 2007

# Anexos:

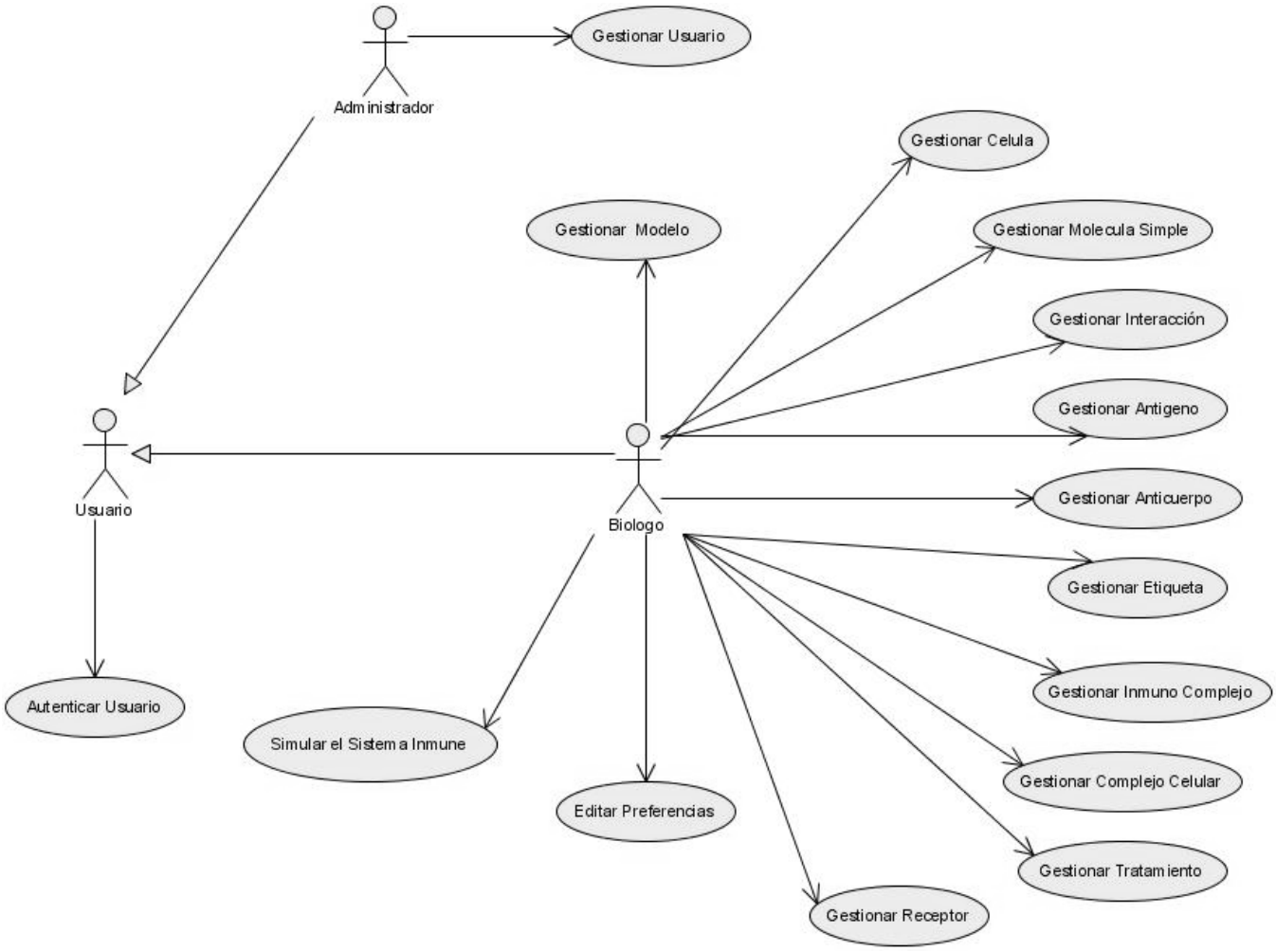


**Figura 10 Diagrama de casos de uso**

## Glosario de términos

**POSA**: Pattern-Oriented Software Architecture (Patrones orientados de la Arquitectura de Software).

**Interoperabilidad**: Es la condición mediante la cual sistemas heterogéneos pueden intercambiar procesos o datos.

**IEEE:** The Institute of Electrical and Electronics Engineers, el Instituto de Ingenieros Eléctricos y Electrónicos, es una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Es la mayor asociación internacional sin fines de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación, ingenieros en informática e ingenieros en telecomunicación.

**Persistente**: Cuyo estado de es almacenado en un medio secundario para su posterior reconstrucción y utilización, por lo que su tiempo de vida es independiente del proceso que los instanció.

**CIM:** Centro de Inmunología Molecular.

**BD**: Base de datos.

**MATLAB:** Es la abreviatura de MATrix LABoratory (laboratorio de matrices). Se trata de un software matemático muy versátil que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M).

**Broadcast:** Un dominio de difusión (broadcast) es un área lógica en una red de computadoras en la que cualquier computadora conectada a la red puede transmitir directamente a cualquier otro en el dominio sin precisar ningún dispositivo de encaminamiento, dado que comparten la misma subred y dirección de puerta de enlace.

**Servlets:** Es un programa escrito en Java que se ejecuta en un servidor. El uso más común de los servlets es generar páginas web de forma dinámica a partir de los parámetros de la petición que envíe el navegador web.

**XML:** Sigla en inglés de Extensible Markup Language («lenguaje de marcas extensible»), es un metalenguaje extensible de etiquetas.

**Clúster:** El término clúster se aplica a los conjuntos de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora.

**Entes biológicos**: Células, moléculas, complejos moleculares, anticuerpos y antígenos.

**Patógenos:** Microorganismos capaces de producir una enfermedad infecciosa, incluye a los virus, bacterias, hongos y protozoos.

**Trigger:** Es un evento en una base de datos que se ejecuta cuando se cumple una condición establecida al realizar una operación.

**API (Application Programming Interface):** Es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

**Taxonomía:** Ciencia que se ocupa de los principios, métodos y fines de la clasificación.

**Ingeniería de Software:** La Ingeniería de software designa el conjunto de técnicas destinadas a la producción de un programa de computadora.

**Caché**: Conjunto de datos duplicados de otros originales, con la propiedad de que los datos originales son costosos de acceder, normalmente en tiempo, respecto a la copia en el caché.

**JSP (Java Server Pages**): Es una tecnología Java que permite generar contenido dinámico para web.

**CPU (Central Processing Unit):** Unidad central de procesamiento o simplemente procesador, es el componente en una computadora digital que interpreta las instrucciones y procesa los datos contenidos en los programas de computadora.

**RAM (Random Access Memory**): La memoria de acceso aleatorio se compone de uno o más chips y se utiliza como memoria de trabajo para programas y datos.

**SSL:** Secure Socket Layers

**HDD (Hard Drive Disk):** Disco duro.