Universidad de las Ciencias Informáticas
Facultad 6

Título: Rediseño de la arquitectura del Software BioSyS.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autor(es): Wendy Wong Iglesias.

Yuriskenia Sarmiento Reyes.

Tutor(es): Ing. Yadira Marrero López.

Ing. Adonis R. Rosales García.

Ciudad de la Habana, 11 de Junio de 2010.

“El hombre que se levanta es aún más grande que el que no ha caído."

Concepción Arenal

DECLARACIÓN DE AUTORÍA

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

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

Wendy Wong Iglesias

_____
Firma del Autor

Yuriskenia Sarmiento Reyes.

_____
Firma del Autor

Ing. Yadira Marrero López

_____
Firma del Tutor

Ing. Adonis R. Rosales García.

_____
Firma del Tutor

Datos de Contacto

Autoras:

Yuriskenia Sarmiento Reyes
Universidad de las Ciencias Informáticas, Habana, Cuba.
E-mail: ysreyes@estudiantes.uci.cu

Wendy Wong Iglesias
Universidad de las Ciencias Informáticas, Habana, Cuba.
E-mail: wwong@estudiantes.uci.cu

Tutores:

Ing. Yadira Marrero López.
Universidad de las Ciencias Informáticas, Habana, Cuba.
E-mail: ymlopez@uci.cu

Ing. Adonis R. Rosales García.
Universidad de las Ciencias Informáticas, Habana, Cuba.
E-mail: arrosales@uci.cu

AGRADECIMIENTOS

Wendy:

El agradecimiento más importante de todos es para mi Querida y adorada Madre , que es la persona que me ha apoyado en cada momento de mi vida, que ha estudiado junto conmigo todo este tiempo y que es la persona más importante de mi vida.
A mi hermano Alejandro por ser la persona que siempre está junto a mí y por creer en mí, eres el mejor hermano del mundo.
A toda mi familia, a mi tía Elba, a mi abuelita linda, a mis primos Leandro y Amaia, a mi tío Leonardo, en fin a todos los que me han apoyado en los momentos buenos y malos.
A mis tutores por la ayuda incondicional que nos han brindado, sin ellos no hubiésemos podido realizar este trabajo.
A Edel y Carlos que también formaron una parte importante de este trabajo, y de verdad que se los agradezco mucho.
A mis amigas queridas Yuriskenia, Evelyn y Veylys, que este último año hemos estado más unidas que nunca tanto en los momentos buenos como en los malos.
Agradezco este trabajo y toda mi carrera primero que todo a mis amigos de la Universidad, que estuvieron ahí durante todo este tiempo y que de una manera u otra formaron parte mi formación tanto profesional como personal.

Yuriskenia:

Primeramente quiero agradecerle a los seres más maravillosos de la tierra mis padres pues sin su lucha incansable de hacerme saber que si se puede nada de esto fuera posible.
A mis dos grandes hermanos Yuri y Oscarito.
A las tías más bellas del mundo Elena, Lina, Francis, Dorita y mi prima Yurisma por estar ahí siempre para mí.
A mis abuelitos que de una forma u otra ayudaron en mi crecimiento.
A mi adorado novio Raúl Concepción González por soportarme todo el tiempo y por su ayuda incondicional.
A mi segunda gran familia que me acogió como su hija y que simplemente se han dedicado a darme mucho amor Alba, Raúl y Ernesto.
A mis amigas de los años que siempre pusieron su granito de arena Michy, Liyanis y Yamira.
Al club de las Supernenas que a pesar de las distancias se preocuparon por mí May, Kil, Yeni, Mire, Yas, Liset y Kenia.
A Edel y Carlos, les agradezco con la vida que nos hayan ayudado.
A nuestros tutores que nos recogieron cuando creíamos que todo estaba perdido.
A mis nuevas guerreras de lucha Wendy, Evelyn y Veylys que estuvimos muy unidas para enfrentarlo todo.
En fin a todos los que han hecho posible que este sueño se haga realidad muchas gracias.

DEDICATORIA

Wendy:

Le dedico este trabajo a las personas más importantes en mi vida, mi madre y a mi querido y adorado hermano, que siempre creyeron en mí. Los quiero mucho.

Yuriskenia:

Quiero dedicarle esta tesis especialmente a mis padres Adela y Fidel por ser mi razón ser, por estar siempre en las buenas y en las malas y por poner todo su empeño en convertirme en una mejor persona.

A mi hermanita del alma que aunque nos veamos poco siempre está en mi corazón.

Y a mi queridísimo abuelo Rey que siempre quiso verme convertida en una profesional, esto es para él.

Ambas:

A la Revolución y a nuestro comandante en jefe Fidel Castro por darnos la oportunidad de pertenecer a esta genial idea que es la Universidad de las Ciencias Informática.

RESUMEN

El Software para la Simulación y Análisis de los Sistemas Biológicos (alasBioSyS) es una herramienta de propósito general que facilita la modelación, edición, simulación y análisis de los sistemas biológicos. La investigación del presente trabajo surge de forma conjunta por el Centro de Inmunología Molecular (CIM) y la Universidad de las Ciencias Informáticas (UCI), dada la necesidad de obtener una integración entre los componentes del software BioSyS que contribuya a la flexibilidad del sistema. Se decidió diseñar una arquitectura que proporcione los elementos necesarios para el desarrollo de una plataforma, que brinde a los investigadores un entorno de trabajo provisto de las funcionalidades necesarias para la evolución de sus investigaciones.

La representación de la arquitectura se realiza mediante el Modelo 4 + 1 vistas de Philippe Kruchten, se identifican los elementos críticos o sensibles desde el punto de vista arquitectónico, así como también del diseño/implementación indispensables para el buen funcionamiento del sistema y se evalúa el diseño arquitectónico aplicando el método de evaluación ARID permitiendo validar que la arquitectura propuesta cumple con los atributos de calidad indispensables para el cumplimiento de la calidad del sistema.

PALABRAS CLAVE

Arquitectura de Software, Sistemas Bilógicos, Estilos Arquitectónicos, Patrones Arquitectónicos, Vistas Arquitectónicas.

TABLA DE CONTENIDO

INTRODUCCIÓN

La ciencia interesada en comprender el funcionamiento de los sistemas biológicos es la Biología de Sistemas, a través de ella se han desarrollado diferentes programas con el objetivo de modelar, simular y analizar sistemas biológicos, algunos de ellos de propósito general y otros más específicos.

La Biología de Sistemas es un área de estudio emergente que se caracteriza por la integración de aproximaciones experimentales y computacionales en la comprensión de los sistemas biológicos. Esta disciplina permite estudiar los mecanismos que gobiernan los sistemas complejos y las interacciones existentes entre los diferentes niveles de información biológica [1].

Para la mejor comprensión de los Sistemas Biológicos, surge una disciplina científica emergente llamada Bioinformática. Esta área de investigación multidisciplinaria puede ser ampliamente definida como la interfase entre dos ciencias: Biología y Computación, ya que utiliza tecnología de la información para organizar, analizar y distribuir información biológica con la finalidad de responder preguntas complejas en esta rama.

En los últimos años se han creado las primeras empresas de biología de sistemas cuyos productos están estrechamente relacionados con la simulación y análisis de problemas biológicos. Gene Network Sciences, Entelos, Physiome y Genomática son ejemplos de estas empresas. Estas trabajan sobre una plataforma de software que permite integrar información, representar matemáticamente los diferentes modelos y simular sistemas biológicos, pero su estrategia de negocios está basada en la formación de alianzas comerciales con empresas biotecnológicas y no en la comercialización directa del software.

El actual desarrollo de software de forma independiente también ha ido en ascenso. Existen referencias a diferentes programas en la página Web del System Biology Markup Lenguage (SBML), que permiten modelar, simular o realizar análisis sobre modelos de sistemas biológicos, algunos de estos programas son Vcell, CellWare y BioSpice.

En Cuba se ha creado el Centro de Inmunología Molecular (CIM) que tiene como principal misión obtener y producir nuevos biofármacos destinados al tratamiento del cáncer y otras enfermedades crónicas no transmisibles e introducirlos en la Salud Pública cubana. Para este centro es muy importante realizar simulaciones para analizar el comportamiento de estas enfermedades.

Teniendo en cuenta estas razones, el Centro de Inmunología Molecular (CIM) en conjunto con la Facultad 6 de la Universidad de las Ciencias Informáticas (UCI), crean el proyecto Software para la Simulación y Análisis de los Sistemas Biológicos (BioSyS), con el objetivo de crear una herramienta de propósito general que facilite la modelación, edición, simulación y análisis de los mismos.

El software BioSyS cuenta con herramientas para la modelación matemática, un editor de ecuaciones y el graficador, algoritmos para la realización de simulaciones distribuidas y para el análisis de las simulaciones realizadas donde incluye la estimación de parámetro, y una base de datos para almacenar los resultados de las simulaciones. Estos algoritmos representan módulos que pueden ser utilizados como librerías genéricas en otros sistemas con las mismas características.

La primera versión del sistema funciona como un todo, lo que implica que se realice todo el trabajo desde el inicio y que los investigadores tengan que usar el sistema completo para realizar su trabajo, independientemente de las funcionalidades que desea utilizar, limitando las posibilidades de escalabilidad y excluyendo la independencia de funcionalidades.

De la situación planteada anteriormente se deriva el **problema científico** que guía la investigación: ¿Cómo obtener una integración entre los componentes del software BioSyS que contribuya a la flexibilidad del sistema?

En correspondencia con el problema científico planteado se define como **objeto de estudio:** La Arquitectura de Software, enmarcado en el **campo de acción**: La Arquitectura de Software para la Simulación y Análisis de Sistemas Biológicos.

Estrechamente vinculado al campo de acción aparece como **objetivo general:** Diseñar la nueva arquitectura del software BioSyS.

Posteriormente y con la intención de cumplir con el objetivo general se fijan los siguientes **objetivos específicos:**

1. Seleccionar estilos y patrones de arquitectura.
2. Describir la arquitectura del sistema.
3. Evaluar el diseño arquitectónico propuesto.

Para cumplir con los objetivos específicos se realizarán las siguientes **tareas:**

1. Revisión de los estilos arquitectónicos existentes.
2. Revisión de los patrones de arquitectura existentes.
3. Identificación de los estilos arquitectónicos a utilizar, fundamentación de estos estilos, así como su aplicación.
4. Identificación de los patrones arquitectónicos a utilizar, fundamentación de estos patrones, así como su aplicación.
5. Definición y fundamentación de las herramientas a utilizar para el desarrollo del software BioSyS.
6. Representación del diseño arquitectónico propuesto.

7. Análisis y diseño de las vistas del sistema.
8. Selección y aplicación de un método para validar el diseño arquitectónico propuesto.

El presente documento está estructurado en tres capítulos.

Capítulo 1: Fundamentación Teórica.
En este capítulo se presentan conceptos asociados al objeto de estudio, así como los diferentes tipos de arquitectura y patrones arquitectónicos más usados mundialmente. Se aborda acerca de los lenguajes, tecnologías y herramientas ya definidas para dar solución al problema científico.

Capítulo 2: Descripción de la arquitectura.
En este capítulo se hace una propuesta de solución al problema planteado. Se diseñan las vistas arquitectónicas y se hace una descripción de la arquitectura para la plataforma, definiendo cómo debe hacerse la integración y comunicación entre los módulos.

Capítulo 3: Evaluación del diseño arquitectónico propuesto.
En este capítulo se presentan los distintos métodos para evaluar un diseño arquitectónico y a partir del método seleccionado se hace un análisis de la solución propuesta.

CAPÍTULO 1

Fundamentación Teórica

CAPÍTULO 1

1.1 INTRODUCCIÓN

En este capítulo se presentan conceptos asociados al objeto de estudio, así como los diferentes tipos de arquitectura y patrones arquitectónicos más usados mundialmente. Se aborda acerca de los lenguajes, tecnologías y herramientas que serán utilizadas para dar solución al problema científico.

1.2 Arquitectura de Software

1.2.1 Surgimiento de Arquitectura de Software

La Arquitectura de Software (AS) remonta sus antecedentes a la década de 1960, su historia no ha sido continua como la del campo más amplio en la que se inscribe, la ingeniería de software. Desde ese entonces se han interpretado de muchas maneras las diferentes ideas que se plantean, permitiendo distinguir corrientes de pensamientos diversas, cuyas diferencias distan de ser triviales a la hora de plasmar las ideas.

En el año 1968 Edsger Dijkstra, de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972, propuso que se establezca una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera. Dijkstra sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas. Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para lo que luego serían las primeras definiciones de Arquitectura de Software [2].

En 1975, Brooks, diseñador del sistema operativo OS/360 y Premio Turing 2000, utilizaba el concepto de arquitectura del sistema para designar “la especificación completa y detallada de la interfaz de usuario” y consideraba que el arquitecto es un agente del usuario, igual que lo es quien diseña su casa, empleando una nomenclatura que ya nadie aplica de ese modo. En el mismo texto, identificaba y razonaba sobre las estructuras de alto nivel y reconocía la importancia de las decisiones tomadas a ese nivel de diseño. También distinguía entre arquitectura e implementación; mientras aquella decía qué hacer, la implementación se ocupa de cómo y planteaba también que las decisiones tempranas de desarrollo serían las que probablemente permanecerían invariantes en el desarrollo de una solución. Esas “decisiones tempranas” constituyen de hecho lo que hoy se llamarían decisiones arquitectónicas [2].En la década de 1980, los métodos de desarrollo estructurado demostraron no escalar suficientemente y

fueron dejando el lugar a un nuevo paradigma, el de la programación orientada a objetos (POO). En teoría, parecía posible modelar el dominio del problema y el de la solución en un lenguaje de implementación. Hacia fines de la década de 1980 y comienzos de la siguiente, la expresión arquitectura de software comienza a aparecer en la literatura para hacer referencia a la configuración morfológica de una aplicación [2].

Puede decirse que Perry y Wolf fundaron la disciplina, y fueron los primeros en dar el sentido que actualmente conocemos por “Arquitectura de Software”. Algunos autores proponen la AS por analogía con la arquitectura de los edificios, analogía de la que algunos abusaron, otros encontraron útil y para unos pocos inaceptables, planteando que primero se desarrolla una intuición para la AS recurriendo a diversas disciplinas arquitectónicas bien definidas.

Dando cumplimiento a las profecías de Perry y Wolf, la década de 1990 fue sin duda la de la consolidación y diseminación de la AS en una escala sin precedentes. Las contribuciones más importantes surgieron en torno del instituto de ingeniería de la información de la Universidad Carnegie Mellon (CMU SEI), antes que cualquier organismo de industria. En la misma década, demasiado pródiga en acontecimientos, surge también la programación basada en componentes, que en su momento de mayor impacto impulsó a algunos arquitectos mayores, como Paul Clements, a afirmar que la AS promovía un modelo que debía ser más de integración de componentes pre-programados que de programación [3].

Otro tema muy importante de la época fue el surgimiento de los patrones, cristalizada en dos textos fundamentales, el de la Banda de los Cuatro en 1995 y la serie POSA desde 1996. 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 dominada por estrategias orientadas a líneas de productos y por establecer modalidades de análisis, diseño, verificación, refinamiento, recuperación, diseño basado en escenarios, estudios de casos y hasta justificación económica, redefiniendo todas las metodologías ligadas al ciclo de vida en términos arquitectónicos. Todo lo que se ha hecho en ingeniería de software debe formularse de nuevo, integrando la AS al conjunto.

Definición de Arquitectura de Software

La Arquitectura de Software es la parte de la Ingeniería de Software que se ocupa de la descripción y el tratamiento de la estructura de un sistema como una serie de componentes, con el fin de organizar adecuadamente los distintos subsistemas, y permitir la integración de diferentes grupos de desarrollo en

el mismo proyecto. Habitualmente se vincula con la fase de Diseño, aunque este detalle no es estrictamente necesario; su principal objetivo es hacer énfasis en la importancia de la descripción estructural de los sistemas software, un aspecto bien conocido pero habitualmente poco tratado [4].

Incluso hoy, con la disciplina relativamente establecida pero aún joven, no es posible dar una definición o descripción de Arquitectura de Software que resulte completa, o cuando menos suficientemente integradora.

Muchos han sido los autores que han abordado el tema de AS y al mismo tiempo han realizado diferentes definiciones por lo que se mencionan las que de alguna forma llevaron el camino a lo que se considera hoy AS.

Perry y Wolf fueron los fundadores de esta disciplina planteando que:

“Una arquitectura de software es un conjunto de elementos arquitectónicos que tienen una determinada forma. Las propiedades restringen la elección de los elementos de la arquitectura mientras que la lógica captura la motivación de la elección de los elementos y la forma [5].”

(Perry y Wolf 1992)

Según Mary Shaw y David Garlan:

“Una arquitectura de software incluye la descripción de elementos a partir de los cuales se construyen los sistemas de software, interacciones entre esos elementos, patrones que guían la composición y restricciones sobre esos patrones. En general, un sistema de software particular se define en términos de una colección de componentes e interacciones entre dichos componentes. Tal sistema puede ser utilizado como un elemento en sistemas más grandes [6].”

(Garlan y Shaw 1996)

La definición oficial que es utilizada es la provista por el documento lEEE STD 1471- 2000, que expresa:

“La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución [7].”

Ninguna de las definiciones de Arquitectura de Software es respaldada unánimemente por la totalidad de los arquitectos de software; a pesar de las diferencias entre las diversas definiciones es común entre todos los autores el concepto de la arquitectura como un punto de vista que concierne a un alto nivel de abstracción.

1.3 Estilos y Patrones

Uno de los elementos importantes en la arquitectura de software es definir los estilos y patrones. Antes de analizar estos términos es importante mencionar las diferencias que existen entre estos conceptos.

En algunas bibliografías referentes a los patrones, los autores suelen llamar de la misma manera a los estilos y patrones, mientras que la mayoría lo ven de manera separada. Ambos se refieren a formas de estructurar los sistemas y cómo relacionar los componentes de estos, la diferencia radica en el nivel de abstracción en que se aplican. Los estilos favorecen un tratamiento estructural que concierne más bien a la teoría, la investigación académica y la arquitectura en el nivel de abstracción más elevado, mientras que los patrones se ocupan de cuestiones que están más cerca del diseño, la práctica, la implementación, el proceso, el refinamiento y el código [8].

Los patrones de arquitectura están claramente dentro de la disciplina arquitectónica, solapándose con los estilos, mientras que los patrones de diseño se encuentran más bien en la periferia, si es que no decididamente afuera.

Vale la pena aclarar la relación entre estilos, patrones de diseño y patrones de arquitectura. Los patrones de diseño de software buscan codificar y hacer reutilizables un conjunto de principios a fin de diseñar aplicaciones de alta calidad. Los estilos se han aplicado en la fase de análisis arquitectónico en términos de patrones de arquitectura. Los patrones de diseños se aplican en principio solo en la fase de diseño, aunque la comunidad ha comenzado a definir y aplicar patrones a otras etapas del proceso de desarrollo. Los patrones de arquitectura se concentran en la estructura de alto nivel del sistema y presenta similitudes con los de diseño.

Los estilos expresan la arquitectura en el sentido más formal y teórico. Son una categorización de sistemas y son independientes al contexto donde pueden ser aplicados, por lo que expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual del diseño. Los patrones son soluciones generales a problemas comunes. Expresan un problema recurrente de diseño muy específico y presentan una solución para él, desde el punto de vista del contexto en que se presenta [8].

Después de analizar los elementos esenciales de los estilos y patrones se concluye que si bien existe convergencia entre ellos, los estilos se refieren más a la teoría sobre la estructura de los sistemas y los patrones se refieren más a las prácticas de reutilización.

1.3.1 Estilos

Los estilos sólo se manifiestan en la arquitectura teórica descriptiva de alto nivel de abstracción, constituyendo así una parte importante del sistema. Se definen como aspectos formales a partir de diversas arquitecturas específicas y encapsula decisiones esenciales sobre los elementos del sistema. Enfatiza restricciones importantes de los elementos y sus relaciones posibles.

Se entiende por estilos a las entidades que ocurren en un nivel sumamente abstracto, puramente arquitectónico, que no coincide ni con la fase de análisis propuesta por la temprana metodología de modelado orientada a objetos (aunque sí un poco con la de diseño), ni con lo que más tarde se definirían como paradigmas de arquitectura, ni con los patrones arquitectónicos [9].

Algunos de los estilos arquitectónicos que se utilizan en la actualidad están divididos en Clases de estilos las cuales se exponen a continuación [9].

- Estilos de Flujo de Datos: Esta familia enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.

- **Tubería y filtros:** El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Estos datos entran al sistema y fluyen a través de los componentes.

- Estilos Centrados en Datos: Esta familia enfatiza la integración de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento**.**

- **Arquitecturas de Pizarra o Repositorio:** En este estilo se han definidos dos subcategorías principales:
  - Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
  - Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

- Estilos de llamada y Retorno: Esta familia enfatiza la modificación y la escalabilidad. Dentro de esta familia se encuentran las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

- **Modelo Vista Controlador (MVC):** Este patrón separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

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

- **Arquitectura en capas:** Definida por Garlan y Shaw 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.
- **Arquitecturas Orientadas a Objetos**: Los componentes del estilo se basan en principios orientados a objetos como encapsulamiento, herencia y polimorfismo. .
- **Arquitecturas Basadas en Componentes:** Permite alcanzar un mayor nivel de reutilización de software y que las pruebas sean ejecutadas probando cada uno de los componentes, antes de probar el conjunto completo de componentes ensamblados. Simplifica el mantenimiento del sistema, cuando existe un débil acoplamiento entre componentes, el desarrollador es libre de actualizar y/o agregar componentes según sea necesario, sin afectar otras partes del sistema. Dado que un componente puede ser construido y luego mejorado continuamente por un experto u organización, la calidad de una aplicación basada en componentes mejorará con el paso del tiempo.

- **Estilos de Código Móvil:** Esta familia 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 (MV):** Este estilo se utiliza habitualmente para construir máquinas virtuales que reducen el vacío que media entre la máquina de computación esperada por la semántica del programa y la máquina físicamente disponible.

- **Estilos heterogéneos:** En esta familia se clasifican aquellos sistemas que no pueden encajar exactamente en ninguno de los tipos anteriores.

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

- **Arquitecturas Basadas en Atributos**: La arquitectura basada en atributos fue propuesta para asociar a la definición del estilo arquitectónico un framework de razonamiento, basado en modelos de atributos específicos.

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

- **Arquitecturas Basadas en Eventos:** Se vinculan históricamente con sistemas basados en publicación-suscripción. Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos.
- **Arquitecturas Orientadas a Servicios (SOA):** Es lo suficientemente flexible, elegante y ágil garantizando las soluciones que las empresas han anhelado siempre. Es una arquitectura de software que construye todo el estudio 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.
- **Arquitecturas Basadas en Recursos**: Define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web (www), donde los URLs identifican los recursos y HTTP es el protocolo de acceso.

En el epígrafe 1.8 luego de analizar las características del sistema se definirá el estilo arquitectónico seleccionado.

1.3.2 Patrones de Software

Los patrones de arquitectura expresan el esquema fundamental de organización para sistemas de software. Proveen un conjunto de subsistemas predefinidos; especifican sus responsabilidades e incluyen reglas y guías para organizar las relaciones entre ellos [9].

Los patrones de software son mecanismos con el objetivo de dar solución a problemas que ocurren repetidamente dentro de un contexto muy bien definido. Las soluciones propuestas a través de estos, involucran algunas clases de estructuras que permiten contemplar los requisitos no funcionales. Estos requisitos atraviesan diferentes niveles de abstracción y etapas del ciclo de vida, partiendo del análisis del

dominio, pasando por la arquitectura de software y llegando hasta el nivel de los lenguajes de programación [9].

Para la utilización de los patrones necesarios para crear un sistema, se deben tener en cuenta las características que debe tener un buen patrón para poder aplicarlo, basándose en [9]:

- La solución de un problema planteado.
- Que sea un concepto probado.
- Que la solución no sea obvia.
- Que describe participantes, relaciones entre ellos y un alto componente humano como estética y utilidad.

Dependiendo del autor, del nivel de abstracción y de la publicación misma se han presentado varios formatos para encapsular la información de un patrón.

Los puntos más significativos que debe contener un patrón son [9]:

Nombre: Es el nombre con el que ha sido registrado el patrón.

Problema: Es el problema específico que resuelve el patrón en cuestión y que además, ha sido el responsable del surgimiento de dicho patrón.

Solución: Es la solución que se le da al problema.

Estructura: Puede ser un diagrama que represente todos los implicados en el patrón (refiriéndonos a objetos), así como la relación existente entre estos.

Aplicación en la programación: Es un ejemplo concreto de cómo este patrón, que puede ser explicado mediante un ejemplo de la vida práctica, se aplica en un determinado sistema, es decir, un problema especifico que se presenta en una aplicación a desarrollar, y como el patrón es aplicable para resolverlo.

Ejemplo de Código: Es un pequeño ejemplo de la implementación del patrón, puede ser en cualquier lenguaje.

Contexto Resultante: El estado en el cual queda el sistema después de aplicar el patrón y las consecuencias de hacerlo

Racionalidad: Una explicación justificada de los pasos o reglas en el patrón

Relaciones: Relaciones estáticas y dinámicas del patrón con otros

Usos conocidos: Describe ocurrencias del patrón conocidas y su aplicación dentro de los sistemas existentes

Los patrones se pueden dividir en varias categorías según la escala o nivel de abstracción siendo estas [9]:

- Patrones de arquitectura: Son aquellos que expresan un esquema organizativo estructural fundamental para sistemas.

- Patrones de diseño: Expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.

- Patrones de análisis: Permite el modelado del dominio, la completitud, la integración y el equilibrio de objetivos múltiples, y el planeamiento para capacidades adicionales comunes.

- Patrones de procesos o de organización: Permite la productividad y la comunicación efectiva y eficiente.

- Patrones de Idiomas: Patrones de bajo nivel específicos para un lenguaje de programación o entorno concreto.

Los patrones más significativos para la arquitectura se software son los patrones arquitectónicos y de diseño, a continuación se muestra el estudio referente a estos.

1.3.2.1 Patrones Arquitectónicos

Los patrones arquitectónicos son los que definen la estructura de un sistema software, los cuales a su vez se componen de subsistemas con sus responsabilidades. Tienen una serie de directivas para organizar los componentes del sistema, con el objetivo de facilitar la tarea del diseño.

Entre las principales ventajas de esto patrones podemos encontrar:

- Sintetizan las soluciones arquitectónicas y estructurales dentro del tipo de problema que modelan.
- Permiten identificar y comprender la arquitectura del sistema.
- Permiten la reutilización de soluciones arquitectónicas de calidad.
- Son de gran ayuda para controlar la complejidad de un diseño.
- Facilitan la documentación de diseños arquitectónicos.
- Proporcionan un vocabulario común que mejora la comunicación entre diseñadores.

Un patrón de arquitectura de software, es un esquema genérico probado para solucionar un problema particular recurrente que surge en un cierto contexto. Este esquema se especifica describiendo los componentes, sus responsabilidades, relaciones, y las formas en que colaboran.

Los patrones arquitectónicos de software más destacados son: Tubería y Filtros, Pizarra o Repositorio, Modelo-Vista-Controlador (MVC), Capas (Layers), Broker, Presentation Abstraction Control, Reflection y

Microkernel. Una buena referencia en este tema lo constituye el libro “Pattern-Oriented Software Architecture: A System of Patterns” de F. Buschmann. [20]

- MVC: Para el diseño de aplicaciones con sofisticadas interfaces. La lógica de una interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio. Se trata de realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad.

- Tuberías y Filtros: El patrón de arquitectura de tubos y filtros provee una estructura para procesar flujos de datos. Cada paso de procesamiento se encapsula en un filtro. Los datos se pasan usando los tubos entre filtros adyacentes. Combinando los filtros se puede construir distintas familias de sistemas relacionados.

- Pizarra o Repositorio: En el modelo de repositorio los datos compartidos son pasivos y el control lo manejan los componentes que lo acceden. Estos componentes del sistema deben compartir información para trabajar efectivamente. Toda la información está contenida en una base de datos central, el repositorio.

- Capas (Layers): Este patrón descompone una aplicación en un conjunto de capas independientes y ordenadas jerárquicamente. Cada nivel o capa usa los servicios de la capa inmediatamente inferior y ofrece servicios a la inmediatamente superior.

- Broker: Se usa para organizar sistemas distribuidos con componentes débilmente acoplados que interactúan entre sí invocando servicios remotos.

- Presentaction Abstract 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.

- Reflection: Proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. Cambia dinámicamente soportando así la modificación de aspectos fundamentales como estructuras tipo y mecanismos de llamadas a funciones.

- Microkernel: Este patrón permite separar un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente. Se aplica para sistemas de software que deben estar en capacidad de adaptar los requerimientos de cambio del sistema y separa un núcleo funcional mínimo del resto de la funcionalidad y de partes específicas pertenecientes al cliente.

1.3.2.2 Patrones de Diseño

Los patrones de diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. Cada patrón permite que algunos aspectos de la estructura del sistema puedan cambiar independientemente de otros aspectos. Facilitan la reusabilidad, extensibilidad y mantenimiento.

Cuando se habla de estos patrones no se da una definición como tal, pero se considera que son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la Programación Orientada a Objetos (POO). Un patrón de diseño puede considerarse como un documento que define una estructura de clases que aborda una situación particular, permitiendo la reutilización de cada patrón las veces que sean necesarias, simplificando así el tiempo en el desarrollo del sistema [20].

Entre los patrones de diseño se destacan los patrones GRASP y los GOF. Los primeros describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades y los segundos son patrones conformados por el grupo de los cuatro: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

Siguiendo el libro de GOF los patrones se clasifican según el propósito para el que han sido definidos:

- **Creacionales:** Solucionan problemas de creación de instancias. Nos ayudan a encapsular y abstraer dicha creación.
- **Estructurales:** Solucionan problemas de composición/agregación de clases y objetos.
- **Comportamiento:** Solucionan problemas respecto a la interacción y responsabilidades entre clases y objetos, así como los algoritmos de encapsulamiento.

- **Patrones Creacionales** [18]:

Abstract Factory: Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.

Factory Method (Virtual Constructor): Define una interfaz para crear un objeto, pero deja que sean las subclases quienes decidan qué clase instanciar. Permite que una clase delegue en sus subclases la creación de objetos.

Singleton (Solitario): Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.

Prototype (Prototipo): Especifica los tipos de objetos a crear por medio de una instancia prototípica, y crear nuevos objetos copiando este prototipo.

Builder: Separa la construcción de un objeto complejo de su representación, de forma que el mismo proceso de construcción pueda crear diferentes representaciones.

- **Patrones Estructurales:**

Composite (Compuesto): Combina objetos en estructuras de árbol para representar jerarquías de parte-todo. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.

Decorator (Decorador): Añade dinámicamente nuevas responsabilidades a un objeto, proporcionando una alternativa flexible a la herencia para extender la funcionalidad.

Proxy (Apoderado): Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Facade (Fachada): Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.

Flyweight (Peso Mosca): Usa el compartimiento para permitir un gran número de objetos de grano fino de forma eficiente.

Adapter (Adaptador): Convierte la interfaz de una clase en otra distinta que es la que esperan los clientes. Permiten que cooperen clases que de otra manera no podrían por tener interfaces incompatibles.

Bridge (Puente): Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.

- **Patrones de Comportamiento:**

Chain of Responsibility (Asignación de responsabilidad): Evita acoplar el emisor de una petición a su receptor, al dar a más de un objeto la posibilidad de responder a la petición. Crea una cadena con los objetos receptores y pasa la petición a través de la cadena hasta que esta sea tratada por algún objeto.

Command: Encapsula una petición en un objeto, permitiendo así parametrizar a los clientes con distintas peticiones, encolar o llevar un registro de las peticiones y poder deshacer la operaciones.

Mediator (Mediador): Define un objeto que encapsula cómo interactúan un conjunto de objetos. Promueve un bajo acoplamiento al evitar que los objetos se refieran unos a otros explícitamente, y permite variar la interacción entre ellos de forma independiente.

Iterator (Iterador): Proporciona un modo de acceder secuencialmente a los elementos de un objeto agregado sin exponer su representación interna.

Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.
Memento (Recuerdo): Representa y externaliza el estado interno de un objeto sin violar la encapsulación, de forma que éste puede volver a dicho estado más tarde.
State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambia su estado interno. Parecerá que cambia la clase del objeto.
Strategy (Estrategia): Define una familia de algoritmos, encapsula uno de ellos y los hace intercambiables. Permite que un algoritmo varíe independientemente de los clientes que lo usan.
Interpreter (Interprete): Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
Template Method (Método Plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos. Permite que las subclases redefinan ciertos pasos del algoritmo sin cambiar su estructura.
Visitor (Visitante): Representa una operación sobre los elementos de una estructura de objetos. Permite definir una nueva operación sin cambiar las clases de los elementos sobre los que opera.

Los patrones GRAS o GRASP (General Responsibility Assignment Software Patterns) se dedican fundamentalmente como su nombre lo indica a asignar responsabilidades a los objetos. Como ya conocemos, las principales responsabilidades de cualquier objeto son [20]:

- **Conocer**: Los atributos y sus relaciones con otro objetos.
- **Hacer**: Lo relacionado con las tareas que debe cumplir un objeto.

Los principales Patrones GRASP son [19]:

1. Experto.
2. Creador.
3. Controlador.
4. Alta Cohesión.
5. Bajo Acoplamiento.
6. No hables con extraños

Experto: Asigna responsabilidades al experto de la información. Una clase tiene la información necesaria para llevar a cabo sus responsabilidades. Posibilita el encapsulamiento de la información, y por ende el bajo acoplamiento. Permite el comportamiento distribuido entre las clases.

Creador: Lo que define este patrón es que una instancia de un objeto la tiene que crear el objeto que tiene la información para ello, esto significa que si un objeto A utiliza específicamente otro B, o si B forma parte de A, o si A almacena o contiene B, o si simplemente A tiene la información necesaria para crear B, entonces A es el perfecto creador de B. Este patrón soporta el bajo acoplamiento.

Controlador: Es un evento generado por actores externos. Se asocian con operaciones y como respuestas a los eventos del sistema, tal como se relacionan los mensajes y los métodos.

Alta Cohesión: Este patrón permite que se incremente la claridad y facilite la comprensión, que se simplifique el mantenimiento, implica casi siempre bajo acoplamiento y además incrementa reutilización.

Bajo Acoplamiento: Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas, lo que facilita la centralización de actividades (validaciones, seguridad, etc.). Permite una mejor comprensión de las clases aisladas, que sea convenientes para reutilizar y no afecta los cambios en otros componentes.

No hables con extraños: Este patrón plantea a quien debe invocar un método, permitiendo que solamente invoque métodos de sí mismo, de su área de parámetros y a un objeto creado en su propio ámbito.

En el epígrafe 1.8 luego de analizar las características del sistema se determinará la elección de los patrones.

1.4 Lenguajes de Descripción Arquitectónica

Los Lenguajes de Descripción Arquitectónica (ADLs) se remontan a la década de 1970, pero se han comenzado a desarrollar con esta definición a partir de 1992 o 1993, poco después de fundada la propia arquitectura de software como especialidad profesional. Este tipo de lenguaje es necesario para disponer de abstracciones útiles, para modelar sistemas complejos desde un punto de vista arquitectónico; a la vez que deben permitir un nivel de detalle suficiente como para describir propiedades de interés en dicho sistema.

Los ADLs son un lenguaje descriptivo de modelado, que se enfoca en la estructura de alto nivel de la aplicación, antes que en los detalles de implementación de sus módulos concretos, que debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones. Son poco utilizados debido a que se necesita aprender una sintaxis especializada, siendo convenientes, pero aún no han demostrado ser imprescindibles. La presencia de UML (Lenguaje Unificado de Modelado) ha impedido que estos lenguajes no hayan ocupado el lugar que les corresponde.

Las principales características que presentan estos lenguajes son:

- **Composición**: Permiten la representación del sistema como composición de una serie de partes.
- **Configuración**: La descripción de la arquitectura es independiente de la de los componentes que formen parte del sistema.
- **Abstracción**: Describen los roles o papeles abstractos que juegan los componentes dentro de la arquitectura.
- **Flexibilidad**: Permiten la definición de nuevas formas de interacción entre componentes.
- **Reutilización:** Permiten la reutilización tanto de los componentes como de la propia arquitectura.
- **Heterogeneidad:** Permiten combinar descripciones heterogéneas.

- **Análisis:** Permiten diversas formas de análisis de la arquitectura y de los sistemas desarrollados a partir de ella.

Existen varios lenguajes que sí son considerados ADLs, ejemplos de ellos son:

Acme – Armani: Es una lenguaje de intercambio de arquitectura que es capaz de soportar el mapeo de especificaciones arquitectónicas. Se considera un lenguaje de descripción arquitectónica de segunda generación, es decir, un metalenguaje, que no es más que una lengua franca para el entendimiento de 2 o más ADLs, llegando a soportar 4 tipos de arquitecturas como son: la estructura, las propiedades de interés, las restricciones y los tipos y estilos.

ADML (Architecture Description Markup Language): Constituye un intento de estandarizar la descripción de arquitecturas en base XML, agrega a los ADLs una forma de representación basada en estándares establecidos en la industria, de modo que ésta puede ser leída por cualquier parser de XML. Permite definir vínculos con objetos externos a la arquitectura, así como interactuar con diversos repositorios.

Jacal: Este lenguaje permite visualizar una simulación de cómo se comportará en la práctica un sistema basado en la arquitectura que se ha representado, puede ser utilizado para expresar arquitecturas de distintos estilos [18].

Como resultado del estudio, y a pesar de las ventajas y utilidades que ofrecen los ADLs, no se propone la utilización de ellos por el esfuerzo que representaría capacitar personas en su uso, cuando existen otras formas de cumplir sus objetivos.

1.5 Lenguaje Unificado de Modelado (UML)

UML es un lenguaje de modelado que proporciona un vocabulario y unas reglas para permitir una comunicación entre elementos de un software y se centra en la representación gráfica para visualizar, construir y documentar los artefactos de un sistema, proporcionando una forma estándar de representar los planos de un sistema y comprende tanto elementos conceptuales, como los procesos de negocio y las funciones del sistema en cuanto a elementos concretos, como las clases escritas de un lenguaje de programación específico, esquemas de base de datos y componentes de software reutilizables.

Se crea con el objetivo de comunicar las ideas a otros desarrolladores y para servir de apoyo en los procesos de análisis de un problema; que sea independiente del lenguaje de programación, de forma tal que los diseños realizados se puedan implementar en cualquier lenguaje que soporte las posibilidades de UML.

Este lenguaje sirve de modelo completo de sistemas complejos, tanto en el diseño de los sistemas de software, como para la arquitectura de software donde se ejecuten. Teniendo en cuenta las características de este lenguaje, se consideran como principales ventajas [20]:

- Presenta mayor rigor en la especificación del sistema.
- Posibilita realizar una verificación y validación del modelo realizado.
- Permite que el modelado y el código estén actualizados, con lo que se puede mantener la visión en el diseño de más alto nivel de la estructura de un proyecto.
- Es un lenguaje consolidado y fácil de aprender.

1.6 Metodología, Herramientas y tecnologías

A continuación se analiza la metodología, herramientas y tecnologías ya definidas en la versión anterior.

Metodología de desarrollo

Una metodología de desarrollo de software es un conjunto de pasos y procedimientos que deben seguirse para desarrollar software, no es más que una colección de documentación formal, referente a los procesos, las políticas y los procedimientos que intervienen en el desarrollo del software.

Proceso unificado abierto (OpenUp/Basic)

OpenUP/Basic es un proceso de desarrollo de software de código abierto, mínimo, completo y extensible. Está dirigido a la gestión y el desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. Es muy útil para equipos de desarrollo pequeños y que le dan valor a la colaboración y a las necesidades de los

stakeholder. Esta metodología procura un equilibrio entre las necesidades de los involucrados con los resultados del proyecto y los costos técnicos.

Este proceso está organizado dentro de cuatro áreas principales de contenido: Comunicación y Colaboración, Intención, Solución y Administración.

Se caracteriza por cuatro principios básicos que se soportan mutuamente:

- Colaboración para unificar intereses y compartir conocimientos.
- Equilibrio de prioridades competentes a maximizar el valor de los involucrados con el resultado del proyecto.
- Enfoque en la articulación de la arquitectura.
- Desarrollo continuo para obtener realimentación y realizar las mejoras respectivas. OpenUP/Basic se centra en articular la arquitectura para facilitar la colaboración técnica, reducir el riesgo y minimizar el sobreesfuerzo de desarrollo.

Herramienta CASE: Visual Paradigm

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

Visual Paradigm es fácil de usar y soporta la última notación UML, ingeniería inversa, generación de código, importación desde Rational Rose, exportación/importación XML, generador de informes, editor de figuras, integración con Microsoft Visio, plugin, integración IDE con Visual Studio, Eclipse, NetBeans y otros. Entre sus características se incluyen el modelado colaborativo con CVS y Subversion, interoperabilidad con modelos UML2 a través de XMLI [20].

Lenguaje de programación: Java

Java fue diseñado en 1990 por James Goslin de Sun Microsystems, como software para dispositivos electrónicos de consumo. Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable. Proporciona un conjunto de clases potente y flexible. Pone al alcance de cualquiera la utilización de aplicaciones que se pueden incluir directamente en páginas Web. Aporta a la Web una interactividad que se había buscado durante mucho tiempo entre usuario y aplicación.

Es multiplataforma lo que permite que el mismo código java que funciona en un sistema operativo, funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual java. Gracias al API de java podemos ampliar el lenguaje para que sea capaz de: comunicarse con equipos mediante red, acceder a bases de datos, crear páginas HTML dinámicas, crear aplicaciones visuales al estilo Windows.
Para el desarrollo de la plataforma se seleccionó el lenguaje de programación Java por ser multiplataforma y orientado a objetos, siendo esta una característica imprescindible que se requiere para la aplicación. La existencia de una primera versión programada en Java, influyó en la selección permitiendo de esta forma la reutilización de gran cantidad de código.

Entorno Integrado de Desarrollo (IDE): NetBeans

En su núcleo, el NetBeans IDE es una herramienta de desarrollo para Java, empleando tecnología Java pura, por lo que se ejecuta en cualquier parte donde se ejecute Java. Aparte de la filosofía de distribución y desarrollo que hay tras NetBeans, el IDE ofrece a los desarrolladores numerosas ventajas en la creación de nuevas aplicaciones multiplataforma. En una era en la cual la arquitectura orientada a servicios (SOA) demanda funciones flexiblemente conjuntas que se dirigen a procesos específicos del negocio; permite la integración de múltiples herramientas y protocolos que da motivos para la migración y brinda facilidad de empleo a lo largo de todo el ciclo de vida del desarrollo.

Gestores de Base de Datos: PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, distribuido bajo licencia BSD y con su código fuente disponible libremente. Es el sistema de gestión de bases de datos de código abierto más potente del mercado. Funciona en todos los sistemas operativos importantes, incluyendo Linux, UNIX y Windows. Tiene soporte total para foreign keys, joins, views, triggers, y stored procedures (procedimientos almacenados) en múltiples lenguajes. Incluye la mayoría de los tipos de datos como son INTEGER, NUMERIC, BOOLEAN, CHAR, VARCHAR, DATE e INTERVAL.
Entre sus principales características se encuentran:

- **DBMS Objeto-Relacional:** Aproxima los datos a un modelo objeto-relacional, y es capaz de manejar complejas rutinas y reglas. Ejemplos de su avanzada funcionalidad son consultas SQL declarativas, control de concurrencia multi-versión, soporte multi-usuario, transacciones, optimización de consultas, herencia, y arreglos.
- **Altamente Extensible:** Soporta operadores, funcionales métodos de acceso y tipos de datos definidos por el usuario.

- **Soporte SQL Comprensivo:** Soporta la especificación SQL99 e incluye características avanzadas tales como las uniones SQL92.
- **Integridad Referencial:** Soporta integridad referencial, la cual es utilizada para garantizar la validez de los datos de la base de datos.
- **API Flexible:** La flexibilidad del API de PostgreSQL ha permitido a los vendedores proporcionar soporte al desarrollo fácilmente para el RDBMS PostgreSQL. Estas interfaces incluyen Object Pascal, Python, Perl, PHP, ODBC, Java/JDBC, Ruby, TCL, C/C++, y Pike.
- **Lenguajes Procedurales:** Tiene soporte para lenguajes procedurales internos, incluyendo un lenguaje nativo denominado PL/pgSQL. Este lenguaje es comparable al lenguaje procedural de Oracle, PL/SQL. Otra ventaja de PostgreSQL es su habilidad para usar Perl, Python, o TCL como lenguaje procedural embebido.
- **Cliente/Servidor:** Utiliza una arquitectura proceso-por-usuario cliente/servidor. Hay un proceso maestro que se ramifica para proporcionar conexiones adicionales para cada cliente que intente conectar a PostgreSQL.

Sistema de Control de Versiones: Subversion (SVN)

Es un sistema de control de versiones que se ha popularizado bastante dentro de la comunidad de desarrolladores de software libre. Está preparado para funcionar en red, y se distribuye bajo una licencia libre de tipo Apache. Surge con la intención de sustituir y mejorar al conocido CVS, que a pesar de sus características, constituyó el estándar de los sistemas de gestión de versiones en el ámbito del software libre. SVN mantiene las ideas fundamentales de CVS pero suple sus carencias y evita sus errores.

Entre las principales características de SVN se encuentran:

- Mantiene versiones no sólo de archivos, sino también de directorios.
- Se mantienen versiones de los metadatos asociados a los directorios.
- Además de los cambios en el contenido de los documentos, se mantiene la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
- Atomicidad de las actualizaciones: Una lista de cambios constituye una única transacción o actualización del repositorio. Esta característica minimiza el riesgo de que aparezcan inconsistencias entre distintas partes del repositorio.
- Posibilidad de elegir el protocolo de red: Además de un protocolo propio (svn), puede trabajar sobre http (o https) mediante las extensiones WebDAV (más conocido como DAV), este es un protocolo que amplía las posibilidades del HTTP/1.1 añadiendo nuevos métodos y cabeceras.

La capacidad de funcionar con un protocolo tan universal como el http simplifica la implantación (cualquier infraestructura de red actual soporta dicho protocolo) y universaliza las posibilidades de acceso (si se quiere, puede utilizarse a través de Internet).

- Soporte tanto de ficheros de texto como de binarios.
- Mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.

Servidor de Aplicaciones: Tomcat

Tomcat (también llamado Jakarta Tomcat o Apache Tomcat) funciona como un contenedor de servlets (objetos que corren dentro del contexto de un servidor de aplicaciones y extienden su funcionalidad), desarrollado bajo el proyecto Jakarta en la Apache Software Foundation. Se considera un servidor de aplicaciones e implementa las especificaciones de los servlets y de Java Server Pages (JSP) de Sun Microsystems. Tomcat funciona con cualquier servidor web con soporte para servlets y JSPs.

Este servidor opera de tal manera en entornos de desarrollo poco exigentes en términos de velocidad y de manejo de transacciones. Dado que fue escrito en Java, funciona en cualquier sistema operativo que disponga de la máquina virtual. Lo desarrollan y lo mantienen miembros de la Apache Software Foundation y voluntarios independientes. Los usuarios disponen de libre acceso a su código fuente y a su forma binaria en los términos establecidos en la Apache Software Licence.

Después del estudio realizado anteriormente se decidió seleccionar Tomcat como servidor de aplicaciones, pues entre otras características funciona como servidor web por sí mismo. Tomcat consume menos recursos que otros servidores de aplicaciones, además de ser gratis, cuenta con un gran número de usuarios y soporte en la comunidad mundial.

Framework: Hibernate

Hibernate es una herramienta para la plataforma Java que facilita el mapeo de atributos entre una base de datos relacional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones. Realiza el mapeo entre el mundo orientado a objetos de las aplicaciones y el mundo entidad-relación de las bases de datos en entornos Java. El término utilizado es ORM (object/relational mapping) y 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 [20].

Este framework ha conseguido en un tiempo record, una excelente reputación en la comunidad de desarrollo posicionándose claramente como el producto Open Source líder en este campo gracias a sus

prestaciones, buena documentación y estabilidad. Entre sus principales características se encuentran [18]:

- Esta herramienta es una clara implementación del patrón DAO. Este patrón permite contar con diversas fuentes de datos, de tal forma que se encapsula la forma de acceder a estos. Hibernate crea una capa separada que se ocupa del acceso a datos con total independencia del gestor y la base de datos, dando la oportunidad de trabajar con varios gestores y bases de datos dentro de la misma aplicación sin que esto cree ningún conflicto en el modelo de objetos.
- Presenta un mecanismo persistente totalmente transparente, donde los objetos desconocen su capacidad de persistir, pueden tener lógica de aplicación, son de fácil manejo y pueden ser usados para transportar los datos a cualquier capa de la aplicación.

Plataforma: JDK

JDK 1.6 (Java Development Kit) Se trata de un conjunto de herramientas (programas y librerías) que permiten desarrollar (compilar, ejecutar, generar documentación, etc.) programas en lenguaje Java [19].

El kit de desarrollo de Java consiste en un compilador y herramientas de desarrollo para crear tanto programas independientes como applets. Desarrollado por Sun Microsystems, los creadores de Java. Esta nueva versión corrige fallos de versiones anteriores e incluye mejoras en el rendimiento general [20].

1.7 El Rol Arquitecto de Software

El rol del arquitecto de software es el de liderar el proceso de arquitectura, coordinar las actividades técnicas y producir los artefactos necesarios como: Documento de descripción de arquitectura y los modelos y prototipos de arquitectura. Se encarga de la definición y documentación de la arquitectura que guiará el desarrollo, y de la continua refinación de la misma en cada iteración; debe construir cualquier prototipo necesario para probar aspectos riesgosos desde el punto de vista técnico del proyecto; definirá los lineamientos generales del diseño y la implementación. Es responsable de diseñar las vistas que definen los requisitos, el diseño, la implementación y el despliegue del sistema.

Figura 1. Rol de Arquitecto de Software.

En la figura 1 se muestra como el arquitecto es quien realiza el análisis y refinamiento de la arquitectura, además de ser el responsable del cuaderno de arquitectura.

El cuaderno de arquitectura describe el contexto de desarrollo del software. Contiene las decisiones, los fundamentos, las hipótesis, las explicaciones y las consecuencias de la formación de la arquitectura. Tiene como propósito lograr la integridad y comprensibilidad del sistema. Orienta a los desarrolladores que van a construir el sistema, constituye un artefacto crítico utilizado para tomar decisiones arquitectónicas, las cuales son explicadas a los desarrolladores. En general guía a los desarrolladores en la construcción del sistema, pero no contiene información de diseño.

El arquitecto de software también debe participar en otras actividades, como son: el desarrollo del documento Visión, captura y descripción de requisitos, realización del plan de proyecto y plan de iteración, el diseño de la solución y la evaluación de los resultados.

1.8 Características del Sistema

BioSyS agrupará un conjunto de operaciones que son necesarias para la investigación de los Sistemas Biológicos. El sistema realiza un estudio de estos sistemas y deben permitir de manera integrada la realización de diferentes operaciones que representan las funcionalidades o módulos. BioSyS se encuentra estructurado de forma tal que cada módulo es representado como un componente que representa una funcionalidad definida.

El Front-End permite a los investigadores adicionar o eliminar plugins, representando estos las funcionalidades de la plataforma.

La simulación permite a los investigadores simular nuevos modelos matemáticos o algunos ya existentes y editar la configuración de las simulaciones.

El editor de ecuaciones permite a los investigadores importar y exportar los Sistemas de Ecuaciones Diferenciales (SED), además de gestionar la edición de estos SED y analizar la dimensionalidad de estos.

La modelación permitirá gestionar las especies y el comportamiento de estas en un sistema biológico, gestionar la reacción química de un modelo matemático y el modificador, permitiendo que los investigadores puedan insertar y eliminar una especie como modificador de una reacción química de un modelo matemático (MM).

El graficador permite simular los distintos modelos matemáticos en dos y tres dimensiones, además de gestionar la gráfica permitiendo que los investigadores puedan cambiar las propiedades, los colores y las escalas.

El análisis permite a los investigadores realizar la estimación de parámetros, el análisis de los clusters de las simulaciones, las salidas gráficas, el análisis mediante un conjunto de reglas definidas y clasificar las simulaciones a partir de modelos generados por simulaciones anteriormente clasificadas.

La estimación de parámetros es un tipo de análisis que se utiliza para el análisis de los sistemas biológicos, que permite al investigador gestionar las estimaciones de los diferentes parámetros del modelo, transformar todos los datos experimentales y gestionar el estudio experimental de los diferentes sistemas.

El sistema cuenta con una Base de Datos que permite guardar los resultados, pero esta no representa un módulo del sistema.

Con estas características BioSyS constituirá una potente herramienta para los investigadores y deberá permitir a estos realizar la funcionalidad que necesite y guardar los resultados de las investigaciones realizadas, permitiendo la interacción entre cada parte del sistema que represente una funcionalidad.

Para la arquitectura propuesta no se utiliza ningún patrón arquitectónico ya que esta consiste en integrar todos los componentes del sistema, lo que constituye un alto nivel de abstracción. Para el diseño de cada uno de los módulos que conforman el sistema se debe seguir el patrón en Capas y solo en los casos en que sea necesario violar la capa de control se seguirá el patrón MVC. Como patrones de diseño se propone utilizar los patrones GRASP para la asignación de responsabilidades, además del Facade que define una interfaz de alto nivel que hace que el subsistema Front-End sea más fácil de usar.

1.9 Conclusiones

En este capítulo se ha realizado un análisis de los temas relacionados con la investigación, tecnologías, tendencias y metodologías más utilizadas en el mundo de la informática, con el objetivo de proporcionar las características necesarias para el diseño de la arquitectura del sistema. Después de haber realizado un estudio de lo anteriormente planteado se seleccionó dentro de la familia de estilos Llamada y retorno, la Arquitectura Basada en Componentes debido a las ventajas que representa permitiendo alcanzar un mayor nivel de reutilización de software y que las pruebas sean ejecutadas probando cada uno de los componentes. Para la arquitectura propuesta no se utiliza ningún patrón arquitectónico ya que esta consiste en integrar todos los componentes del sistema, lo que constituye un alto nivel de abstracción. Para el diseño de cada uno de los módulos que conforman el sistema se debe seguir el patrón en Capas y solo en los casos en que sea necesario violar la capa de control se seguirá el patrón MVC. Como patrones de diseño se propone utilizar los patrones GRASP para la asignación de responsabilidades, además del Facade que define una interfaz de alto nivel que hace que el subsistema Front-End sea más fácil de usar. Para describir la arquitectura se utiliza UML como lenguaje de modelado y Visual Paradigm como herramienta CASE para el diseño de la arquitectura. La metodología de desarrollo utilizada es OpenUp/Basic. El lenguaje de programación escogido es Java, utilizando la plataforma JDK. Se utiliza Netbeans como Entorno Integrado de Desarrollo y Subversión como Sistema de control de Versiones. El framework Hibernate se utiliza por los beneficios que presenta y por ser una clara implementación del patrón DAO. El SGBD escogido es PostgreSQL y como servidor de aplicaciones se utilizará el Tomcat.

CAPÍTULO 2

2.1 INTRODUCCIÓN

En este capítulo se hace una propuesta de solución al problema planteado. Se diseñan las vistas arquitectónicas y se hace una descripción de la arquitectura para la plataforma, definiendo cómo debe hacerse la integración y comunicación entre los módulos.

2.2 Representación Arquitectónica

En la Figura 2 se muestra la forma en que se propone organizar el sistema, para ofrecer una mejor idea y facilitar la comprensión de la arquitectura propuesta.

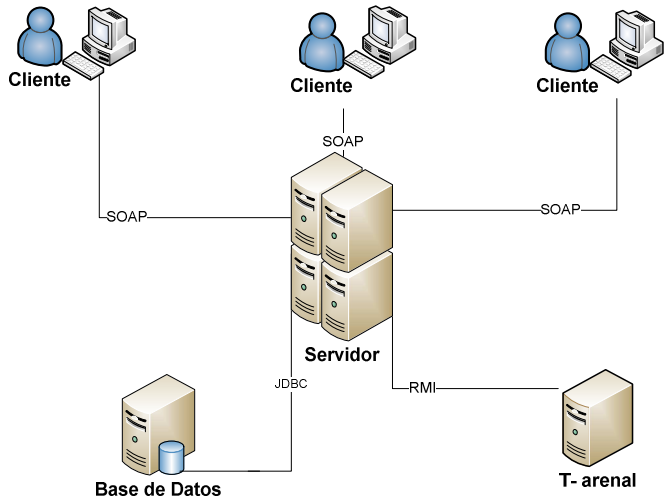

Figura 2. Representación arquitectónica.

Como se muestra en la figura anterior, el sistema contará con un servidor donde serán gestionadas las peticiones realizadas por el cliente. Esas peticiones serán realizadas por las computadoras (PC) clientes a través del protocolo SOAP. Desde el servidor se realizará la petición a la Base de Datos (BD), verificando si los datos se encuentran o no. Una vez obtenida la información, el servidor T-arenal enviará el trabajo a las diferentes PC donde se realizará el cálculo distribuido. Después de realizadas esas operaciones el servidor elabora la respuesta y la envía a las PC clientes, este las visualiza y así termina el proceso.

Cliente

Como se ilustra en la figura pueden existir varios clientes, que contarán con una aplicación de escritorio para modelar sistemas biológicos.

Servidor de Base de Datos

Contendrá la BD donde se encontrará la información almacenada referente a los datos del sistema.

Servidor T-arenal

Tiene a su disposición una red de máquinas que permite el cálculo distribuido.

Servidor

Constituirá la parte fundamental dentro de la arquitectura, ya que aquí es donde se gestionan las solicitudes y respuestas de los clientes. El funcionamiento de este será como se muestra en la figura 3.

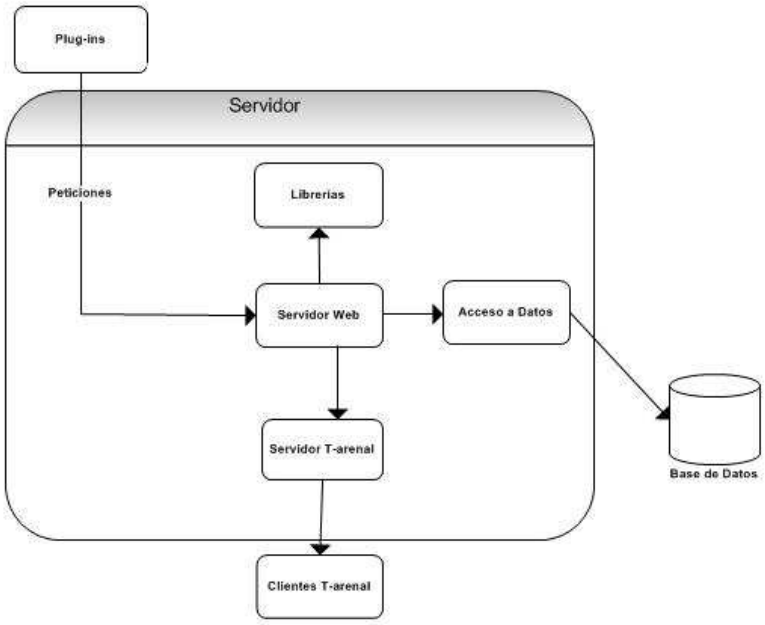

Figura 3. Funcionamiento del Servidor.

Las peticiones realizadas por los clientes serán manejadas a través del Servidor Web. Estos Servicios haciendo uso de las librerías se conecta al servidor T-arenal enviándole las solicitudes de trabajo. En este servidor se gestiona la conexión a la base de datos, extrayendo los datos necesarios y distribuyendo el trabajo. El Servidor y las PCs distribuidas se relacionan entre sí mediante el Servidor T-arenal que se encuentra en el servidor y el Cliente T-arenal que se encuentra en las PCs distribuidas. Para realizar esta

distribución el Servidor T-arenal envía los datos a las PCs distribuidas para realizar el trabajo. Cuando este servidor obtiene la solución se encarga de completar la respuesta y a través de los servicios web es enviada al usuario.

2.3 Metas y Restricciones

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Son las características que hacen al producto atractivo, usable, rápido o confiable.

Dentro de las metas y restricciones arquitectónicas para la plataforma, se encuentran los requisitos no funcionales, estos se enuncian a continuación.

Usabilidad

- El sistema podrá ser usado por aquellos usuarios que posean conocimientos básicos en el campo de la modelación de sistemas biológicos.
- El sistema le ofrecerá al investigador la posibilidad de modelar un sistema biológico, editar las ecuaciones del modelo, variar sus parámetros, realizar simulaciones distribuidas y analizar los resultados de las simulaciones, el diseño de la aplicación está centrado en ello y en específico en el diseño de las interfaces que harán posible el intercambio de datos de modo que resulte fácil el uso del sistema.
- El sistema le ofrecerá al investigador la posibilidad de realizar simulaciones distribuidas o locales, en este sentido se centra el diseño de la aplicación y en específico de las interfaces que harán posible el intercambio de datos.
- La aplicación dará la posibilidad de almacenar los resultados obtenidos a partir de la modelación y simulación de sistemas biológicos, mediante interfaces que permitan el intercambio de datos de manera fácil para el usuario.
- Se debe lograr un diseño adaptable, con la capacidad de poder soportar funcionalidades adicionales o modificar las funcionalidades existentes sin impactar el resto de los requerimientos contemplados en el sistema.

Fiabilidad

Tiempo medio de reparación: La reparación del sistema en caso de surgir fallas en el mismo debe realizarse en el menor tiempo posible, poniendo todos los esfuerzos en función de que no supere las 72 horas.

- Cada uno de los módulos del sistema debe proporcionar funcionalidades propias que satisfagan las necesidades establecidas y actuar bajo determinadas condiciones especificadas.
- Confidencialidad: Se requiere de usuario y contraseña para poder acceder a la información de la base de datos, también para poder acceder al T-arenal.
- Disponibilidad: En caso de tener el usuario y la contraseña se le garantiza poder acceder a la información almacenada en todo momento
- De ocurrir algún fallo en las conexiones para realizar los cálculos distribuidos, el sistema está diseñado para detectar este error y enviar la parte del trabajo que no se concluyó a otra máquina que esté disponible para que esta la continúe.
- Integridad: la información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes, de la misma forma será considerada igual a la fuente o autoridad de los datos. Pueden incluir también mecanismos de chequeo de integridad y realización de auditorías.

Soporte

- El sistema estará bien documentado de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.
- El sistema permite implementar cambios, ya sea cualquier corrección, mejora o adaptación del software, por ejemplo: adicionar un nuevo módulo, sin efectos inesperados.
- El sistema debe funcionar en cualquier sistema operativo sobre el cual se haya instalado la máquina virtual de Java 1.5 o superior.
- Instalación: La instalación del sistema debe caracterizarse por su facilidad, claridad y sencillez.
- Portabilidad: El sistema debe ser multiplataforma (ser capaz o caracterizarse por poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas).
- El sistema debe permitir la interacción con los demás módulos que componen la plataforma.
- Se realizarán distintas pruebas al software una vez concluido para comprobar su funcionalidad.
- Terminado el software se prestarán los servicios de instalación y configuración de la aplicación.
- Se prestarán servicios de mantenimiento del software.

Requisitos de Software

- Para el desarrollo de BioSyS se ha utilizado el lenguaje de programación Java lo que implica que es multiplataforma. El mismo puede ejecutarse sobre cualquier sistema operativo.
- Máquina virtual de Java 1.5 o superior.

- Gestor de Base de Datos PostgreSQL.
- Matlab (si se quiere utilizar el mismo como asistente matemático para la resolución de los sistemas de ecuaciones diferenciales).
- Plataforma de Cálculo Distribuido T-arenal (si se quiere hacer uso del procesamiento distribuido).

Requisitos de Hardware

- Se requiere como mínimo 256 MB de memoria RAM.
- Procesadores Pentium IV.
- Disco duro de 40 GB (puede variar dependiendo de la cantidad de información a almacenar).
- La Base de Datos del sistema permitirá como promedio más de 20 conexiones simultáneas y 10000 conexiones de inserciones simultáneas.

Restricciones en el diseño y la implementación

- Lenguaje de programación Java.
- IDE de desarrollo NetBeans.
- Herramienta CASE Visual Paradigm.
- Gestor de bases de datos PostgreSQL.
- Para el desarrollo del sistema se va a usar el Framework para Java Hibernate, que sigue de forma estricta el patrón DAO, se hará uso del mismo para el acceso a los datos.
- El diseño de cada uno de los módulos que conforman el sistema debe seguir el patrón en Capas y solo en los casos en que sea necesario violar la capa de control se seguirá el patrón MVC.
- Para la implementación de esta versión se reutilizará código implementado en la anterior versión del mismo.
- El Sistema de Cómputo Distribuido en Java como componente de proveedores externos, debe estar disponible siempre que sea solicitado su uso.
- La implementación del código del sistema tiene que regirse por la guía de estilo de codificación elaborada por el grupo de arquitectura de la facultad, dicha guía contiene un grupo de normas básicas a utilizar cuando se programa en Java y estas deben ser adoptadas.

Requisitos de Licencia

El sistema se utilizará primeramente en el Centro de Inmunología Molecular y se piensa que posteriormente en los demás centros del Polo Científico, en el caso que luego se decida comercializar en el extranjero entonces deberá contar con una licencia. Se deberán tener en cuenta los requisitos de Legales que establece la UCI.

Requisitos Legales, de Derecho de Autor y otros.

Los derechos de autor serán registrados entre la UCI y el CIM.

2.3 Diseño de las Vistas Arquitectónicas

Es muy complejo capturar la arquitectura de software en un sólo modelo (o diagrama). Para manejar esta complejidad se representan diferentes aspectos y características de la arquitectura en múltiples vistas. Una vista es “una presentación de un modelo, la cual es una descripción completa de un sistema desde una particular perspectiva” (Kruchten, 1995). El modelo más aceptado a la hora de establecer las vistas necesarias para describir una arquitectura de software es el modelo 4+1 vistas (Kruchten, 1995) [19].

Este modelo define 4 vistas principales:

- Vista Lógica (Logical View): modelo de objetos, clases, entidad – relación.
- Vista de Proceso (Process View): modelo de concurrencia y sincronización.
- Vista de Desarrollo (Development View): organización estática del software en su entorno de desarrollo (librerías, componentes, .ear, .jar).
- Vista Física (Physical View): modelo de correspondencia software - hardware (aspectos de distribución en máquinas, por ejemplo).

Y una vista más, la "+1", que se muestra y traza en cada una de las anteriores y que está formada por las necesidades funcionales que cubre el sistema, y que en ocasiones se identifica como vista de "casos de uso". De donde se deduce que según este modelo, la arquitectura es en realidad evolucionada desde escenarios.

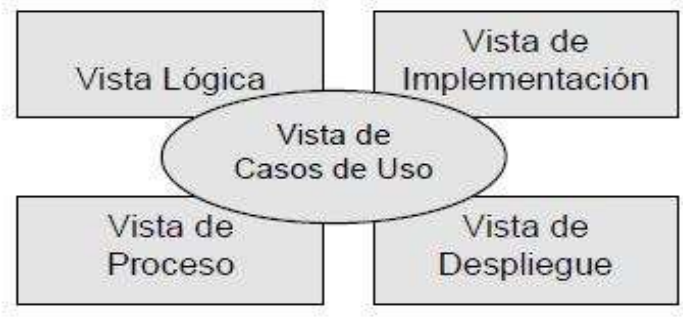

Figura 4. Modelo 4+1 Vistas.

2.3.1 Vista de Caso de Uso

Esta vista describe el proceso de negocio más significativo y el modelo del dominio. Presenta los actores y los casos de uso para el sistema. Es decir, esta vista presenta la percepción que tiene el usuario de las funcionalidades del sistema. Si el sistema se hace extenso entonces se debe organizar en paquetes, lo cual facilitaría la comprensión de la vista.

Actores: Un actor representa un conjunto coherente de roles que los usuarios de los casos de uso juegan cuando interactúan con el sistema.

Casos de Uso (CU): Un caso de uso describe qué hace un sistema (o subsistema, clase o interfaz) pero no especifica cómo lo hace. Describen un conjunto de secuencias de acciones, incluyendo variaciones que un sistema lleva a cabo y que conduce a un resultado observable de interés para un determinado actor.

Paquete: Un paquete es el elemento de organización básica de un modelo de sistema UML. Es un mecanismo general para dividir modelos y agrupar elementos.

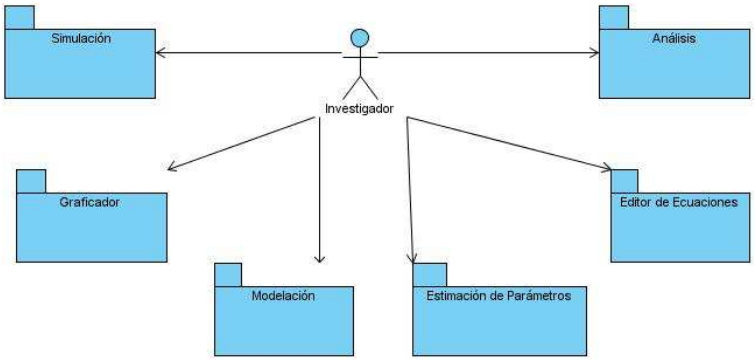

Figura 4. Vista de Casos de Uso.

En la figura 4 se muestra que el sistema cuenta con un actor que es el Investigador. Este representa el usuario que interactúa con el sistema.

La vista se organizó en paquetes para simplificar su comprensión. En la elaboración de estos se tuvo en cuenta la estrecha relación entre los casos de uso, pues responden a funcionalidades altamente relacionadas.

A continuación se mostrarán los casos de uso arquitectónicamente significativos correspondientes a cada paquete, los cuales fueron seleccionados porque el proceso que describen es de suma importancia para el sistema.

2.3.1.1 Vista de CU del Paquete Simulación

Simular Modelo Matemático: Permite simular un modelo matemático.

Editar Configuración para Simular: Permite editar la configuración para simular algún modelo matemático dependiendo de las características.

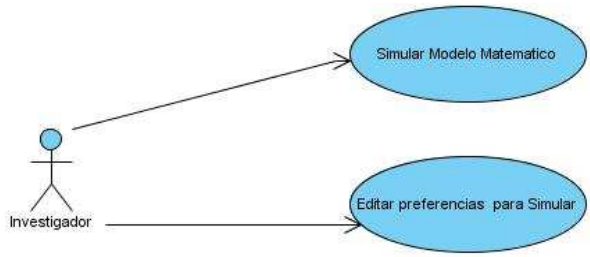

Figura 6. Vista de CU del Paquete Simulación.

2.3.1.2 Vista de CU del Paquete Editor de Ecuaciones

Importar SED (Sistema de Ecuaciones diferenciales): Permite salvar los sistemas de ecuaciones diferenciales.

Exportar SED: Permite al investigador cargar un sistema de ecuaciones diferenciales de diferentes formatos.

Gestionar Edición SED: Permite al investigador gestionar las diferentes operaciones los sistemas de ecuaciones diferenciales.

Analizar dimensionalidad: Permite al investigador el análisis de las dimensiones de los SED viendo si son correctas o no.

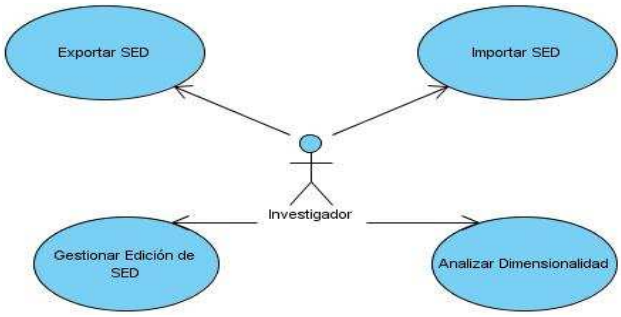

Figura 7. Vista de CU del Paquete Editor de Ecuaciones.

2.3.1.3 Vista de CU del Paquete Modelación

Gestionar Especies: Permite al investigador gestionar las especies de un modelo, como son: insertar, modificar y eliminar especies.

Gestionar compartimiento: Permite al investigador gestionar los compartimientos del modelo matemático necesario como son: insertar, modificar y eliminar compartimientos.

Gestionar Reacción Química: Permite que el investigador pueda gestionar las reacciones químicas como son insertar, modificar y eliminar la reacción química de un modelo matemático.

Gestionar Modificador: Permite al investigador insertar y eliminar una especie como modificador de una reacción química de un modelo matemático.

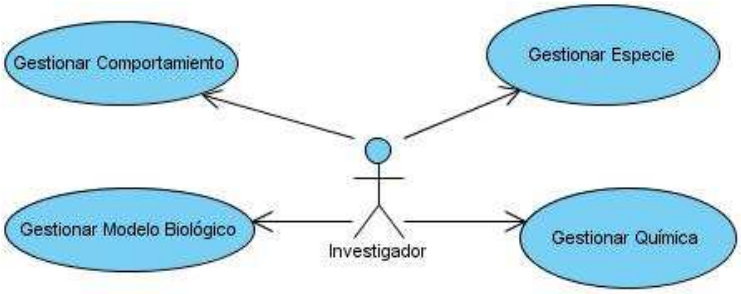

Figura 8. Vista de CU del Paquete Modelación.

2.3.1.4 Vista de CU del Paquete Estimación de Parámetros

Gestionar Estimación: Permite al investigador gestionar las estimaciones de los diferentes parámetros.

Transformar Datos Experimentales: Permite al investigador transformar todos los datos experimentales.

Gestionar Estudio Experimental: Permite al investigador gestionar el estudio experimental de los diferentes sistemas biológicos.

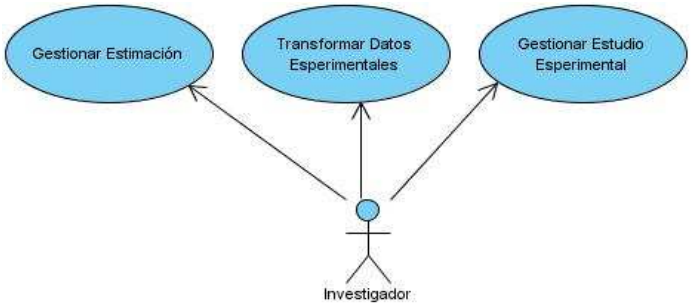

Figura 9. Vista de CU del Paquete Estimación de Parámetros.

2.3.1.5 Vista de CU del Paquete Graficador

Gestionar Simulación en 2D: Permite al investigador ver la simulación del modelo en 2 dimensiones.

Gestionar Simulación en 3D: Permite al investigador ver la simulación del modelo en 3 dimensiones.

Gestionar Gráfica: Permite al investigador cambiar todas las propiedades, colores y escalas de la gráfica.

Editar propiedad de la gráfica: Permite al investigador modificar las propiedades o los valores de la gráfica del modelo.

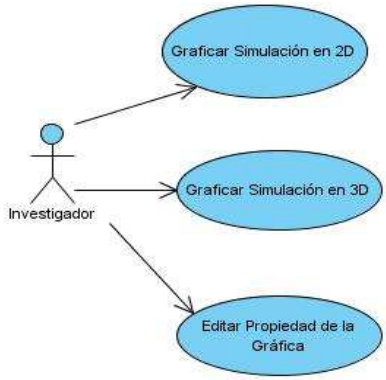

Figura 10. Vista de CU del Paquete Graficador.

2.3.1.6 Vista de CU del Paquete Análisis

Estimación de Parámetros: Permite al investigador estimar los parámetros del modelo matemático.

Realizar Clusters: Permite al investigador el análisis de clúster de las simulaciones de un modelo matemático.

Mostrar Dinámicas: Permite al investigador analizar las salidas gráficas de las simulaciones.

Realizar Análisis por Reglas: Permite al investigador realizar el análisis mediante un conjunto de reglas definidas.

Realizar Clasificación: Clasificar simulaciones a partir de modelos generados de simulaciones previamente clasificadas.

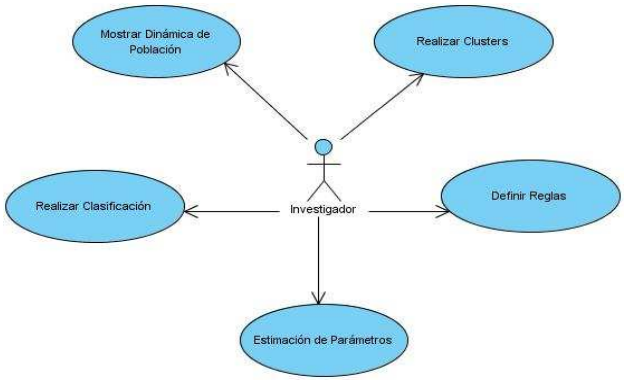

Figura 11. Vista de CU del Paquete Análisis.

2.3.2 Vista Lógica

La vista lógica representa los elementos de diseño (en términos de clases y objetos) más importantes para la arquitectura del sistema. En esta se describen las clases más importantes organizadas en paquetes y subsistemas.

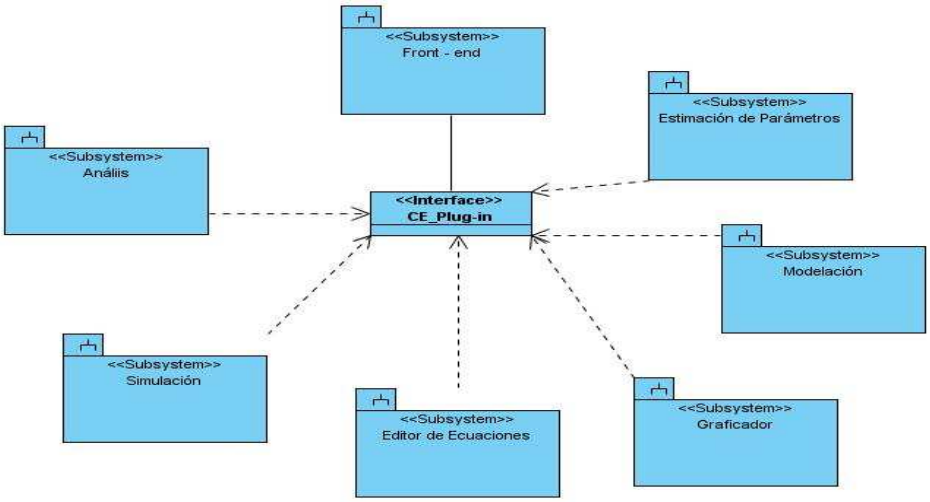

Figura 12. Vista Lógica.

En la figura 12 se muestra el sistema estructurado por subsistemas. Estos subsistemas son la unidad de implementación compuesta por la agrupación de elementos funcionales del sistema. Un subsistema puede contener componentes, clases y otros subsistemas.

Entre los subsistemas se pueden encontrar diferentes tipos de relaciones, dependiendo de si un subsistema utiliza elementos de él mismo o de otro. Cada uno de estos brinda interfaces que serán bien definidas por el arquitecto. Las interfaces definirán operaciones que unos utilizarán y otros brindarán.
La vista lógica está estructurada por subsistemas y cada uno de ellos se detallará a continuación:

Subsistema Front – End

Este subsistema es la interfaz de usuario que permite nuevas facilidades para el usuario final.

Subsistema Análisis

Este subsistema permite la realización de Clusters y de análisis por reglas de los modelos. Realiza la estimación de parámetros, el análisis de estabilidad y mostrar dinámicas de población.

Subsistema Plugins de Simulación

Este subsistema permite simular distintos modelos matemáticos y edita la configuración para simular.

Subsistema Editor de Ecuaciones

Este subsistema permite importar, exportar y gestionar los sistemas de ecuaciones diferenciales, además de analizar la dimensionalidad.

Subsistema Graficador

Este subsistema permite graficar en 2 y 3 dimensiones los modelos, además de gestionar la gráfica y editar las propiedades de la misma.

Subsistema Modelación

Este subsistema permite gestionar los compartimientos, las especies, las reacciones químicas y los modelos biológicos.

Subsistema Estimación de Parámetros

Este subsistema permite gestionar la estimación y el estudio experimental, además de transformar datos experimentales.

Interface CE_Plug-in

Esta interface se encarga de contener las operaciones que permiten visualizar los resultados de cada uno de los subsistemas representados en la figura 12.

2.3.3 Vista de Despliegue

La vista de despliegue es la representación de la arquitectura física mediante nodos interconectados. Estos nodos son elementos hardware sobre los cuales pueden ejecutarse los elementos de software. En la figura 13 se muestra la vista de despliegue.

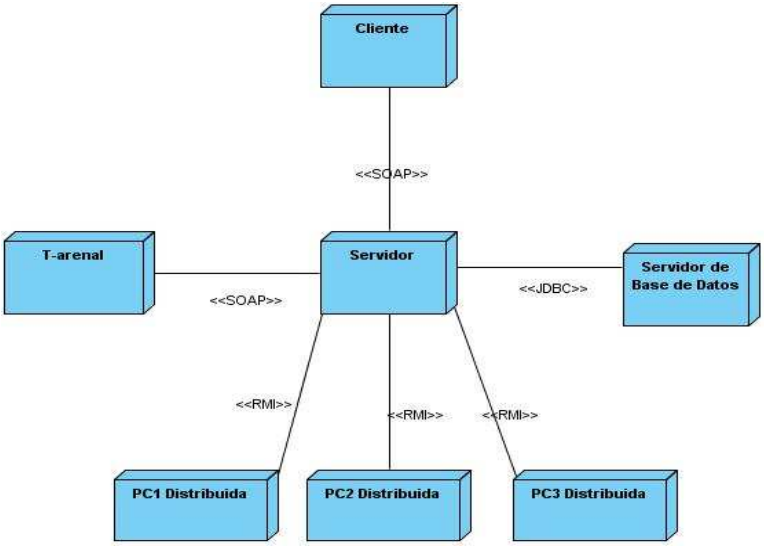

Figura 13. Vista de despliegue.

El cliente realiza las peticiones al servidor de aplicaciones a través del protocolo SOAP. Existirá un servidor de Base de Datos que se encontrará conectado al servidor de aplicaciones mediante el driver JDBC, para realizar las consultas y accesos a la BD. El servidor será el encargado de distribuir las peticiones a las PC que estarán conectadas a él mediante el protocolo RMI, en las que se encontrarán las aplicaciones y funciones que soportará la plataforma. Si las PCs distribuidas para realizar el cálculo no son suficientes, puede hacer uso del servidor T-arenal.

De esta manera se garantizará el funcionamiento de la plataforma y el desarrollo en cualquier entorno investigativo que cuente con acceso a internet o con una red local.

2.3.4 Vista de Procesos

La vista de procesos es un modelo de concurrencia y sincronización que incluye elementos tales como: tareas, hilos de ejecución o procesos. En esta se tratan algunos requisitos no funcionales como: ejecución, disponibilidad, tolerancia a fallos e integridad.

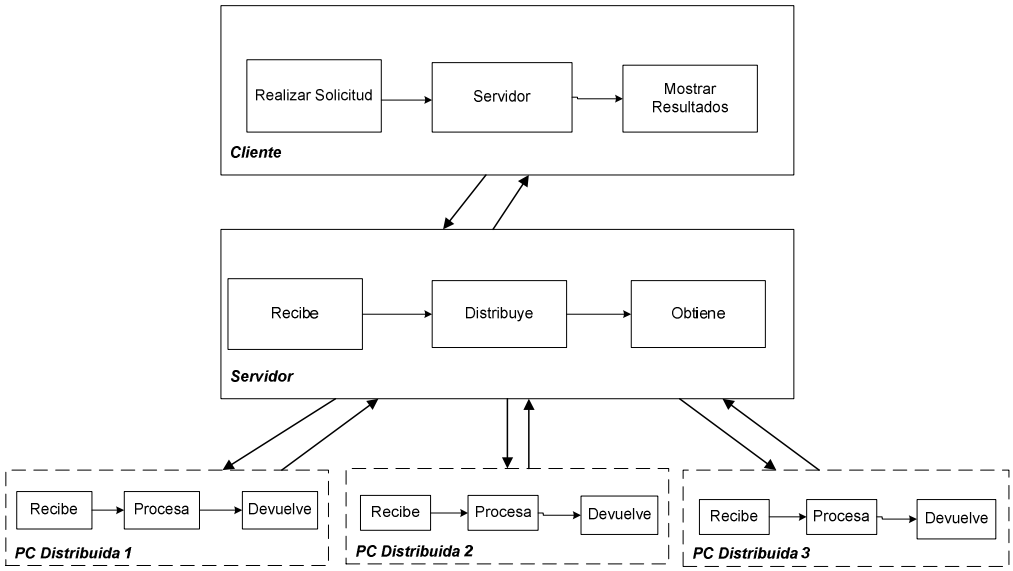

Figura 14. Vista de Procesos.

En esta vista existe un solo hilo de concurrencia ya que el sistema es una aplicación de desktop. El proceso fundamental cuenta con tres actividades. El punto inicial de este proceso comienza con la petición que realiza el cliente al servidor. El servidor obtiene estas peticiones y a través de la clase DataManager mediante el método GenarateWorkUnit se genera la unidad de trabajo para enviarla a las PCs Distribuidas. Estas PCs se reportan al servidor cada cierto tiempo, realizan los cálculos ejecutando la clase Algoritm y mediante el método ProcessUnit se encarga de procesar la unidad de trabajo que se le envió y devuelve los resultados al servidor. Estos resultados son recibidos en el servidor por la clase DataManager mediante el método ProcessResult y son enviadas al cliente para mostrar los resultados.

2.3.5 Vista de Implementación

La vista de implementación se enfoca en la organización de los módulos del software en el entorno de desarrollo. El software es empaquetado mediante librerías, subsistemas, componentes y ficheros. El sistema se estructuró en componentes ejecutables y subsistemas de implementación como se muestra en la figura 15.

Componente: Es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación y un conjunto de interfaces y proporciona la realización de los mismos. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos (ficheros ejecutables, binarios).

Subsistema de Implementación: Es una colección de componentes y otros subsistemas de implementación usados para estructurar el modelo de implementación y dividirlos en pequeñas partes. Los subsistemas se organizan en capas jerárquicas, y cada capa proporciona una interfaz bien definida a sus capas superiores.

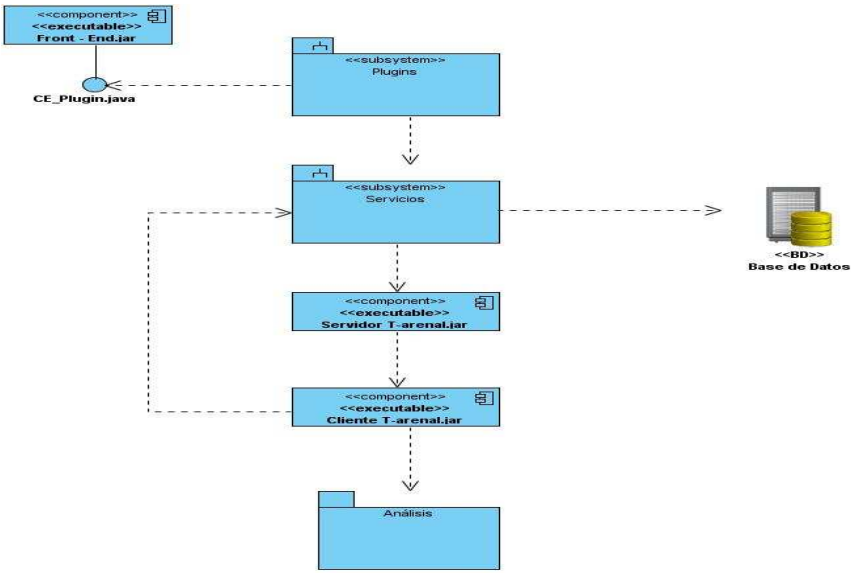

Figura 15. Diagrama de Componentes.

Para la selección de los componentes se siguió el criterio de encapsulamiento según las funcionalidades de la plataforma. Estos componentes son muy importantes porque representan una parte funcional del sistema. El componente Front-End es la interfaz que permite realizar las funcionalidades del sistema. El subsistema Plugins agrupa todos los plugins que serán utilizados en el sistema y se detallarán más adelante. Este subsistema implementa los métodos definidos en la interface CE_Plugin y permite adicionar las funcionalidades. El Servicio de Acceso a Datos que se encuentra dentro del subsistema de Servicios, utiliza la librería Acceso a Datos que se muestra en la figura 19 para conectarse a la Base de Datos. El componente Servidor T-arenal le envía al Cliente T-arenal los datos necesarios para realizar el cálculo distribuido. En el paquete de Análisis se encuentran los diferentes tipos de análisis que se pueden realizar para dichos cálculos. Después de realizados los diferentes cálculos, el Cliente T-arenal

puede guardar los resultados en la Base de Datos mediante el Servicio Acceso a Datos, además va enviando las soluciones al Servidor T-arenal y éste conforma la respuesta final que le será mostrada al cliente.

Para una mejor comprensión de este diagrama se omitió la representación de la librería tarenalLibrary.jar, pero esta librería será utilizada por los Servicios Web para comunicarse con el servidor T-arenal, este se muestra en la figura 19.

En la figura 16, se muestra el subsistema de implementación Servicios que contiene los componentes: Servicio Acceso a Datos, Servicio Simulación y el Paquete de Análisis donde se encuentran: Servicio Clustering, Estimación de Parámetros, Bifurcaciones, Análisis por Reglas y de Estabilidad.

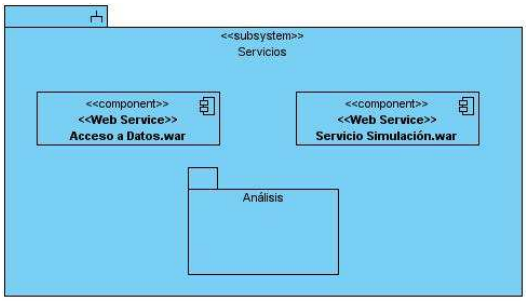

Figura 16. Subsistema de Servicios

En la figura 17 se muestra el subsistema Plugins que contiene los plugins: Editor de Ecuaciones, Modelación, Graficador y el Paquete de Análisis donde se encuentran: Plugin de Análisis de Clustering, de Bifurcaciones, Estimación de Parámetros, de Estabilidad y por Reglas.

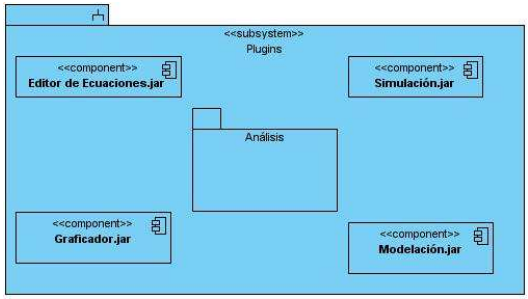

Figura 17. Subsistema Plugins.

Cada plugin deberá implementar la Inteface CE_Plugin que brinda el Front-End. En el paquete de Análisis se encuentra: Análisis de Clustering, de Bifurcaciones, Estimación de Parámetros, de Estabilidad y por Reglas como se muestra en la figura 18.

En el paquete de análisis se encuentran los algoritmos que se utilizan para los diferentes análisis de los sistemas biológicos. Cada uno de estos plugins se relacionará con un Servicio Web del Subsistema de servicios.

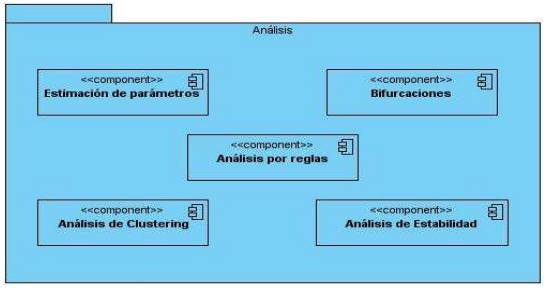

Figura 18. Paquete Análisis.

Distribución Física de los Componentes

En la figura 19 se muestra la distribución de los componentes en nodos físicos, además de los componentes desarrollados por la plataforma y programas independientes que son reutilizados como son: MatLab, Octave y tarenalLibrary. MatLab y Octave son programas que permiten la edición y representación gráfica de Sistemas de Ecuaciones Diferenciales (SED) que son la base para el funcionamiento del sistema. La librería tarenalLibrary permite la comunicación entre los Servicios Web y el servidor T-arenal.

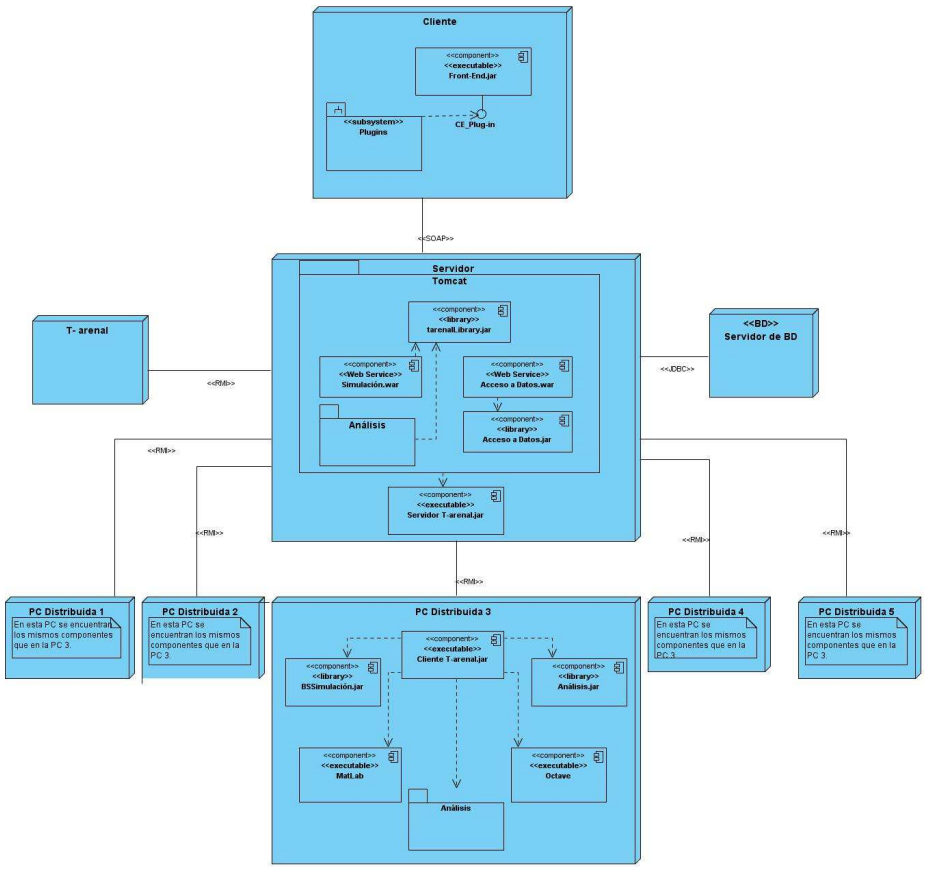

Figura 19. Distribución Física de los Componentes.

En el nodo cliente se encuentra el componente Front – End y el subsistema Plugins ya que en estos el usuario puede trabajar estando en su puesto de trabajo sin hacer ninguna petición al servidor. También se encontrarán los plugins que brindarán la interfaz gráfica, la comunicación con los servicios web y la interface CE_Plug-in. Los plugins Editor de Ecuaciones, Modelación y Graficador utilizan el Servicio Web de acceso a datos para insertar u obtener información de la base de datos. En el nodo Servidor se encontrará instalado el servidor de aplicaciones Tomcat, donde se alojarán los Servicios Web que tendrá la plataforma y la librería tarenalLibrary. En este nodo se encontrará un componente importante que es el Servidor T-arenal, que se encargará de distribuir el trabajo a las PCs distribuidas. En las PCs distribuidas estará ubicado el Cliente T-arenal que será el responsable de recibir los trabajos enviados por el servidor. En estas PCs se encontrarán los componentes de mayor peso, como son: El Paquete de Análisis que utilizará la librería Análisis.jar para realizar los diferentes tipos de análisis. MatLab y Octave harán uso de la librería BSSimulation.jar para mostrar los resultados del análisis en una gráfica.

Para una mejor comprensión del sistema será explicado mediante un ejemplo donde se mostrarán los detalles. Cuando el investigador desea analizar un nuevo modelo, se realiza la petición al servidor y este distribuye el trabajo a las PC distribuidas. Para realizar el análisis se necesita saber si el modelo se encuentra almacenado o no en la Base de Datos. Esto se efectúa a través de la petición que hace el Cliente T-arenal al Servidor T-arenal y este a su vez se conecta a través del servicio Acceso a Datos a la BD. Si el modelo está almacenado se obtienen los datos y se realiza el análisis. Si el modelo no se encuentra se realiza el análisis completo.

2.4 Conclusiones

El desarrollo de este capítulo permitió definir la arquitectura que se utilizará en el software BioSyS. Las figuras muestran los detalles de la organización del sistema para su mejor entendimiento. Se plasmaron los requisitos no funcionales del software. Cada una de las vistas arquitectónicas se definieron y explicaron detalladamente: Vista de Casos de Uso, Vista Lógica, Vista de Implementación, Vista de Despliegue y Vista de Procesos.

CAPÍTULO 3

Evaluación del diseño Arquitectónico Propuesto

CAPÍTULO 3

3.1 INTRODUCCIÓN

En este capítulo se presentan los distintos métodos para evaluar un diseño arquitectónico y a partir del método seleccionado se hace un análisis de la solución propuesta.

3.2 Evaluando la Arquitectura de Software

Evaluar una arquitectura de software es lo que permite prevenir todos los problemas de un diseño que no cumple con los requerimientos de calidad y para saber si la arquitectura diseñada para el sistema es la adecuada.

El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe señalar que los requerimientos no funcionales también son llamados atributos de calidad [18].

La evaluación de una arquitectura no te dice si la arquitectura es buena o mala, ni le da una calificación, sino que te dice dónde están los puntos riesgos del sistema, así como las fortalezas y debilidades que presenta la arquitectura [18].

3.2.1 Motivos para evaluar una Arquitectura de Software

Cuanto más temprano se encuentre un problema en un proyecto de software, mejor. El encontrar y arreglar un error en las etapas más tempranas del desarrollo del software, tiene mucho menor costo que en la fase de verificación. Debido a que la arquitectura es un producto temprano de la fase de diseño, esta tiene una profunda importancia en el sistema.

Un mal diseño de la arquitectura puede llevar a un proyecto al fracaso, ya que todos los requerimientos de calidad pueden quedar insatisfechos.

La arquitectura determina también la estructura del proyecto, por lo que es mejor cambiar la arquitectura y no otros artefactos.

3.2.2 Momento para evaluar la arquitectura

La evaluación de la arquitectura es evaluada generalmente después de que ya esta ha sido especificada o definida, y antes de que se comience la implementación. Si se utiliza un proceso iterativo y/o incremental, entonces la evaluación se puede realizar después de cada ciclo. Sin embargo la evaluación de la arquitectura se puede efectuar en cualquier etapa de la vida de una arquitectura de software. Existen 2 momentos para la evaluación de la arquitectura, estos con: Evaluación temprana y Evaluación tardía.

Evaluación temprana: En este tipo evaluación no es necesario que la arquitectura esté completamente especificada para poder evaluarla, esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

Evaluación tardía: Este tipo de evaluación se realiza cuando la arquitectura se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

La evaluación de la arquitectura debe realizarse cuando hay suficiente de la arquitectura para justificarlo. Existen 2 reglas que son llamadas reglas de oro para determinar el momento de las evaluaciones, estas son [18]:

- Realizar una evaluación cuando el equipo de desarrollo inicia a tomar decisiones que afectan directamente a la arquitectura
- Cuando el costo de no tomar estas decisiones podría tomar más, que el costo de realizar una evaluación.

Personas involucradas en la evaluación de la arquitectura

Generalmente las evaluaciones a la arquitectura se hacen por miembros del equipo de desarrollo, pero existen 2 grupos de personas que son los que más involucrados se encuentran:

- Equipo de Evaluación: Son las personas que conducirán la evaluación y realizaran el análisis.
- Stakeholders: Son los interesados en la arquitectura, y en el sistema que se construirá a partir de ella

3.2.3 Resultados de la evaluación de la arquitectura

Un resultado importante que produce la evaluación de la arquitectura es el grado en que se cumplieron los atributos de calidad. La evaluación de una arquitectura no produce resultados cuantitativos y ayuda a encontrar las debilidades del sistema. Lo que más importante de la evaluación es aprender cómo un

atributo de calidad es afectado por una decisión del diseño arquitectónico, para que de esta forma se pueda estudiar con cuidado esta decisión.

La evaluación arquitectónica dice si es adecuada o no una arquitectura respecto a un conjunto de metas y es una problemática con respecto a otro grupo de metas. En algunas ocasiones pueden existir contradicciones entre estas metas o algunas pueden ser más importantes que otras. Los resultados de la evaluación arquitectónica te ayudarán a encontrar las debilidades y te dirá donde están los riesgos que puede tener el sistema.

3.2.4 Cualidades por las que puede ser evaluada una arquitectura

No se puede afirmar que el sistema alcanzará todas sus metas de calidad con solo mirar la arquitectura, pero esta puede ser evaluada mediante atributos de calidad.

Estos atributos de calidad se pueden clasificar en [18]:

- Observables vía ejecución: Son aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución. La descripción de algunos de estos atributos se presenta en la tabla 1.
- No observables vía ejecución: Aquellos atributos que se establecen durante el desarrollo del sistema. La descripción de algunos de estos atributos se presenta en la tabla 2.

Tabla 1. Descripción de los atributos de calidad observables.

ATRIBUTOS DE CALIDAD	DESCRIPCIÓN
Disponibilidad	Es la habilidad del sistema de continuar operando sobre el tiempo.
Confidencialidad	Tiempo que el sistema está funcionando. Se mide como el tiempo transcurrido entre fallas, y cuán rápido el sistema está listo para reanudar y quedar operativo durante una falla.
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido.
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria.
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo.

Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información.
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos.
Flexibilidad	Es la habilidad del sistema para adaptarse a ambientes y situaciones variables y para soportar cambios en políticas de negocios y reglas de negocio. Un sistema flexible es uno que es fácil de reconfigurar o que se adapta en respuesta a los diferentes requerimientos de usuarios y del sistema.

Tabla 2. Descripción de los atributos de calidad no observables.

ATRIBUTOS DE CALIDAD	DESCRIPCIÓN
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema.
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados.
Integridad	Es la ausencia de alteraciones inapropiadas de la información.
Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema.
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema.
Mantenibilidad	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.
	Es la capacidad de diseñar un sistema de forma tal que su

Reusabilidad	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.

Si hubiese algún otro atributo de calidad, además de los mencionados y es importante para determinado proyecto, entonces este también debería formar parte de la evaluación.

3.2.5 Salidas de una evaluación arquitectónica

Las salidas de una evaluación arquitectónica son información e ideas sobre la arquitectura.

- **Lista priorizada de los atributos de calidad requeridos**: Una evaluación arquitectónica solo puede proceder si se conoce el criterio de adecuabilidad. Es más, la obtención de los atributos de calidad requeridos contra los cuales la arquitectura será juzgada, constituye la mayor parte del trabajo. Pero ninguna arquitectura puede tener una lista interminable de atributos de calidad, y por lo tanto, se utilizan métodos de priorización consensuados [19].
- **Riesgos y no riesgos:** Los riegos son decisiones arquitectónicas potencialmente problemáticas. Los no riesgos son buenas decisiones, que confían en asunciones que con frecuencia son implícitas en la arquitectura [19].

3.2.6 Costos y beneficios de realizar una evaluación arquitectónica

El mayor beneficio de la evaluación de la arquitectura es que descubre los problemas que si no se hubiesen encontrado, habría sido mucho más costos corregirlos luego. Entre los beneficios se encuentran [18]:

- Financieros.
- Reúne a los stakeholders.
- Fuerza una articulación en las metas específicas de calidad.

- Fuerza una mejora a la documentación de la arquitectura.
- Mejora la arquitectura.
- Detección temprana de problemas.
- Valido requerimiento y los priorizo.
- Recolecta los fundamentos y las decisiones arquitectónicas tomadas.

3.3 Métodos de Evaluación de Arquitecturas de Software

Los métodos de evaluación de arquitectura de software surgen hace poco tiempo. La necesidad de una forma de evaluar los diferentes tipos de arquitecturas que se pueden utilizar hace que surjan diferentes métodos que han sido propuestos.

3.3.1 Método de Análisis de Arquitecturas de Software

El Método de Análisis de Arquitecturas de Software (SAAM, del inglés Software Architecture Analysis Method) fue el primer método que fue ampliamente divulgado y documentado. 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.

Este método 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. SAAM puede ser utilizado para evaluar una o múltiples arquitecturas. Con la aplicación de este método, si el objetivo es evaluar una sola arquitectura, se obtienen los lugares en los que esta puede fallar, en término de los requerimientos de modificabilidad. En el caso que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar que opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones.

SAAM consta de un grupo de pasos [20]:

Tabla 3. Pasos del método SAAM.

1. Desarrollo de Escenarios.	Un escenario es una breve descripción de usos anticipado o deseados del sistema. De igual forma estos pueden incluir cambios a los que puede estar expuesto el sistema

	en el futuro.
2. Descripción de la arquitectura.	La arquitectura (o las candidatas) debe ser descrita haciendo uso de alguna notación arquitectónica que sea común a todas las partes involucradas en el análisis. Deben incluirse los componentes de datos y conexiones relevantes, así como la descripción del comportamiento general del sistema.
3. Clasificación y asignación de prioridad de los escenarios.	La clasificación de los escenarios se puede hacer en 2 categorías: **Escenario directo:** Es aquel que requiere modificaciones en la arquitectura para poder satisfacerse. **Escenario indirecto:** Estos son los de principal interés en este método, pues son los que permiten medir el grado en el que una arquitectura puede ajustarse a los cambios de evolución que son importantes para los involucrados en el desarrollo.
4. Evaluación individual de los escenarios indirectos.	Para cada escenario indirecto se listan los cambios necesarios sobre la arquitectura y se calcula su costo. Una modificación sobre la arquitectura significa que debe introducirse un nuevo componente o conector, o que alguno de los existentes requiere cambios en su especificación.
5. Evaluación de la interacción entre escenarios.	Cuando dos o más escenarios indirectos proponen cambios sobre un mismo componente, es necesario evaluar este hecho, puesto que la interacción de los componentes semánticamente no relacionados revela que los componentes de la arquitectura efectúan funciones semánticamente distintas. De forma similar, puede verificarse si la arquitectura se encuentra documentada a un nivel correcto de descomposición estructural.

6. Creación de la evaluación global.	Debe asignarse un peso a cada escenario, en términos de su importancia relativa al éxito del sistema. Esta asignación de peso suele hacerse con base en las metas del negocio que cada escenario soporta. En caso de la evaluación de múltiples arquitecturas, la asignación de pesos puede ser utilizada para la determinación de una escala general.

3.3.2 Método de Análisis de Acuerdos de Arquitectura

El Método de Análisis de Acuerdos de Arquitectura (ATAM, del inglés Architecture Trade-off Analysis Method) está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y en el SAAM. El nombre de este método 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 [20].

ATAM 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, también permiten describir en la que el sistema puede crecer, responder a cambios e integrarse con otros sistemas.

Este método consta de nueve pasos agrupados en cuatro fases. A continuación se muestran en forma de tablas las fases con cada paso y la descripción de cada una de ellas.

Tabla 4. Fases del Método ATAM [20].

FASE 1. PRESENTACIÓN.	
1. Presentación del ATAM	El líder de evaluación describe el método a los participantes, trata de establecer las expectativas y responde a las preguntas propuestas.
2. Presentación de las metas del negocio.	Se realiza la descripción de las metas del negocio que motivan el esfuerzo y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura.	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los

	objetivos del negocio.
Fase 2. Investigación y análisis.	
4. Identificación de los enfoques arquitectónicos.	Estos elementos son detectados pero no analizados.
5. Generación de Utility Tree.	Se necesitan los atributos de calidad que engloban las “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, entre otros) especificados en forma de escenarios. Se anotan los estímulos y respuestas, y se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos.	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3. Pruebas.	
7. Lluvia de ideas y establecimiento de prioridad de escenarios.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.
8. Análisis de los enfoques arquitectónicos.	Este paso repite las actividades del paso 6 haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de pruebas para confirmar el análisis realizado hasta el momento.
Fase 4. Reporte.	
9. Presentación de los resultados.	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

3.3.3 Método de Revisiones Activas para Diseños Intermedios

Generalmente las arquitecturas crecen poco a poco a través de una serie de pasos, donde cada uno de estos pasos puede presentar complejos problemas de subdiseño. Si estos diseños intermedios se resuelven de una manera inapropiada, la arquitectura puede no ser adecuada para el proyecto.

Debido a estos problemas, hacerle revisiones a la arquitectura en las etapas intermedias brinda una valiosa visón de la viabilidad de la arquitectura a construir, y también nos permite descubrir errores e inconsistencias en el sistema. En un futuro la mayoría de los proyectos realizarán estas revisiones en sus subsistemas.

Lo que realmente se quieres en estas etapas, es una breve evaluación que se concentre en la conveniencia, exponiendo el diseño a los interesados provocando interés y que pueda ser llevada a cabo en la ausencia de una detallada documentación.

El Método de Revisiones Activas para Diseños Intermedios (ARID, del inglés Active Reviews for Intermediate Design) cumple con todas estas características y se basa en las mejores cualidades de los métodos basados en escenarios (ATAM o SAAM) y las revisiones activas de diseños (ADRS). Esta mezcla de métodos se basa en ensamblar el diseño de los stakeholders para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios.

ARID es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de 9 pasos agrupados en dos fases (Actividades Previas y Evaluación).

Tabla 5. Fases del método ARID [19].

FASE 1. PRE – REUNIÓN.	
1. Identificar los revisores.	Esto lo hacen los ingenieros de software que van a usar el diseño.
2. Preparar la presentación del diseño.	El arquitecto prepara un informe que explica el diseño, el mismo deberá ser lo suficientemente detallado como para que una capacitada audiencia pueda usar el diseño.
3. Preparar los escenarios.	El arquitecto y el líder preparan los escenarios

	que sirven para ilustrar los conceptos a los revisores.
4. Preparar los materiales.	Se realizan copias de las presentaciones, escenarios, la agenda invitando a interesados externos y asegurando la presencia de los revisores.
Fase 2. Evaluación.	
1. Presentación del ARID	El líder utiliza 30 minutos para explicar los pasos de la evaluación a los participantes.
2. Presentación del diseño.	El arquitecto realiza una presentación del diseño mostrando los ejemplos. Durante esta se evita hacer preguntas concernientes a la implementación, ni tampoco se proponen diseños alternativos. El objetivo es ver si el diseño es adecuado, no saber porque el diseño se hizo de esa manera.
3. Lluvia de ideas y priorización de escenarios.	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
4. Se realiza la revisión.	Comenzando con el escenario que contó con más votos, el facilitador solicita el seudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos:

	➢ Se agota el tiempo destinado a la revisión. ➢ Se han estudiado los escenarios de más alta prioridad. ➢ El grupo se siente satisfecho con la conclusión alcanzada.

3.4 Evaluando la arquitectura del Software para la Simulación y Análisis de los Sistemas Biológicos

Luego de un estudio realizado, se decidió escoger el método ARID para la evaluación de la arquitectura propuesta, ya que proporciona la evaluación en fases tempranas del diseño, permitiendo la realización de cambios cuando estos no requieran demasiados esfuerzos. ARID es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo y la técnica de evaluación basada en escenarios.

Para la evaluación de la arquitectura propuesta se seleccionaron algunos atributos de calidad y un listado de riesgos para ver la relación que existe entre ellos.

Los atributos de calidad seleccionados fueron:

- Escalabilidad.
- Mantenibilidad.
- Integridad.
- Disponibilidad.
- Portabilidad.

Los riesgos asociados a los atributos seleccionados son:

- Añadir servicios a la plataforma.
- Migración del SGBD.
- Mantenimiento de las funcionalidades de la plataforma.
- Acceso a los datos.
- Funcionamiento del sistema.
- Cambio de sistema operativo.

Tabla 6. Atributo Escalabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Escalabilidad	Escalabilidad	Añadir servicios a la plataforma.
Relación atributo – escenario.		
Para la arquitectura la escalabilidad es la característica fundamental. Los servicios que se brindan a los investigadores pueden añadirse con facilidad debido a que fueron programados como componentes independientes. Para agregar una nueva funcionalidad se implementaría un nuevo componente compuesto por un Servicio Web, un Plugin y una aplicación que permita realizar la operación que se desea.		
Ambiente	Operación normal.	
Estímulo	Agregar nuevas funcionalidades al sistema.	
Respuesta	La arquitectura definida soporta la incorporación de nuevas funcionalidades.	

Tabla 7. Atributo Mantenibilidad.

Atributo de calidad	**Perfil**	**Escenario**
Mantenibilidad	Mantenimiento	Migración del SGBD.
Relación atributo – escenario.		
Para el desarrollo de la plataforma se utiliza el framework Hibernate, que es una clara implementación del patrón DAO. Este framework crea una capa separada que se ocupa del acceso a datos de manera independiente al gestor y la BD, brinda la posibilidad de trabajar con varios gestores y BD dentro de la misma aplicación sin crear ningún conflicto en el modelo de objetos. Estas características posibilitan un cambio de SGBD de forma sencilla, incluso, la existencia de varios de ellos.		
Ambiente	Operación normal.	
Estímulo	Necesidad del cliente o del equipo de desarrollo de migrar a otro SGBD.	
Respuesta	No hay cambios significativos ni en las decisiones arquitectónicas establecidas.	

Tabla 8. Atributo Mantenibilidad.

Atributo de calidad	**Perfil**	**Escenario**
Mantenibilidad	Mantenimiento	Mantenimiento de las funcionalidades de la plataforma
Relación atributo – escenario.		
Debido a que se utiliza una Arquitectura Basada en Componentes se simplifica el mantenimiento cuando existe un débil acoplamiento entre componentes, esto permite actualizar o agregar componentes según sea necesario, sin afectar otras partes del sistema, dado que un componente puede ser construido y luego mejorado continuamente.		
Ambiente	Operación normal.	
Estímulo	Agregar o actualizar funcionalidades a la plataforma.	
Respuesta	No hay cambios significativos en el sistema.	

Tabla 9. Atributo Integridad.

Atributo de calidad	**Perfil**	**Escenario**
Integridad	Integridad	Acceso a los datos.
Relación atributo – escenario.		
El usuario deberá autenticarse para acceder a la BD y solo tendrá acceso a los sistemas y modelos que el halla insertado. La información manejada por el sistema será objeto de cuidadosa protección contra la corrupción y estados inconsistentes, de la misma forma será considerada igual a la fuente o autoridad de los datos. Pueden incluir también mecanismos de chequeo de integridad y realización de auditorías.		
Ambiente	Operación normal.	
Estímulo	El usuario hace uso de las funcionalidades del sistema.	
Respuesta	No hay cambios en la implementación de dicha funcionalidad.	

Tabla 10. Atributo Disponibilidad.

Atributo de calidad	**Perfil**	**Escenario**

Disponibilidad	Disponibilidad	Funcionamiento del sistema.
Relación atributo – escenario.		
El funcionamiento del sistema tiene una gran disponibilidad. En caso de fallas de red o problemas con el servidor, no existirá pérdida alguna de los datos ya que cuentan con recuperación de errores y además la existencia de un gestor de BD local que permitirá continuar el trabajo en caso de fallas del sistema.		
Ambiente	Operación normal.	
Estímulo	En caso de fallas con el servidor.	
Respuesta	El sistema no presentará pérdidas de datos.	

Tabla 11. Atributo Portabilidad.

Atributo de calidad	**Perfil**	**Escenario**
Portabilidad	Portabilidad	Cambio de sistema operativo
Relación atributo – escenario.		
Para obtener un producto portable la decisión más importante fue escoger las tecnologías y herramientas para el desarrollo de la plataforma. El lenguaje utilizado para la implementación fue Java y el gestor de BD PostgreSQL permiten al sistema que se puedan ejecutar tanto en el sistema operativo Windows o Linux, y sobre cualquier sistema operativo que soporte la máquina virtual de Java.		
Ambiente	Operación normal.	
Estímulo	Despliegue de aplicaciones.	
Respuesta	La arquitectura definida soporta que pueda desplegarse en ambos sistemas operativos.	

3.5 Conclusiones

En este capítulo se analizó la importancia de la evaluación de la arquitectura y un estudio de los métodos de evaluación que existen. La arquitectura propuesta se evaluó mediante el método ARID, permitiendo así obtener una evaluación de la arquitectura en las etapas tempranas del diseño, mostrando cómo se analiza el cumplimiento de los atributos de calidad en cada una de las vistas desarrolladas.

CONCLUSIONES GENERALES

- Se utilizó del estilo Llamada y Retorno la Arquitectura Basada en Componentes.
- Se propuso para el diseño de cada módulo que se debe seguir el modelo 3 Capas y solo en los casos en que sea necesario violar la capa de control se seguirá el patrón MVC.
- Se diseñaron las vistas arquitectónicas del sistema utilizando el Modelo 4 + 1 Vistas de Philippe Kruchten.
- Se realizó la evaluación del diseño arquitectónico propuesto a través del método ARID.

RECOMENDACIONES

Luego de haber analizado los resultados del presente trabajo de diploma, resulta factible arribar a las siguientes recomendaciones:

- Poner en práctica el diseño arquitectónico propuesto.
- Realizar otras pruebas a la Arquitectura propuesta a través de los demás métodos de evaluación.

REFERENCIAS BIBLIOGRÁFICAS

1. López, Marta, Ruiz, Gema and Vega, MIguel. 2007.

2. Ledón, Irilys. *Propuesta del Diseño Arquitectónico del Simulador de Sistemas Biológicos: BioSyS.* 2008.

3. Cristiá, Maximiliano. [Online] 2008.

4. Quintero, Carlos Enrique Cuesta. [Online] [Cited: 10 02, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=8&ved=0CBgQFjAH&url=http%3A%2F%2Fwww.cervantesvirtual.com%2Fservlet%2FSirveObras%2F01361653144571409089802%2F010349.pdf&rct=j&q=definicion+de+arquitectura+de+software+de+perry+y+wolf&ei=i6zcSuz4D5ClIA.

5. Soto, Lauro. [Online] 12 02, 2009. [Cited: 01 03, 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=10&ved=0CB4QFjAJ&url=http%3A%2F%2Ftriana.escet.urjc.es%2Faspf%2FASPF-Tema4-.

6. URJC, Informáticos. [Online] 12 06, 2009. [Cited: 01 03, 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=10&ved=0CB4QFjAJ&url=http%3A%2F%2Ftriana.escet.urjc.es%2Faspf%2FASPF-Tema4-Decisiones.pdf&rct=j&q=perry+y+wolf+%2B+definicion+de+arquitectura+de+software&ei=yMDcSqq8LNXhlAeu-4CiAQ&usg=AFQjCNEuns4mWpVO.

7. Casanova, Josep. [Online] 09 09, 2004. [Cited: 01 05, 2010.] http://www.desarrolloweb.com/articulos/1622.php.

8. Reynoso, Carlos and Kiccillof, Nicolás. [Online] 2004. [Cited: 11 11, 2009.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

9. Figueroa, Pablo. [Online] 1997. [Cited: 01 15, 2010.] http://agamenon.uniandes.edu.co/~pfiguero/soo/Magister_Patrones/intropatrones.html.

10. **Bushmann, Frank.** [Online] [Cited: 11 02, 2009.] http://www.ejournal.unam.mx/cys/vol01-02/CYS01207.pdf.

11. Gracia, Joaquin. [Online] 05 27, 2005. [Cited: 02 16, 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

12. Javeriana, Pontificia Universidad. [Online] [Cited: 12 12, 2009.] http://sophia.javeriana.edu.co/~lcdiaz/ADOO2006-3/grasp_cpaternostro-lvargas-jviafara.pdf.

13. Catarina. [Online] [Cited: 03 26, 2010.] http://www.willydev.net/descargas/prev/ADL.pdf.

14. Puert, Sergio, et al. [Online] [Cited: 04 02, 2010.] http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

15. Falcones, Idelfonso. [Online] [Cited: 04 05, 2010.] http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf.

16. Software, Hispavista. [Online] 12 12, 2009. http://www.assembla.com/wiki/print/andalucia/Entregable_E6.

17. Dávila, Mauricio, et al. [Online] [Cited: 04 20, 2010.] http://windows.software.hispavista.com/nt/n247-java-developers-kit-jdk-6/

18. Puebla, Yoan Arlet Carrascoso, Gómez, Enrique Chaviano and Vega, Anisleydi Céspedes. [Online] 05 07, 2009. http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

19. Dávila, Mauricio, et al. [Online] [Cited: 04 10, 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAYQFjAA&url=http%3A%2F%2Fwww.fing.edu.uy%2Finco%2Fcursos%2Fgestsoft%2FPresentaciones%2FEvaluacion%2520de%2520Arquitecturas%2520-%2520G10%2FEvaluacion%2520de%2520Arquitecturas.doc&rct=j&q=M%E9t.

20. Camacho, Erika, Cordeso, Fabio and Nuñez, Gabril. [Online] 04 2004. [Cited: 04 29, 2010.] http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf

BIBLIOGRAFÍA

1. López, Marta, Ruiz, Gema and Vega, Miguel. [Online] 2007. [Cited: 10 10, 2009.] http://www.uvic.cat/eps/dept/biologiasistemes/es/inici.html.

2. Ledón, Irilys. *Propuesta del Diseño Arquitectónico del Simulador de Sistemas Biológicos: BioSyS.* 2008.

3. Quintero, Carlos Enrique Cuesta. [Online] [Cited: 10 12, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=8&ved=0CBgQFjAH&url=http%3A%2F%2Fwww.cervantesvirtual.com%2Fservlet%2FSirveObras%2F01361653144571409089802%2F010349.pdf&rct=j&q=definicion+de+arquitectura+de+software+de+perry+y+wolf&ei=i6zcSuz4D5CllA.

4. Soto, Lauro. [Online] 12 2, 2009. [Cited: 10 16, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=10&ved=0CB4QFjAJ&url=http%3A%2F%2Ftriana.escet.urjc.es%2Faspf%2FASPF-Tema4-.

5. URJC, Informáticos. [Online] 12 6, 2009. [Cited: 10 20, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=10&ved=0CB4QFjAJ&url=http%3A%2F%2Ftriana.escet.urjc.es%2Faspf%2FASPF-Tema4-Decisiones.pdf&rct=j&q=perry+y+wolf+%2B+definicion+de+arquitectura+de+software&ei=yMDcSqq8LNXhlAeu-4CiAQ&usg=AFQjCNEuns4mWpVO.

6. Casanova, Josep. [Online] 09 09, 2004. [Cited: 10 22, 2009.] http://www.desarrolloweb.com/articulos/1622.php.

7. Reynoso, Carlos and Kiccillof, Nicolás. [Online] 2004. [Cited: 10 25, 2009.] http://www.willydev.net/descargas/prev/Estiloypatron.pdf.

8. Figueroa, Pablo. [Online] 1997. [Cited: 10 30, 2009.] http://agamenon.uniandes.edu.co/~pfiguero/soo/Magister_Patrones/intropatrones.html.

9. Bushmann, Frank. [Online] [Cited: 11 2, 2009.] http://www.ejournal.unam.mx/cys/vol01-02/CYS01207.pdf.

10. *Conferencia de ingenieria de software II. .* **(UCI), Universidad de las Ciencias Informáticas.** 2008-2009.

11. Reynoso, Carlos. [Online] 03 2004. [Cited: 01 25, 2010.] http://www.willydev.net/descargas/prev/ADL.pdf.

12. Sierra, María. [Online] [Cited: 01 30, 2010.] http://personales.unican.es/ruizfr/is1/doc/lab/01/is1-p01-trans.pdf.

13. Aragón, Sergio Puerta, et al. [Online] [Cited: 02 04, 2010.] http://www.assembla.com/wiki/print/andalucia/Entregable_E6.

14. Falcones, Idelfonso. [Online] [Cited: 04 05, 2010.] http://www.agapea.com/libros/Java-JDK-6-isbn-8441522200-i.htm.

15. Hispavista Software. [Online] 12 12, 2006. http://windows.software.hispavista.com/nt/n247-java-developers-kit-jdk-6/

16. Puebla, Yoan Arlet Carrascoso, Gómez, Enrique Chaviano and Vega, Anisleydi Céspedes. [Online] 07 04, 2009. [Cited: 02 10, 2010.] http://www.gestiopolis.com/administracion-estrategia/procedimiento-para-la-evolucion-de-las-arquitecturas-de-software.htm.

17. Dávila, Mauricio, et al. [Online] [Cited: 04 20, 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAYQFjAA&url=http%3A%2F%2Fwww.fing.edu.uy%2Finco%2Fcursos%2Fgestsoft%2FPresentaciones%2FEvaluacion%2520de%2520Arquitecturas%2520-%2520G10%2FEvaluacion%2520de%2520Arquitecturas.doc&rct=j&q=M%E9t.

18. Camacho, Erika, Cordeso, Fabio and Nuñez, Gabril. [Online] 04 2004. [Cited: 04 29, 2010.] http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf

19. González, Mario. [Online] [Cited: 10 21, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=2&ved=0CAkQFjAB&url=http%3A%2F%2Fwww.ldc.usb.ve%2F~abianc%2Fmaterias%2Fci4712%2FSoftware%2520Architecture-kazman.pdf&rct=j&q=+estilos+arquitectonicos+%2B+arquitectura+de+software&ei=nV7fSr7AFZOqlAeX18.

20. Lago, Ramiro. [Online] 04 2007. [Cited: 10 09, 2009.] http://www.proactiva-calidad.com/java/patrones/index.html.

21. Gutierrez, Jorge A. Saavedra. [Online] 05 08, 2007. [Cited: 10 30, 2009.] http://jorgesaavedra.wordpress.com/category/patrones-grasp/.

22. Pelaez, Juan Carlos. [Online] 05 29, 2009. [Cited: 11 05, 2009.] http://geeks.ms/blogs/jkpelaez/archive/2009/05/29/arquitectura-orientada-a-servicios-soa.aspx.

23. Geograma. [Online] [Cited: 12 10, 2009.] http://www.geograma.com/es/publicaciones/arquitectura-soa-para-la-integraci-n-entre-software-libre-y-software-propietario-en-entornos-mixtos.html.

24. Regalado, Yamila Vigil and Cabana, Erich Fouces. [Online] 03 17, 2008. [Cited: 12 16, 2009.] http://www.desarrolloweb.com/articulos/1622.php.

25. Carlos Reynos, Nicolas Kiccillof. [Online] 03 2004. [Cited: 11 17, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=1&ved=0CAcQFjAA&url=http%3A%2F%2Fwww.willydev.net%2Fdescargas%2Fprev%2FADL.pdf&rct=j&q=lenguajes+de+descripcion+arquitectonico&ei=2u4CS7qsH8rglAfowvDhAQ&usg=AFQjCNEzPJL5jwg6kl7YvoN6jAIeVBGXfg.

26. Lovelle, Juan Manuel Cueva. [Online] 10 1999. [Cited: 11 26, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=4&ved=0CBgQFjAD&url=http%3A%2F%2Fgidis.ing.unlpam.edu.ar%2Fdownloads%2Fpdfs%2FIntroduccionUML.PDF&rct=j&q=uml&ei=H-gOS4tyh8KUB8e8sZIE&usg=AFQjCNFH_iNC8c8fjkbGhKMYjpQcvd79mA.

27. Orallo, Enrique Hernandez. [Online] [Cited: 12 1, 2009.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=3&ved=0CAwQFjAC&url=http%3A%2F%2Fwww.disca.upv.es%2Fenheror%2Fpdf%2FActaUML.PDF&rct=j&q=Lenguaje+unificado+de+Modelado&ei=cmEVS9SJJcWTlAfY5KnABQ&usg=AFQjCNExdi1eDksTTxmXd8asVpdSnSQQfg.

28. Sanchez, María A. Mendoza. [Online] 06 07, 2007. [Cited: 01 12, 2010.] http://www.willydev.net/Descargas/cualmetodologia.pdf.

29. El rincón del vago. [Online] 09 17, 2003. [Cited: 12 15, 2009.] http://pdf.rincondelvago.com/herramientas-case_2.html.

30. Seco, José Antonio González. [Online] [Cited: 01 18, 2010.] http://programmatium.blogspot.com/2008/01/el-lenguaje-de-programacin-c_23.html.

31. Rayes, Andres. [Online] 12 29, 2003. [Cited: 02 23, 2010.] http://www.programacion.com/articulo/lenguajes_patrones/

32. Gestión de Proyectos. [Online] 09 07, 2008. [Cited: 03 12, 2010.] http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html.

33. González, Carlos D. [Online] 05 2010. [Cited: 05 25, 2010.] http://www.usabilidadweb.com.ar/postgre.php.

34. Juan Pablo Cabrera Rosique, Jose Fernando Valera Fdez. [Online] [Cited: 03 09, 2010.] http://dis.um.es/~lopezquesada/documentos/IES_0506/RAL_0506/doc/prac6ut1.pdf.

35. Gutiérrez, Javier J. [Online] [Cited: 03 09, 2010.] http://www.lsi.us.es/~javierj/investigacion_ficheros/Framework.pdf.

36. Bagüés, Ramiro Lago. [Online] 01 2008. [Cited: 03 10, 2010.] http://www.proactiva-calidad.com/java/spring/introduccionSpring.html.

37. MoisesDaniel's Bloc. [Online] 12 18, 2007. [Cited: 03 10, 2010.] http://www.moisesdaniel.com/bloc/archives/2007/12/entry_17.html.

38. Garzás, Javier. [Online] [Cited: 03 16, 2010.] http://sites.google.com/site/jgarzas/4mas1.

39. Bianchi, Alejandro. [Online] 05 19, 2009. [Cited: 04 12, 2010.] http://www.sg.com.mx/content/view/866.

40. Cristiá, Maximiliano. *Introducción a la Arquitectura.* 2008.

41. Gracía, Joaquin. [Online] 05 27, 2005. [Cited: 02 20, 2010.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php

42. Javeriana, Pontificia Universidad. Departamento de Ingeniería de Sistemas. [Online] http://sophia.javeriana.edu.co/~lcdiaz/ADOO2006-3/grasp_cpaternostro-lvargas-jviafara.pdf.

Glosario de Términos

ADLs: Lenguajes de Descripción Arquitectónicas.
ADR: (Active Design Review). Método 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.
AOP: Paradigma de Orientación de Aspectos.
API: Interfaz de Programación de Aplicaciones (Aplication Programming Interface siglas en inglés). Conjunto de especificaciones de comunicación entre componentes de software. Representa un método para conseguir abstracción en la programación.
AS: Arquitectura de Software.
ARID: Revisiones Activas para Diseños Intermedios (ARID, del inglés Active Reviews for Intermediate Design) proporciona la evaluación en fases tempranas del diseño, permitiendo la realización de cambios cuando estos no requieran demasiados esfuerzos.
AS: Arquitectura de Software.
ATAM: Método para la evaluación de la arquitectura que permite determinar los efectos que un determinado atributo de calidad tiene sobre otros atributos.
DataManager:
BD: Base de Datos.
Bioinformática: Área de investigación multidisciplinaria puede ser ampliamente definida como la interfase entre dos ciencias: Biología y Computación.
Biología de Sistemas: La Biología de Sistemas es un área de estudio emergente que se caracteriza por la integración de aproximaciones experimentales y computacionales en la comprensión de los sistemas biológicos. Esta disciplina permite estudiar los mecanismos que gobiernan los sistemas complejos y las interacciones existentes entre los diferentes niveles de información biológica.
BSD: Licencia de Distribución de Software Bekerley (Bekerley Software Distribution siglas en inglés).
CAD: Sistema de Pizarra o Repositorio (Patrones Arquitectónicos).
CIM: Centro de Inmunología Molecular.
CU: Casos de Usos.
CVS: Sistema de versiones concurrentes.
DAO: Patrón de Acceso a Datos (Data Access Object, según sus siglas en inglés).
GPL: Licencia Pública General (General Public Licence siglas en inglés).
GUI: Interfaz Gráfica de Usuario (Graphic User Interface).
Http: Protocolo de Transferencia de HyperTexto (HyperText Transfer Protocol" por sus siglas en inglés)
IoC: Técnica de Inversión de Control
JDBC: Es el acrónimo de Java Database Connectivity, un API que permite la ejecución de operaciones sobre base de datos desde el lenguaje de programación Java.
JDK: Kit de Desarrollo de Java (Java Development Kit).
JSP: Pagina Servidora de Java (Java Server Pages).
MM: Modelo Matemático.

MVC: Patrón Modelo Vista Controlador.
Open Source: Cualidad de algunos programas de incluir el código fuente en la distribución del programa. Se usa para referirse al software libre.
Open UP/ Basic: Proceso Unificado Abierto.
PC: Es la expresión estándar que se utiliza para denominar a las computadoras personales en general.
PC Distribuida: Computadora que se encontrará como cliente dedicado al servidor T-arenal.
Plug-ins: Aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.
POO: Programación Orientada a Objeto.
RMI: Invocación a métodos remotos, tecnología Java para el trabajo distribuido.
SED: Sistema de Ecuaciones Diferenciales.
Servlet: Objeto Java que se ejecuta dentro del contexto de un servidor web y que sirve para manipular peticiones y respuestas del cliente web. La mayor parte de los frameworks utilizan un Servlet como núcleo.
SGBD: Sistema Gestor de Base de Datos.
Sistemas Biológicos: Grupo de organismos que se relacionan entre sí.
SOA: Arquitectura Orientada a Servicios.
SOAP: Siglas de Simple Object Access Protocol, Protocolo simple de acceso a datos.
SVN: Sistema Controlador de Versiones Subversion.
URL: Las siglas vienen de Uniform Resource Locator (en español: "Localizador Uniforme de Recursos"), se presenta como una secuencia de caracteres bajo una forma estándar para darle nombre a determinados recursos en una red.
XML: Siglas en inglés de Extensible Markup Language (Lenguaje Extensible de Marcas).