Universidad de las Ciencias Informáticas

Facultad 6

Título: “Propuesta del Diseño Arquitectónico del Simulador de Sistemas Biológicos: BioSyS “

Trabajo de Diploma para optar por el título de Ingeniero Informático

Autor: Irilys Ledón Robaina

Tutor: Vilmavis La Rosa Sordo.

Junio 2008

DECLARACIÓN DE AUTORÍA

Declaro ser autora de la presente tesis y reconozco 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 junio del año 2008.

Irilys Ledón Robaina	Ing. Vilmavis La Rosa Sordo
_____	_____
Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Vilmavis La Rosa Sordo: Ing. Informático, graduado en el 2006, instructor.

Correo electrónico: vlarosa@uci.cu

DEDICATORIA

A mi mami, por su inmenso amor, su constante dedicación, por ser mi total inspiración y por que sin su total apoyo hubiese sido imposible conseguirlo. A ti todo mi esfuerzo.

A mi papá por su ayuda, apoyo, por esperar tanto de mi y porque sé que aunque de una manera ¨muy especial¨ me adora y se enorgullece de tenerme como hija.

A mi viejo por estar siempre para mí y por quererme tanto.

A mi abuela Elisa (mima) donde quiera que este por enseñarme tantas cosas de niña, porque la añoro, y porque... daría cualquier cosa por tenerte justo ahora que se han cumplido muchos de tus sueños aún cuando estás lejos de mi.

A mi familia toda que siempre me ha dado su apoyo y me ha querido tanto especialmente a mis tías Xiomara y Kenia.

AGRADECIMIENTOS

A mi mamá por estar pendiente de cada una de estas líneas y por darme fuerzas para seguir adelante.

A mi papá por siempre confiar en mí, aún cuando dudé.

A mi tutora Vilmavis por su ayuda.

A Noel y Edel por su ayuda e incondicional apoyo siempre que los necesité.

A mis vecinos por estar siempre pendientes y por cuidar de mi mami por mí.

A mis dos hermanitas Ide e Isma por ser tan especiales, por haberme enseñado que en verdad existe la palabra ¨amistad¨ y porque juntas hemos vivido momentos inolvidables.

A los amigos que siempre estuvieron, aún en los momentos difíciles, en especial a Mabel y Alejandro.

A todas aquellas personas que de una forma u otra han hecho posible este sueño.

A la revolución por haberme dado esta maravillosa oportunidad.

RESUMEN

BioSyS (Software para la Simulación de Sistemas Biológicos) es un software de propósito general producto del desarrollo del software independiente que permite modelar, editar ecuaciones, estimar parámetros, simular tanto de forma local como distribuida y hacer meta análisis de los resultados obtenidos en las simulaciones a modelos matemáticos asociados a sistemas biológicos que puedan ser descritos mediante sistemas de ecuaciones diferenciales.

El presente trabajo centra su atención en un diseño arquitectónico para BioSyS que permita la obtención de una arquitectura estable, flexible, reusable y que cumpla con los requisitos necesarios para el funcionamiento adecuado de dicho sistema.

Durante el desarrollo del trabajo se representa la arquitectura haciendo uso del modelo 4+1 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 ATAM permitiendo comprobar que la arquitectura propuesta cumple con los requisitos indispensables para el cumplimiento de la calidad del sistema.

Como resultado de este trabajo se ha logrado el diseño de una arquitectura reusable que permite la creación de nuevos simuladores modulares con características semejantes en un tiempo de desarrollado relativamente corto sin tener que hacer cambios significativos a la arquitectura y además flexible permitiendo la agregación o eliminación de nuevas funcionalidades al sistema.

PALABRAS CLAVE

- Arquitectura de software.
- Estilos.
- Patrones.
- Vistas Arquitectónicas.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La industria del software es ya la cuna de la economía del mundo. El software se ha convertido en el elemento clave de la evolución de los sistemas y productos informáticos. En las pasadas cuatro décadas, ha pasado de ser una resolución de problemas especializadas y una herramienta de análisis de información, para ser una industria por sí misma.

Muchos software han sido desarrollados con el propósito de modelar, simular y analizar sistemas biológicos, unos de propósito general y otros más específicos, gracias a la inspiración de la Biología de Sistemas, ciencia interesada en comprender el funcionamiento de los sistemas biológicos.

La Biología de Sistemas se puede dividir en dos grandes áreas, una experimental y una computacional. En el lado experimental tenemos la manipulación, consistente en la realización de trabajos en el laboratorio y la medición, que hace énfasis en la colección sistemática de datos y el desarrollo de nuevos métodos y tecnologías experimentales. En el computacional, encontramos la minería de datos, encargada de la búsqueda de relaciones que subyacen ocultas dentro de las bases de datos que almacenan información biológica y que pueden ser capturadas mediante el uso de modelos predicativos. Finalmente encontramos el diseño, que es una fase importante en el estudio de los sistemas biológicos y se centra en la creación de modelos de estos sistemas para realizar estudios y probar nuevas hipótesis en un contexto mucho más barato y rápido que el laboratorio.

Está última área, la computacional ha sido la de más rápido desarrollo en los últimos años. Actualmente existen empresas de biología de sistemas cuyos productos están intrínsecamente relacionados con la simulación de problemas biológicos. Ejemplo de tales empresas son: Gene Network Sciences, Entelos, Physiome, Genomática. Sin embargo todas ellas siguen como estrategia de negocios la formación de alianzas comerciales con empresas biotecnológicas y no la comercialización directa del software.

El desarrollo del software independiente también ha ido en ascenso, baste destacar que han sido reportados más de 100 software que permiten modelar, simular o realizar análisis sobre modelos de sistemas biológicos. [1] Muchos de ellos trabajan con algún tipo de modelo, ya sea gráfico o matemático que les permite representar de una forma relativamente simple el problema en estudio. No obstante, existen ciertas limitaciones. Por ejemplo, las bases de datos que guardan información de los sistemas biológicos se dedican, por lo general a almacenar todo lo relacionado con los modelos, ya sean matemáticos o gráficos pero no almacenan información sobre los estudios que sobre estos

modelos se realizan. Esto implica que aunque podamos disponer de grandes repositorios de modelos, si queremos saber cómo funciona dicho sistema tenemos que repetir el trabajo que ya otros han realizado. Todos estos sistemas solo se centran en objetivos específicos para los cuales fueron diseñados.

Estas desventajas limitan en gran medida el desarrollo en esta área ya que los especialistas dedican gran parte de su tiempo a hacer trabajos que el sistema podría hacer perfectamente. Teniendo en cuenta todas estas razones surge el proyecto Simulación de Sistemas Biológicos (BioSyS), con el objetivo de crear una herramienta de propósito general que facilite la modelación, edición, simulación y análisis de los mismos.

La primera versión de BioSyS cuenta con herramientas para la modelación matemática, algoritmos para la realización de simulaciones distribuidas, una base de datos para almacenar los resultados de las simulaciones y algoritmos para el análisis de las simulaciones realizadas. A pesar de la independencia de estos algoritmos, los mismos no están implementados de forma modular, están integrados al sistema como un todo. Este hecho limita las posibilidades de escalabilidad del sistema, de incorporación de nuevos módulos y nuevas funcionalidades.

Es por ello que el equipo de trabajo de BioSyS se ha dado a la tarea de diseñar e implementar una versión 2.0 del sistema donde cada uno de estos algoritmos serán componentes independientes y van a representar módulos que finalmente serán librerías genéricas que podrán ser usadas por otros sistemas con las mismas características que BioSyS..

De esto se deriva el **problema científico**: ¿Cómo lograr la integración entre los componentes del sistema BioSyS?,

Que se enmarca en el **Objeto de estudio**: La Arquitectura de Software y el **Campo de Acción**: Arquitectura del Software para Sistemas Biológicos.

Estrechamente vinculado al campo de acción aparece como **Objetivo General:** Diseñar una propuesta de Arquitectura para BioSyS.

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

1. Definir estilos y patrones arquitectónicos a utilizar.
2. Diseñar las vista arquitectónicas del sistema.

3. Describir la arquitectura del sistema.
4. Evaluar el diseño arquitectónico propuesto.

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

1. Realización de búsquedas bibliográficas referentes al tema de la arquitectura del software.
2. Estudio de las tecnologías actuales en materia de arquitectura.
3. Análisis de los temas relacionados con los estilos y patrones arquitectónicos.
4. Diseño de las vistas del sistema.
5. Estudio de los lenguajes de descripción de arquitectura
6. Selección del lenguaje a utilizar en el diseño de las vistas de la arquitectura de BioSyS.
7. Análisis y Selección de los métodos para la evaluación arquitectónica.
8. Aplicación de los métodos para la evaluación arquitectónica.

Posibles resultados: Propuesta de una Arquitectura evaluada para BioSyS.

El contenido del trabajo está estructurado en 3 capítulos:

Capítulo 1: *Fundamentación Teórica.* En este capítulo se realiza un estudio del campo de la Arquitectura de Software, se definen estilos y patrones arquitectónicos y de diseño, existentes y más usados, así como también se describen los lenguajes de descripción arquitectónica y las principales herramientas y tecnologías que permiten diseñar una buena arquitectura de software.

Capítulo 2: *Diseño de la Arquitectura*. En este capítulo se define la estructura del sistema, sus metas y restricciones arquitectónicas y de diseño/implementación, se define como va a ser representada la arquitectura del mismo, se hace también una breve descripción de las vistas arquitectónicas y se muestran los diagramas correspondientes a cada una de ellas según el modelo “4+1” de Philippe Krucht.

Capítulo 3: *Evaluación de la Arquitectura propuesta*. El capítulo persigue como principal objetivo evaluar la arquitectura propuesta y para ello se hace un estudio previo de los modelos de calidad, las técnicas y los métodos de evaluación de arquitecturas software.

Capítulo 1

FUNDAMENTACIÓN TEÓRICA

CAPITULO 1: FUNDAMENTACIÓN TEÓRICA

Introducción

En el presente capítulo se realiza un estudio del campo de la Arquitectura de Software, se definen estilos y patrones arquitectónicos y de diseño, existentes y más usados, se describen los lenguajes de descripción arquitectónica y las principales herramientas y tecnologías que permiten diseñar una buena arquitectura de software.

1.1. Arquitectura de Software

En la medida que el tamaño de los sistemas crece, los algoritmos y las estructuras de datos dejan de convertirse en el mayor problema. El nuevo reto es diseñar y especificar la estructura global del sistema. La aparición de nuevas tecnologías en los últimos quince años, sumada a la complejidad creciente de los productos que se desarrollan, ha provocado que el concepto de "Arquitectura de software" tome una mayor trascendencia.

1.1.1. ¿Qué es Arquitectura de Software?

Actualmente se dice que la Arquitectura del Software (AS) es un campo específico de estudio para los investigadores e ingenieros del software, esto se debe a que a medida que aumenta la complejidad de los sistemas de software surgen nuevos aspectos del desarrollo de aplicaciones que hasta ese momento no se habían tenido en cuenta.

Existen varias definiciones de arquitectura y no es novedad que ninguna definición de la AS es respaldada unánimemente por la totalidad de los arquitectos sin embargo se ha acordado que la definición oficial sea la brindada por la IEEE 1471-2000, adoptada también por Microsoft:

"*La AS es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución.* " [2]

1.1.2. Surgimiento de la Arquitectura del Software

La primera referencia a la frase "arquitectura del software" fue hace muchos años: 1969, en una conferencia sobre técnicas de ingeniería del software organizada en Roma, sus antecedentes remontan aproximadamente hasta la década de 1960, sin embargo su historia no ha sido tan continua

como la del campo más amplio en el que se inscribe, la ingeniería de software. [2]

Por unos años, "arquitectura" fue una metáfora y no fue hasta fines de la década de 1980 y comienzos de la siguiente que la expresión arquitectura de software comienza a aparecer en la literatura. Anterior a los años 80, la palabra "arquitectura" fue usada ampliamente en el sentido de arquitectura del sistema o como la estructura física de un computador.

Las contribuciones más importantes surgieron en torno del instituto de Ingeniería de la Información de la Universidad Carnegie Mellon (CMU SEI), [2] 1999 fue un año clave para la arquitectura del software, el Open Group introdujo el Lenguaje de Marcas de la Descripción de la Arquitectura (ADML), un lenguaje basado en XML que provee soporte para amplios modelos de arquitectura compartidos y en ese mismo año emergieron nuevos métodos de evaluación de AS o se consolidaron: SAAM, BAPO, ATAM.

La AS constituye una práctica joven de apenas 12 años de trabajo constante, considerada como disciplina por mérito propio beneficiosa como marco de referencia para satisfacer requerimientos, una base esencial para la estimación de costos y administración del proceso y para el análisis de las dependencias y la consistencia del sistema. [2]

1.1.3. Clasificaciones de Arquitectura

Es imposible determinar cuántas arquitecturas existen dado que las combinaciones son infinitas, pero generalmente cada arquitectura se puede clasificar en centralizada o distribuida. [3]

Arquitectura Centralizada

Hace algún tiempo, casi todas las empresas manejaban una arquitectura única y centralizada que recibía el nombre de SNA (*System Network Architecture*) de IBM. Esta arquitectura contempla uno o varios equipos centrales muy grandes (mainframes) en el centro de cómputo, a los cuales se accedía a través de terminales "brutas". [3] La mayoría del proceso de datos sucedía en un solo sitio.

Arquitectura Distribuida

Las arquitecturas de sistemas distribuidos empezaron a obtener popularidad en la década de los 80 cuando el computador central fue reemplazado por un gran número de pequeños servidores instalados a lo largo y ancho de las empresas. La arquitectura distribuida nació como una respuesta a los altos costos y la baja flexibilidad que representaba la arquitectura centralizada. La arquitectura Cliente/Servidor es una arquitectura distribuida, los servidores almacenan los datos, y los clientes la analizan de acuerdo con los requerimientos del usuario. Un sistema distribuido es aquel en el que dos

o más máquinas colaboran para la obtención de un resultado. En todo sistema distribuido se establecen una o varias comunicaciones siguiendo un protocolo prefijado mediante un esquema cliente-servidor.
A modo de concluir se puede decir que si el proceso de datos se va a suceder en varias ubicaciones, se tiene una arquitectura distribuida. Si por el contrario, la mayoría del proceso se llevará a cabo en un solo sitio, se puede decir que se tiene una arquitectura centralizada. [3]

1.2. Estilos Arquitectónicos

Uno de los aspectos fundamentales de la AS lo constituye el razonamiento sobre estilos de arquitectura, sin duda la clave del trabajo arquitectónico tiene que ver con la correcta elección del estilo arquitectónico.

1.2.1. ¿Qué es un estilo arquitectónico?

Desde 1992 y hasta el 2001 innumerables han sido las definiciones de estilos arquitectónicos, sin embargo se percibe una unanimidad en todos ellos.

Buschmann define en 1996 estilo arquitectónico como una familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Se consideran como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo incluyendo información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación. [4]

Los estilos han sido históricamente bien consensuados, las discrepancias han sido en cuanto a la cantidad y sus clasificaciones, en realidad existen más clasificaciones que estilos de arquitectura.

A continuación se describen no todos los estilos arquitectónicos que se han propuesto sino los más representativos agrupados en estilos y sub-estilos.

1.2.2. Descripción de los estilos

Flujo de datos: estilo ideal para realizar transformaciones de datos dividiéndolos en pasos sucesivos. Esta familia de estilos enfatiza la reutilización y la modificabilidad. Es apropiado para sistemas que

implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote. [5]

- *Arquitecturas de tubería-filtros:* Se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes. La aplicación típica del estilo es un procesamiento clásico de datos: el cliente hace un requerimiento, el requerimiento se valida, un Web Service toma el objeto de la base de datos, se lo convierte a HTML, se efectúa la representación en pantalla. [5]

Llamada y retorno: el sistema se constituye de un programa principal que tiene el control del sistema y varios subprogramas que se comunican con éste mediante el uso de llamadas. Se representa haciendo una descomposición modular jerárquica, es decir, programa principal y subrutinas.

Una subrutina (componente) recibe el control y los datos de uno de los módulos ascendientes, lo transforma y lo pasa a los módulos descendientes. El retorno se realiza en sentido contrario. Facilita la modularidad.

Permite construir una estructura de programa relativamente fácil de modificar y ajustar. Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala. Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas. [5]

- *Modelo-Vista-Controlador (MVC):* en ocasiones se lo define más bien como un patrón de diseño o como práctica recurrente, y en estos términos es referido en el marco de la estrategia arquitectónica de Microsoft. Entre sus ventajas están las siguientes: soporte de vistas múltiple y adaptación al cambio. [5]

- *Arquitectura en Capas:* está definida 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. Permite la estandarización, reutilización de un mismo nivel en varias aplicaciones y el cambio de un nivel no afecta al otro. Existen cerca de cien patrones que son variantes del patrón básico de capas. Patrones de uso común relativos al estilo son Fachada, Adaptador, Puente y Estratégico. [5]

- *Arquitecturas orientadas a objetos:* nombres alternativos para este estilo han sido Arquitecturas

Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes del estilo se basan en principios orientados a objetos: encapsulamiento, herencia y polimorfismo. Las interfaces están separadas de las implementaciones. Las representaciones de los datos y las operaciones están encapsuladas en un tipo abstracto de datos u objeto. La comunicación entre los componentes es a través de mensajes. [5]

- *Arquitectura Basada en Componentes:* se centra en el diseño y construcción de sistemas computacionales que utilizan componentes de software reutilizables. Define la composición de software como "el proceso de construir aplicaciones mediante la interconexión de componentes de software a través de sus interfaces (de composición)", abogaba por la utilización de componentes prefabricados sin tener que desarrollarlos de nuevo.
  Un componente de software, es una unidad de composición que se puede comprar hecho con interfaces especificadas contractualmente y dependencias del contexto explícitas. [5]
  Esta arquitectura también se conoce como el paradigma de ensamblar componentes y escribir código para hacer que estos funcionen, permite la reutilización de código, simplifica las pruebas, el mantenimiento del sistema y proporciona ciclos de desarrollo más cortos.

Centrado de datos: resulta apropiado para sistemas que se centran en el acceso y actualización de datos en estructuras de almacenamiento que son compartidos por un número indefinido de componentes consumidores. Esa familia de estilos enfatiza la integrabilidad de los datos. Sub-estilos característicos de la familia serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra. [5]

- *Arquitecturas de Pizarra o Repositorio:* en esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Si es central la comprensión de los datos de la aplicación, su manejo y su representación, se considera una arquitectura de repositorio o de tipo de dato abstracto. Si los datos son perdurables, se considera repositorios. [5]

Código Móvil: se caracteriza por su enorme portabilidad. Las arquitecturas que siguen este estilo tienen una parte del sistema que pertenece al propio entorno nativo de la máquina que lo incluye, la otra parte no. Para realizar una comunicación entre ambas partes se necesita de un intérprete que permita la traducción. Hay que tener en cuenta a la hora de diseñar una arquitectura de este tipo cómo llevar a cabo esta traducción y cómo realizar la integración de la parte del sistema no incluida en la máquina física. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. [5]

- *Arquitectura de Máquinas Virtuales:* Esta arquitectura se conoce como intérpretes basados en tablas o sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación. [5]

Peer To Peer: esta familia se conoce también como 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. Sistemas peer-to-peer son sistemas distribuidos consistentes de nodos interconectados capaces de organizarse a sí mismo dentro de topologías de redes con el propósito de compartir recursos tales como contenidos, ciclos de CPU, almacenaje y ancho de banda. Miembros de la familia son los estilos basados en eventos, en mensajes, en servicios y en recursos. [5]

- *Arquitecturas Basadas en Eventos:* se han llamado también arquitectura de invocación implícita, y se vinculan históricamente con sistemas basados en actores y redes de conmutación de paquetes (publicación-suscripción). La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa un componente puede anunciar mediante difusión uno o más eventos. [5]

- *Arquitecturas Orientadas a Servicios (SOA en inglés):* este estilo construye toda la topología de la aplicación como una topología de interfaces, implementaciones y llamados a interfaces; es una relación entre servicios y consumidores de servicios, ambos lo suficientemente amplios como para representar una función de negocio completa. Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. [5]

- *Arquitecturas Basadas en Recursos:* este estilo define recursos identificables y métodos para acceder y manipular el estado de esos recursos. El caso de referencia es nada menos que la World Wide Web, donde los URLs identifican los recursos y HTTP es el protocolo de acceso. El argumento central es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación. [5]

Los estilos han llegado a la arquitectura de software para quedarse y han hecho mella tanto en el desarrollo de los lenguajes específicos de descripción arquitectónica como en los lenguajes más

amplios de modelado. [5] Se les denomina alternativamente patrones de arquitectura, se les confunde con patrones, ya sea de arquitectura o de diseño y sin embargo las diferencias entre ellos se hace notar.

Una vez que se han descrito los estilos podemos pasar a otro tema de sumo interés, los patrones.

1.3. Patrones.

1.3.1. ¿Qué es un patrón?

Buschmann define en 1996 patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. [6]

Patrón = < problema, contexto, solución > + <recurrencia, enseña, nombre>

En líneas generales, un patrón sigue el siguiente esquema:

Contexto: Es una situación de diseño en la que aparece un problema de diseño.

Problema: Es un conjunto de fuerzas que aparecen repetidamente en el contexto.

Solución: Es una configuración que equilibra estas fuerzas. Ésta abarca:

- Estructura con componentes y relaciones.
- Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc. [6]

Existen diversas clases de patrones: de análisis, de arquitectura (divididos progresivamente en estructurales, sistemas distribuidos, sistemas interactivos, sistemas adaptables), de diseño (conductuales, creacionales, estructurales), de organización o proceso, de programación y los llamados idiomas, entre otros. [6] Para la realización de este trabajo resulta apropiado analizar un poco más las clases: Patrones de Arquitectura y Patrones de Diseño.

1.3.2. Patrones de arquitectura

Partiendo de la definición de patrón, Buschmann propone los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución. [6]

Así mismo, plantea que los patrones arquitectónicos expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos.

Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación y que tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software. [6]

Algunos patrones arquitectónicos coinciden con los estilos hasta en el nombre con que se les designa. Según el muy influyente grupo de Buschmann (Buschmann, Meunier, Rohnert, Sommerlad y Stal) el catálogo de patrones arquitectónicos esta conformado por los patrones: Capas, Tubería-filtros, Pizarra, Bróker, Modelo-Vista-Controlador, Presentation-Abstraction-Control, Reflection (metanivel que hace al software consciente de sí mismo) y Microkernel (núcleo de funcionalidad mínima). [7]

A continuación una breve descripción de los mismos.

Capas: ayuda a estructurar aplicaciones que pueden descomponerse en grupos de sub-tareas, de forma que las tareas de cada grupo se encuentran en el mismo nivel de abstracción. Cada capa se encarga de un aspecto concreto de la comunicación y usa los servicios de la capa inferior. [7]

Tubería-filtros: provee una estructura para sistemas que procesan un flujo de datos. Cada etapa del proceso es encapsulada como un filtro. Los datos se pasan entre filtros adyacentes mediante Pipes. Recombinando filtros se obtienen familias de sistemas relacionados. [7]

Pizarra: útil para sistemas en que no se conoce una solución o estrategia determinista. Varios subsistemas especializados ensamblan su conocimiento para construir una posible solución parcial. [7]

Bróker: patrón arquitectónico aplicado a la estructuración de sistemas distribuidos, en los cuales es necesaria la interacción remota de componentes altamente desacoplados. Permite cambios a nivel dinámico, extensibilidad, reutilización, portabilidad (adaptabilidad) y además transparencia respecto a la ubicación. [7]

Modelo-Vista-Controlador o MVC: separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres componentes diferentes:

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

cambien según resulte apropiado. [7]

Presentation-Abstraction-Control: estructura una aplicación interactiva como una jerarquía de agentes que cooperan. Cada agente es responsable de un determinado aspecto de la funcionalidad y consta de tres componentes: Presentación, Abstracción y Control, que separan la interacción con el usuario de la funcionalidad central y la comunicación con otros agentes. [7]

Reflection: proporciona un mecanismo para cambiar la estructura y el comportamiento del sistema dinámicamente. La aplicación se divide en dos partes: 1 meta-nivel que hace al software autoconsciente y un nivel-base. [7]

Microkernel: separar un mínimo núcleo funcional de funcionalidad extendida y partes específicas del cliente. El Microkernel también sirve como punto donde engarzar estas piezas y coordinar su colaboración. [7]

Una vez que han sido tratados los temas estilos y patrones arquitectónicos y antes de pasar a los patrones de diseño es de suma importancia detenernos en sus diferencias para que no sean usados de forma errónea.

1.3.3. Estilos y Patrones Arquitectónicos, diferencias

Con la intención de hacer una comparación clara entre estilo arquitectónico y patrón arquitectónico, la tabla presenta las diferencias entre estos conceptos, construida a partir del planteamiento de Buschmann:

Estilo Arquitectónico	**Patrón Arquitectónico**
Sólo describe el esqueleto estructural y general para aplicaciones.	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación.
Son independientes del contexto en que puedan ser aplicados.	Partiendo de la definición de patrón, requiere de la especificación de un contexto del problema.
Cada estilo es independiente de los otros.	Depende de patrones más pequeños que contiene, patrones con los que interactúan, o de patrones que

	lo contengan.
Expresan técnicas de diseño desde una perspectiva que es independiente de la situación actual del diseño.	Expresa un problema recurrente de diseño muy específico, y presenta una solución para él, desde el punto de vista del contexto en el que se presenta.
Son una categorización de sistemas.	Son soluciones generales a problemas comunes.

Tabla 1. Diferencias entre estilos y patrones arquitectónicos.

Los estilos y patrones arquitectónicos ayudan al arquitecto a definir la composición y el comportamiento del sistema de software, y una combinación adecuada de ellos permite alcanzar los requerimientos de calidad. Existen además otro patrones que están contenidos dentro de los patrones de arquitectura y no por ello dejan de ser importantes, los patrones de diseño.

1.3.4. Patrones de diseño

Según Buschmann un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describen la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular [8]

Los patrones de diseño son menores en escala que los patrones arquitectónicos, están contenidos dentro de los patrones de arquitectura y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema, debido a que especifican a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema. [8]

Entre los patrones de diseño podemos encontrar los patrones surgidos por la banda de los 4 GoF y los patrones de asignación de responsabilidades Grasp.

1.3.4.1. Patrones de diseño GoF (Gans of Four, Grupo de los Cuatro)

Los patrones GoF constituyen patrones de diseño surgidos en 1995 por la banda de los cuatro, Erich Gamma, Richard Helm, Ralph Jonson y John Vissidess, promueven una expansión de la programación orientada a objetos y se pueden clasificar según su propósito en patrones de creación

(para creación de instancias), estructurales (relaciones entre clases, combinación y formación de estructuras mayores) y de comportamiento (interacción y cooperación entre clases).

Patrones de creación: Abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas. Según donde se tome dicha decisión podemos clasificar a los patrones de creación en patrones de creación de clase (la decisión se toma en los constructores de las clases y usan la herencia para determinar la creación de las instancias) y patrones de creación de objeto (se modifica la clase desde el objeto). [9]

Patrones estructurales: Tratan de conseguir que cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y, éstas están determinadas por las interfaces que soportan los objetos. Estudian como se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos. [9]

Patrones de comportamiento: Los patrones de comportamiento se utilizan para organizar, manejar y combinar comportamientos. Estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal. [9]

Algunos de los patrones básicos según su clasificación:

Creación:	Estructurales:	Comportamiento:
Factoría Abstracta	Adaptador	Intérprete
Instancia Única	Puente	Mediador
Prototipo	Composición	Memento
Método Factoría	Decorador	Observador
	Fachada	Estado
	Proxy	Estratégico

Tabla 2. Clasificación de Patrones GoF. [10]

Patrones de creación. Descripción.

- Patrón Factoría Abstracta: Proporciona una interfaz para crear familias de objetos o que dependen entre sí, sin especificar sus clases concretas.
- Patrón Instancia Única: Garantiza que una clase sólo tenga una instancia, y proporciona un punto de acceso global a ella.
- Patrón Prototipo: Especifica los tipos de objetos a crear por medio de una instancia prototípica,

y crear nuevos objetos copiando este prototipo.

- Patrón Método Factoría: Define una interfaz para crear un objeto dejando a las subclases decidir el tipo específico al que pertenecen.

Patrones estructurales. Descripción.

- Patrón 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.
- Patrón Puente: Desvincula una abstracción de su implementación, de manera que ambas puedan variar de forma independiente.
- Patrón Composición: para la creación de vistas compuestas. Combina objetos en estructuras de árbol para representar jerarquías de parte-todo, permite tratar a cada vista compuesta igual que una a una vista normal. Permite que los clientes traten de manera uniforme a los objetos individuales y a los compuestos.
- Patrón Decorador: Añade dinámicamente nuevas responsabilidades a un objeto, por ejemplo en una vista (el scroll), proporcionando una alternativa flexible a la herencia para extender la funcionalidad.
- Patrón Fachada: Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema se más fácil de usar.
- Patrón Proxy: Para distribuir la arquitectura (Modelo y Vista-Controlador) en diferentes emplazamientos. Proporciona un sustituto o representante de otro objeto para controlar el acceso a éste.

Patrones de comportamiento. Descripción.

- Patrón Intérprete: Dado un lenguaje, define una representación de su gramática junto con un intérprete que usa dicha representación para interpretar las sentencias del lenguaje.
- Patrón 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.
- Patrón Observador: Para el mecanismo de publicación y suscripción que permite la notificación de los cambios en el modelo a las vistas. 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.
- Patrón Estado: Permite que un objeto modifique su comportamiento cada vez que cambia su

estado interno. Parecerá que cambia la clase del objeto.

- Patrón Estratégico: Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
- Patrón Memento: Permite volver a estados anteriores del sistema, captura y restaura un estado interno de un objeto.

1.3.4.2. Patrones de diseño GRASP (patrones generales de software para asignar responsabilidades)

Los patrones GRASP son parejas de problema solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. [11]

Se pueden destacar 5 patrones principales dentro de los patrones Grasp:

1. Experto.
2. Creador.
3. Alta cohesión.
4. Bajo acoplamiento.
5. Controlador.

Patrón Experto: Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele útil en el diseño orientado a objetos. El cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. El patrón Experto asigna responsabilidades a las clases que tienen la información necesaria para cumplir con la responsabilidad. [11]

Beneficios:

1. Se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos y de fácil mantenimiento.
2. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clase “sencillas” y más cohesivas que son más fáciles de comprender y de mantener. Así se brinda soporte a una alta cohesión.

Patrón creador (Creator, GRASP de Craig Larman): Guía la asignación de responsabilidades

relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. Lo que define este patrón es que una instancia de un objeto la tiene que crear el objeto que tiene la información para ello. ¿Qué significa esto?, pues 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. [11]

Patrón Alta Cohesión: Mantiene la complejidad dentro de límites manejables, es decir asigna una responsabilidad de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme. [11]

Beneficios:

1. Mejoran la claridad y facilidad con que se entiende el diseño.
2. Se simplifica el mantenimiento y las mejoras de funcionalidad.
3. A menudo se genera un bajo acoplamiento.
4. Soporta mayor capacidad de reutilización.

Bajo Acoplamiento: Es la idea de tener las clases lo menos ligadas entre sí que se pueda. De tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. [11]

Beneficios:

1. No se afectan por cambios de otros componentes.
2. Fáciles de entender por separado.
3. Fáciles de reutilizar.

Patrón Controlador: Es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Asigna las responsabilidades de capturar los eventos del sistema a las clases. [11]

De acuerdo con el patrón Controlador, disponemos de las siguientes opciones:

1. El “sistema” global (controlador de fachada).
2. La empresa u organización global (controlador de fachada).
3. Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas).
4. Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados “Manejador< NombreCasodeUso >” (controlador de casos de uso).

En la decisión de cuál de las cuatro clases es el controlador más apropiado influyen también otros factores como la cohesión y el acoplamiento.

Beneficios:

1. Mayor potencial de los componentes reutilizables. Garantiza que la empresa o los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz.
2. Reflexionar sobre el estado del caso de uso. A veces es necesario asegurarse de que las operaciones del sistema sigan una secuencia legal o poder razonar sobre el estado actual de la actividad y las operaciones en el caso de uso subyacente.

Por ejemplo, tal vez deba garantizarse que la operación X no ocurra mientras no se concluya la operación Y. De ser así, esta información sobre el estado ha de capturarse en alguna parte, el controlador es una buena opción, sobre todo si se emplea a lo largo de todo el caso**.**

Luego de haber estudiado estos 3 conceptos (estilo arquitectónico, patrón arquitectónico y patrón de diseño), se puede ver qué relación existe entre ellos.

1.3.5. Relación entre estilo arquitectónico, patrón arquitectónico y patrón de diseño

Estilo Arquitectónico.

Descripción del esqueleto estructural y general para aplicaciones.

Es independiente de otros estilos.

Expresa componentes y sus relaciones.

Patrón Arquitectónico.

Define la estructura básica de una aplicación.

Puede contener o estar contenido en otros patrones.

Provee un subconjunto de subsistemas predefinidos, incluyendo reglas y pautas para su organización.

Es una plantilla de construcción.

Patrón de Diseño.

Esquema para refinar subsistemas o componentes.

Como se puede notar estilo arquitectónico, patrón arquitectónico y patrón de diseño son conceptos bien relacionados y sin embargo no por ello iguales. Un estilo tiene asociado varios patrones arquitectónicos y estos a su vez contienen uno o varios patrones de diseño que representan el menor nivel, están más próximos a la implementación. Existen notables diferencias entre ellos e incluso podemos citar algunas:

En cuanto a las representaciones visuales: los estilos se describen mediante simples cajas y líneas, mientras que los patrones suelen representarse en UML.

En cuanto al código: Si hay algún código en las inmediaciones de un estilo, será código del lenguaje de descripción arquitectónica o del lenguaje de modelado, de ninguna manera será código de lenguaje de programación, mientras que los patrones arquitectónicos, por su parte, se han materializado con referencia a lenguajes y paradigmas también específicos de desarrollo.

Posteriormente a esta reflexión se hace notar un nuevo concepto, los lenguajes de descripción arquitectónica, utilizados para expresar la estructura de las aplicaciones mediante un estilo arquitectónico.

1.4. Lenguajes de Descripción Arquitectónica

Actualmente, la forma más exacta de descripción del sistema es el código fuente o el código compilado. De aquí que el problema de la descripción de una arquitectura de software es encontrar una técnica que cumpla con los propósitos del desarrollo de software, en otras palabras, la comunicación entre las partes interesadas, la evaluación y la implementación. [12]

Hasta 1998 las arquitecturas de software fueron representadas por esquemas simples de cajas y líneas en los que la naturaleza de los componentes, sus propiedades, la semántica de las conexiones y el comportamiento del sistema como un todo, se definían de manera muy pobre. Aunque este tipo de representación ofrecía una imagen intuitiva de la construcción del sistema no se podía saber cual era la esencia de los componentes, su tipo, qué hacían, cómo se comportaban, de qué otros componentes dependían y de qué manera, así como qué significaban las conexiones y qué mecanismos de comunicación estaban involucrados. De ahí surgen los lenguajes de descripción arquitectónica (*Architecture Description Languages*) o ADL como solución a estos problemas. [12]

Un ADL debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones. Se estima deseable, además, que un ADL suministre soporte de herramientas para el desarrollo de soluciones basadas en arquitectura y su posterior evolución. [13]

1.4.1. ¿Que beneficios proporcionan los ADLs?

- Facilidad para introducir y mantener la información referente al sistema.
- Efectuar análisis a distintos niveles de detalle y establecer cambios de tipos sobre los componentes.
- Realizar análisis de desempeño, disponibilidad o seguridad, en tanto el lenguaje de descripción arquitectónica provea la facilidad para ello.
- Los componentes pueden ser refinados en la medida que sea necesario, para distintos tipos de análisis.

1.4.2. ADLs fundamentales de la arquitectura de software contemporánea

Existe una larga lista de lenguajes de descripción arquitectónica, todos se usan con el mismo propósito, muchos comparten elementos comunes de su antología, otros se diferencian en su disponibilidad para la plataforma Windows, las herramientas gráficas y su capacidad para generar código ejecutable. [13]
Algunos de ellos son:
Acme –Armani, Aesop, ArTek, C2 (C2, SADL, C2SADEL, xArch, xADL), CHAM, Darwin, Jacal, LILEANANA, MetaH/AADL, UML, Rapide, UniCon, Weaves, Wright, [13]

Descripción de algunos de los ADLs más conocidos:

CHAM (Chemical Abstract Machine): no es estrictamente un ADL, aunque algunos autores, aplicaron CHAM para describir la arquitectura de un compilador. CHAM proporciona una base útil para la descripción de una arquitectura debido a su capacidad de componer especificaciones para las partes y describir explícitamente las reglas de composición. Sin embargo, la formalización mediante CHAM es idiosincrática y (por así decirlo) hecha a mano, de modo que no hay criterios claros para analizar la consistencia y la completitud de las descripciones de configuración. Convendrá contar entonces con alguna herramienta de verificación. [13]

Darwin: es un lenguaje de descripción arquitectónica desarrollado por Jeff Magee y Jeff Kramer. Darwin describe un tipo de componente mediante una interfaz consistente en una colección de

servicios que son, ya sea provistos (declarados por ese componente) o requeridos (o sea, que se espera ocurran en el entorno). Las configuraciones se desarrollan instanciando las declaraciones de componentes y estableciendo vínculos entre ambas clases de servicios. [13]

Jacal: es un lenguaje de descripción de arquitecturas de software de propósito general creado en la Universidad de Buenos Aires, por un grupo de investigación del Departamento de Computación de la Facultad de Ciencias Exactas y Naturales. El objetivo principal de Jacal es lo que actualmente se denomina "animación" de arquitecturas, que sería poder visualizar una simulación de cómo se comportaría en la práctica un sistema basado en la arquitectura que se ha representado. [13]

UML: forma parte del repertorio conocido como lenguajes semi-formales de modelado, superaba la incapacidad de los primeros lenguajes de especificación orientada a objeto para modelar aspectos dinámicos y de comportamiento de un sistema introduciendo la noción de casos de uso. Proporciona herramientas para modelar requerimientos de comportamiento (diagramas de estado, de actividad y de secuencia). Aunque UML no es en modo alguno un ADL en el sentido usual de la expresión debido al hecho de que constituye una herramienta de uso habitual en modelado, importantes autoridades en ADLs han investigado la posibilidad de utilizarlo como metalenguaje. [13]

Rapide: se puede caracterizar como un lenguaje de descripción de sistemas de propósito general que permite modelar interfaces de componentes y su conducta observable. Su estructura es sumamente compleja, y en realidad articula cinco lenguajes: lenguaje de tipos describe las interfaces de los componentes, lenguaje de arquitectura describe el flujo de eventos entre componentes, lenguaje de especificación describe restricciones abstractas para la conducta de los componentes, lenguaje ejecutable describe módulos ejecutables y el lenguaje de patrones describe patrones de los eventos. [13]

En ocasiones se utilizan notaciones y formalismos como si fueran ADLs para la descripción de arquitecturas, algunos casos son CHAM y UML. A pesar que UML no califica en absoluto como ADLs y teniendo en cuenta las dificultades que brindan algunos Lenguajes de Descripción Arquitectónica y la posibilidad que brinda UML para modelar todo el ciclo de desarrollo del software y apoyar el modelo en capas y el Modelo Vista Controlador (MVC) y la colaboración entre usuarios, se decide utilizar dicho lenguaje de modelado para la descripción del sistema en desarrollo.

1.5. UML como Lenguaje de Modelado

Tal como indica su nombre, UML es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software. También para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir. Capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Permite representar en mayor o menor medida todas las fases de un proyecto informático: desde el análisis con los casos de uso, el diseño con los diagramas de clases y objetos, hasta la implementación y configuración con los diagramas de despliegue. [14]

1.5.1. Objetivos del UML

Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones:
Visualizar: UML permite expresar de una forma gráfica un sistema de forma que otro lo puede entender.
Especificar: UML permite especificar cuáles son las características de un sistema antes de su construcción.
Construir: A partir de los modelos especificados se pueden construir los sistemas diseñados.

Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado que pueden servir para su futura revisión. [14]

Este lenguaje nos indica cómo crear y leer los modelos, pero no dice cómo crearlos. Esto último es el objetivo de las metodologías de desarrollo. Su esencia son las vistas y los diagramas. Cada diagrama usa la anotación pertinente y la suma de estos diagramas crean las diferentes vistas.

Se usará UML como lenguaje de modelado además de ser usado también como lenguaje de descripción arquitectónica atendiendo a las características antes mencionadas.

1.6. Metodologías para el desarrollo de la Arquitectura del Software

Decidir que metodología utilizar se torna muy importante, pues como arquitectos de Software, se debe tener un plano en que apoyarnos. Las metodologías de desarrollo de software son un conjunto de procedimientos, técnicas y ayudas a la documentación para el desarrollo de productos software.
Si no se lleva una metodología de por medio, lo que obtenemos es clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos.

Las Metodologías más usadas son:

- Rational Unified Process RUP
- Programación Extrema (XP)
- Microsoft Solution Framework (MSF)
- OpenUp/Basic

Rational Unified Process (RUP).

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

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

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. [15]

Extreme Programing (XP).

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo, equipo pequeño y cuyo plazo de entrega era ayer. Consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto. [15]

Características de XP, la metodología se basa en:

1. Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.
2. Re fabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
3. Programación en pares: consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa. [15]

Microsoft Solution Framework (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. Se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. [15]

Características de MSF:

Adaptable: es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
Escalable: puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas a más.
Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología. [15]

OpenUp/Basic:

Es una versión más ágil de lo qué es el RUP, aplica propuestas iterativas e incrementales dentro del ciclo de vida, puede hacer frente a una amplia variedad de tipo de proyectos. Estructura el ciclo de vida del software en 4 fases, Inicio, Elaboración, Construcción y Transición. [16]

La metodología plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto, utiliza solo los procesos que sean necesarios y para ello necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no es necesario para que el proyecto no tenga errores, propone no utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo ser este modificado, mejorado y extendido. [16]
Se decide utilizar OpenUp/Basic como metodología de desarrollo, por estar diseñada para equipos pequeños, ser una metodología ágil que permite obtener un producto completo en breve espacio de tiempo y por ser un proceso de desarrollo mínimo, completo y extensible.

1.7. Herramientas CASE

Las herramientas CASE (*Computer Aided Software Engineering* o Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

Existe una larga lista de Herramientas CASE, algunas de ellas son:

Embarcadero ER/Studio 7.5: ofrece soporte XML para la generación de esquemas, permitiendo a los modeladores de datos y desarrolladores de aplicaciones colaborar en iniciativas de arquitectura orientada a servicios (SOA). Permite la conversión mejorada para incrementar la precisión durante la migración de modelos de datos desde herramientas tales como: CA ERwin, Data Modeler, Sybase PowerDesigner y muchas otras. [17]

Rational Rose: es la herramienta CASE que comercializan los desarrolladores de UML. Proporciona mecanismos para realizar la denominada ingeniería inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño. Es completamente compatible con la metodología RUP. Utiliza un proceso de desarrollo iterativo controlado donde el desarrollo se lleva a cabo en una secuencia de iteraciones. [18] Brinda muchas facilidades en la generación de la documentación del software que se está desarrollando. Es una potente herramienta para el desarrollo de software, sin embargo necesita de alta capacidad de procesamiento.

Visual Paradigm: Es una herramienta CASE que utiliza "UML": como *lenguaje* de modelaje. [19] Proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Está diseñada para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, apoya los estándares más recientes de las notaciones de Java y UML. Se integra con los IDEs NetBeans, JBuilder, Eclipse. [19]

Visual Paradigm ofrece:

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

Como herramienta CASE se usará Visual Paradigm por utilizar UML como lenguaje de modelado, apoyar los estándares de notaciones en Java y por integrarse con el IDE NetBeans.

1.8. Herramientas de soporte al desarrollo, Lenguajes y Tecnologías

1.8.1. Lenguajes de programación

Un lenguaje de programación es una construcción incremental del ser humano para expresar programas. Está constituido por un grupo de reglas gramaticales, un grupo de símbolos utilizables, un grupo de términos con sentido único y una regla principal que resume las demás.

Dentro de los lenguajes de programación podemos encontrar:

Java: es un lenguaje de programación orientado a objetos desarrollado por Sun MicroSystems a principios de los años 90, toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple. Java puede funcionar como una aplicación sola o como un "applet", que es un pequeño programa hecho en Java. [20] Fue diseñado para crear software altamente fiable y para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos.

C++: es un lenguaje orientado a objetos derivado del C, en realidad un superconjunto de C, nació para añadirle cualidades y características de las que carecía, es un lenguaje de propósito general basado en el C, al que se han añadido nuevos tipos de datos, clases, plantillas, mecanismo de excepciones, sistema de espacios de nombres, funciones inline, sobrecarga de operadores, referencias, operadores para manejo de memoria persistente, y algunas utilidades adicionales de librería (en realidad la librería Estándar C es un subconjunto de la librería C++). [21]

PHP (Hypertext Pre-processor o Procesador de HIpertextos): es un lenguaje de programación Open Source interpretado de alto nivel usado normalmente para la creación de páginas web dinámicas y ejecutadas en el servidor. [22] Permite la conexión a diferentes tipos de servidores de bases de datos tales como MySQL, Postgres, Oracle, ODBC, DB2, Microsoft SQL Server, Firebird y SQLite. PHP está diseñado para ser más seguro que Perl o C. Con PHP es posible utilizar los protocolos de red más famosos como IMAP, SMTP, POP3 e incluso HTTP, o utilizar los socket (enchufes).

Para el desarrollo del sistema se usará Java como lenguaje de programación por las características antes mencionadas y porque ya se cuenta con una primera versión del sistema que además esta implementada en java y eso facilitaría la reusabilidad del código.

1.8.2. Entornos de Desarrollo Integrado (IDE)

Un entorno de desarrollo integrado (IDE) es un programa compuesto por un conjunto de herramientas para un programador que puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Consisten en un editor de código, un compilador, un depurador y un GUI.

Se centrará la atención en solo algunos de los IDE para java.

NetBeans: es una herramienta de desarrollo Java, escrita puramente sobre la base de la tecnología Java, de modo que puede ejecutarse en cualquier ambiente que ejecute Java, lo cual, por supuesto, es casi en todas partes. Es un producto de código abierto, con todos los beneficios del software disponible en forma gratuita, ha sido examinado por una comunidad de desarrolladores e incluye una amplia integración de las características específicas de la tecnología Java que no se encuentran disponibles en otros conjuntos de herramientas de aplicaciones multiplataforma. Las características de NetBeans de flexibilidad entre plataformas, el cumplimiento de UML y la capacidad de administrar la complejidad ayudan a garantizar que las aplicaciones cumplan con los requerimientos específicos del negocio. [23] Su última versión, la 6.0, soporta varios sistemas de control de versiones como el Sistema Concurrente de Versiones y Subversion y además guarda versiones locales de los archivos.

Eclipse: es una plataforma de desarrollo Open Source basada en Java. Es un desarrollo de IBM cuyo código fue puesto a disposición de los usuarios. Constituye un marco y un conjunto de servicios para construir un entorno de desarrollo a partir de plug-in [24] Eclipse es ahora desarrollado por la Fundación Eclipse, comunidad de código abierto. Permite codificar, compilar y ejecutar aplicaciones desarrolladas en Java de forma amigable.

Se usará NetBeans 6.0 como entorno de desarrollo por integrarse con UML, ser multiplataforma, sin restricciones de uso y soportar varios sistemas de control de versiones.

1.8.3. Sistemas de Control de Versiones

Los sistemas de control de versiones son herramientas cuyo objetivo es administrar el código fuente y su evolución, de una forma u otra ir grabando ese proceso, y presentar al usuario esa información de forma útil y práctica. Suelen conocerse como VCS o CMS (Code Management System). Son ampliamente utilizados en los proyectos de desarrollo de software para mantener las versiones del código fuente. No obstante, su aplicación no está limitada a esta actividad, sino que permiten gestionar documentos, imágenes y ficheros de todo tipo.

Sistemas de control de versiones hay muchos y muy variados: CVS, Subversion, Darcs, StarTeam, etc.

CVS: conocido como *Concurrent Versions System* o Sistema Concurrente de Versiones, es una herramienta de configuración utilizada para almacenar el código fuente de grandes proyectos de software en un repositorio central, almacena todas las versiones de todos lo ficheros de tal forma que nunca se pierde nada y su utilización por varias personas es registrada. Proporciona una forma de combinar el código de dos o más personas que estén trabajando simultáneamente en el mismo fichero y mantiene una lista de los cambios entre cada versión de un fichero. [25]

SVN: también conocido como Subversion, es un sistema de control de versiones que se ha popularizado bastante, en especial dentro de la comunidad de desarrolladores de software libre. Está preparado para funcionar en red, y se distribuye bajo una licencia libre de tipo Apache. Surge con la intención de sustituir y mejorar al conocido CVS mantiene las ideas fundamentales de CVS pero suple sus carencias y evita sus errores. [26]

Las principales características de SVN y sus mejoras frente a CVS son:

1. Mantiene versiones no sólo de archivos, sino también de directorios
2. Mantiene versiones de los metadatos asociados a los directorios.
3. Mantiene los cambios en el contenido de los documentos, así como la historia de todas las operaciones de cada elemento, incluyendo la copia, cambio de directorio o de nombre.
4. Posibilita la elección del protocolo de red.
5. Soporta tanto ficheros de texto como de binarios.
6. Proporciona mejor uso del ancho de banda, ya que en las transacciones se transmiten sólo las diferencias y no los archivos completos.
7. Provee una mayor eficiencia en la creación de ramas y etiquetas que en CVS. [26]

Luego de ver las mejoras de SVN frente a CVS se puede decir que se utilizará Subversion como sistema de control de versiones.

1.8.4. Gestores de Base de Datos

Los Sistemas de Gestión de Base de Datos (SGBD) son un tipo de software muy específico, dedicado a servir de interfaz entre la base de datos, el usuario y las aplicaciones que la utilizan. Se compone de un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

Existen varios SGBD, algunos de ellos son PostgreSQL, MySQL y Oracle:

PostgreSQL: es un poderoso sistema manejador de bases de datos, es decir, un sistema diseñado

para administrar grandes cantidades de datos, que tiene la fama de ser la base de datos de código abierto Open Source más avanzada del mundo sin costos ni licencia. Se ejecuta en la mayoría de los Sistemas Operativos más utilizados en el mundo incluyendo, Linux, varias versiones de UNIX y por supuesto Windows. Usa una arquitectura proceso-por-usuario cliente/servidor. [27]
Las bases de datos de PostgreSQL no son archivos que se pueden subir a su sitio web como los de Access, sino que residen en un servidor de datos separado. Por ello debe utilizar algún programa cliente que le permita conectarse al servidor de datos con el fin de crear las tablas, subir datos, editar registros, etc. Aunque PostgreSQL está en un servidor Linux, usted puede gestionar sus bases de datos desde ordenadores con cualquier sistema operativo utilizando las aplicaciones adecuadas. [27]

MySQL: es un sistema de gestión de bases de datos relacional, licenciado bajo la GPL de la GNU, surgió cómo una necesidad de un grupo de personas sobre un gestor de bases de datos rápido, por lo que sus desarrolladores fueron implementando únicamente lo que precisaban, intentando hacerlo funcionar de forma óptima. Es por ello que, aunque MySQL se incluye en el grupo de sistemas de bases de datos relacionales, carece de algunas funcionalidades. Es probablemente, el gestor más usado en el mundo del software libre, debido a su gran rapidez y facilidad de uso. [28]

Oracle: es básicamente una herramienta cliente/servidor para la gestión de Bases de Datos. Es un producto vendido a nivel mundial, aunque la gran potencia que tiene y su elevado precio dificultan su uso, sólo se ve en empresas muy grandes y multinacionales, por norma general. Como es un sistema muy caro no está tan extendido como otras bases de datos, por ejemplo, Access, MySQL, SQL Server, etc. [29] Ofrece la posibilidad de crear y almacenar en la base de datos procesos programados en Oracle PL/SQL, lenguaje de programación potente.

Se seleccionó como gestor de base de datos PostgreSQL por administrar grandes cantidades de datos y ejecutarse en la mayoría de los sistemas operativos, y no se seleccionó MySQL por carecer de algunas funcionalidades así como tampoco se seleccionó Oracle por ser un producto vendido a nivel mundial lo cual dificulta su uso.

1.8.5. Framework o Marco de Trabajo

Un framework es un esquema (un esqueleto, un patrón) para el desarrollo y la implementación de una aplicación, o sea la forma de estructurar y normalizar la información de un modo conocido para poder manejarla: almacenarla, recuperarla, etc. Pueden llegar al detalle de definir los nombres de ficheros, su estructura y las convenciones de programación. [30]

Existen innumerables frameworks para las diferentes plataformas y lenguajes. Sin embargo la elección del framework concreto a utilizar vendrá marcada por:

1. El tipo de aplicación a desarrollar
2. El lenguaje de programación y otras tecnologías concretas: base de datos, sistema operativo, etc.

Dentro de Java, lenguaje de programación seleccionado existen varios frameworks asociados a diferentes ámbitos. Algunos ejemplos de frameworks para ámbitos específicos son:

Aplicaciones Web: Struts, "Java Server Faces", o Spring.
Webservices: Axis.
Interfaz de Usuario Web Dinámica: Ajax – DWR.
Interfaz Gráfica de Usuario: Swinn.
Procesos de Negocio: BPMS (WebSphere, AquaLogic, o Oracle).
Para aplicaciones de escritorio: Hibernate.

Se centra atención para el desarrollo del sistema en Hibernate, por ser el framework para Java específico para aplicaciones de escritorio, utilizándose como puente entre nuestro sistema y la base de datos y sus funciones irían desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes.

Hibernate: es una herramienta ORM (Mapeo de Objeto Relacional) para la plataforma Java de libre distribución bajo los términos de la GPL (Licencia Pública General Menor de GNU) que soporta la mayoría de los sistemas de bases de datos SQL y que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación. Su característica principal es el mapeo de clases en Java a tablas de una base de datos y de tipos de datos de Java hacia tipos de datos de SQL. Genera las sentencias SQL, libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias y para lograr la persistencia de los objetos (almacenarlos y recuperarlos) hace uso del API de Java JDBC. [31]

Hibernate soporta la implementación de clases de acceso a datos DAO (Data Acces Object) con las operaciones que se pueden hacer con ellas (insertar, eliminar, salvar, etc.) o también conocidas como interface DAO que son de suma importancia para nuestro sistema permitiendo el acceso a los datos.

Se usará Hibernate como herramienta ORM por ser el sistema una aplicación de escritorio, escrita puramente en Java y por ser una herramienta de libre distribución.

1.9. El Rol Arquitecto de software

Según la metodología OpenUp/Basic , el Arquitecto de Software 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. [32]

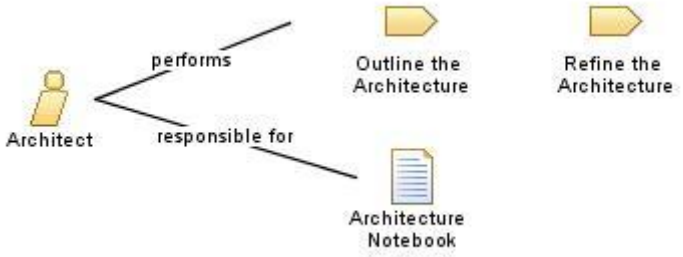

Figura 1. Rol Arquitecto.

1.9.1. Funciones del arquitecto

Realiza:

Descripción de la arquitectura: Analizar los requisitos y limitaciones arquitectónicos.

Refinamiento de la Arquitectura: Refinar la arquitectura a un nivel adecuado de detalle para apoyar el desarrollo.

Responsable de:

Cuaderno de Arquitectura.

Además Realiza:

Documento Visión, Busca y Describe los Requisitos, Detalla los Requisitos, Diseña una solución, Plan de Proyecto, Evalúa los Resultados y el Plan de Iteración.

Modifica:

Cuaderno de Arquitectura

Modelo de Diseño. [32]

1.9.2. Flujo de trabajo en los que participa

El arquitecto esta relacionado con varias disciplinas:

- Disciplina Requisitos: proporciona a la arquitectura importantes requisitos.
- Disciplina Desarrollo (diseño e implementación): formula y ejecuta la arquitectura.
- Disciplina Prueba: verifica la estabilidad y la exactitud de la arquitectura.
- Disciplina Gestión de Proyecto: los planes del proyecto y cada iteración.

1.9.3. Artefactos que genera.

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 aunque hace referencia a elementos de diseño arquitectónicamente significativos. [32]

1.10. Características del Sistema

BioSyS ha sido estructurado siguiendo un diseño modular, cuenta con una totalidad de 5 módulos: Modelación, Editor de Ecuaciones, Estimación de Parámetros, Simulación y Análisis, como centro del sistema se tiene una Base de Datos (BD) relacional, sobre la cual escriben y leen todos los demás módulos. El sistema realiza exploraciones intensivas en sistemas biológicos, que puedan ser descritos mediante sistemas de ecuaciones diferenciales, gestiona la información generada y permite hacer meta análisis sobre los resultados, éstas son algunas de las funcionalidades que cumplen los módulos que lo integra.

Además cuenta con dos técnicas de minería de datos, las técnicas de agrupamiento (clustering) y las de clasificación para facilitar el proceso de análisis de las simulaciones, ya que se hace difícil hacerlo de forma manual. Para la realización de las simulaciones se hace uso del asistente matemático MatLab. Se utilizan algoritmos implementados en Weka, para resolver el problema de la reutilización de código y de software. Además de los algoritmos de minería de datos, se ha implementado un algoritmo que permite hacer un análisis de bifurcaciones del sistema. BioSyS ha sido implementado siguiendo el paradigma de la programación orientada a objeto, específicamente en Java, lo que lo

convierte en un software multiplataforma. Cuenta con una plataforma de cálculo distribuido conocida como T-arenal donde existe un servidor intermedio que es el que se encarga de gestionar la utilización de los recursos disponibles en la red.

1.11. Conclusiones del Capítulo

Se definirá una Arquitectura Distribuida del tipo Llamada y Retorno, Basada en Componentes donde cada uno de los componentes deberá seguir una estructura en 3 Capas y en solo algunos casos seguirán el Modelo-Vista-Controlador. El sistema se desarrollará siguiendo la metodología de desarrollo de software OpenUp/Basic, así como Visual Paradigm como herramienta CASE y UML como lenguaje de modelado. Se hará uso del lenguaje de programación Java, como entorno de desarrollo para Java se consideró más adecuado el NetBeans 6.0 que soporta el control de versiones Subversion. Para el almacenamiento y gestión de los datos que se almacenan se consideró utilizar PostgreSQL. Para la gestión de la persistencia de los datos se usará Hibernate como herramienta ORM para Java. Para el acceso a los datos se utilizará DAO como patrón de diseño usado por el ORM seleccionado. Debido a que solo se va a hacer un diseño de alto nivel sin llegar a profundizar el comportamiento de los componentes del sistema se propone utilizar los patrones de diseño Decorador, y Fachada además de los patrones de asignación de responsabilidades Grasp.

Capítulo 2

DISEÑO DE LA ARQUITECTURA

CAPÍTULO 2: DISEÑO DE LA ARQUITECTURA

Introducción

En este capítulo se define la estructura del sistema, sus metas y restricciones arquitectónicas y de diseño/implementación, se define cómo va a ser representada la arquitectura del mismo, se hace también una breve descripción de las vistas arquitectónicas y se muestran los diagramas correspondientes a cada una de ellas según el modelo “4+1” de Philippe Krucht.

2.1. Estructura del sistema

El sistema va a estar conformado por una PC cliente que representa el puesto de trabajo del investigador y donde estará también ubicada la aplicación de escritorio como un único sistema donde estarán integrados todos los componentes que lo conforman, lo cual quiere decir que aunque estos sean componentes reutilizables independientes ahora dejan de serlo para liarse a un único sistema de propósito general. Las peticiones del cliente serán atendidas por el sistema por medio de interfaces gráficas definidas para cada una de sus funcionalidades, éstas interfaces serán levantadas por encima de la interfaz gráfica del sistema y desde esta máquina cliente se establecerá la conexión al servidor de base de datos una vez que se cargue el sistema de donde serán extraídos los datos necesarios para satisfacer al cliente. En caso que el cliente lo solicite el sistema establecerá la conexión con un servidor de cálculo distribuido para Java (T-arenal) el cual está conectado a una red de máquinas clientes que son las encargadas de realizar el trabajo solicitado y a medida que este se vaya realizando será enviado su resultado al servidor de base de datos, quedando satisfecho el cliente. De esta forma ha sido estructurado el sistema y posteriormente serán aclarados cada uno de los términos aquí mencionados.

En la figura 2 se muestra la forma en que se propone organizar el sistema y con el fin de lograr una mejor comprensión de la arquitectura propuesta a continuación una breve descripción de sus partes.

PC Cliente: representa el puesto de trabajo del investigador y el lugar donde va a estar instalado el sistema.

Servidor de Base de Datos (BD): representa el lugar físico donde están los datos asociados al sistema, permitirá la gestión de dicha información.

Servidor T-arenal: representa un servidor de cálculo distribuido, que será utilizado con ese propósito, tiene a su disposición una red de máquina para realizar el trabajo solicitado por el cliente.

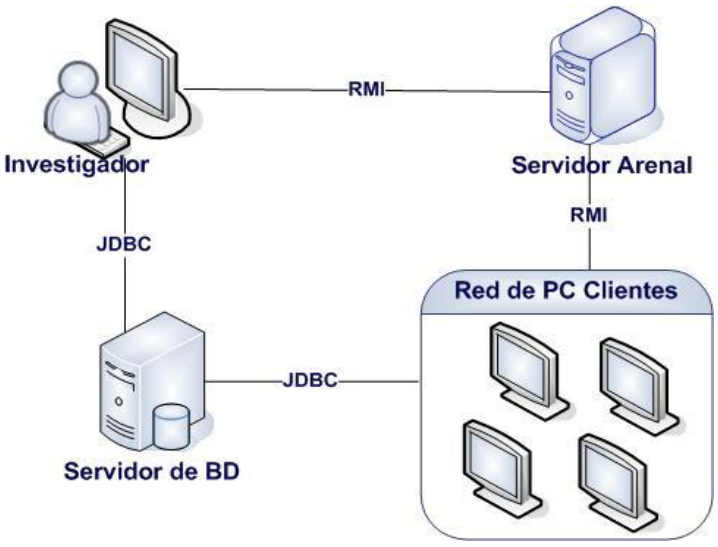

Figura 2. Estructura del Sistema BioSyS.

2.2. Representación Arquitectónica

Con el fin de conseguir el objetivo principal de la AS, aportar elementos que ayuden a la toma de decisiones y al mismo tiempo proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participan en el proyecto se construirán abstracciones, materializándolas en forma de diagramas comentados.

Estos diagramas se organizarán en vistas y representarán la arquitectura del sistema, para ello se seguirá uno de los modelos más conocidos, el “4+1” de Philippe Kruchten, que define cuatro vistas diferentes, siguiendo UML, que propone exactamente las mismas vistas.

A continuación se describen cada una de las vistas propuestas por el modelo “4+1” de Philippe Kruchten.

Vista de casos de uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de usos o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizar en paquetes, lo cual facilitaría la comprensión de la vista de casos de uso. Aquí se representa el diagrama de casos de uso de los mismos con una breve descripción de cada uno.

Vista lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representa los elementos de diseño más importantes para la arquitectura del sistema. Éste describe las clases

más importantes, su organización en paquetes y subsistemas y éstos a su vez en capas siempre que sea posible. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema.

Vista de procesos: Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

Vista de despliegue: Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Esto incluye la asignación de tareas provenientes de la vista de procesos en los nodos.

Vista de implementación: Esta vista describe la descomposición del software en capas si procede y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Esta contiene:

1. Una enumeración de los subsistemas.
2. Diagramas de componentes que ilustran la organización en capas y jerarquías de los subsistemas.
3. Dependencia entre subsistemas.

Cada una de ellas será representada más adelante, exceptuando la vista de procesos por ser esta una vista opcional y por aportar además elementos de poca importancia para la descripción de la arquitectura propuesta.

2.3. Metas y Restricciones

Las restricciones arquitectónicas no son más que afirmaciones sobre el sistema o algunas de sus partes, de manera que una violación de alguna de ellas hace al sistema menos deseable. En esta sección se mostrarán las restricciones que tienen mayor relevancia a nivel arquitectónico, siendo estas los requisitos no funcionales, se incluirán también las restricciones de diseño/implementación que afectan a la arquitectura.

Requisitos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener para que

sea atractivo, confiable, usable y seguro. Están vinculados a requerimientos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. [33]

Los requisitos no funcionales para el desarrollo del sistema se encuentran clasificados en varias categorías:

Software:

- La máquina virtual de java tiene que estar instalada en su versión 1,5 o superior.
- El sistema operativo a instalar en la PC donde se ejecuta el sistema, sin restricciones, por ser un software multiplataforma.
- El Sistema Operativo a instalar en las PC clientes del T-arenal con cualquier distribución de Linux.
- Asistente matemático MatLab instalado en cada una de las máquinas de la plataforma distribuida, en caso de no estarlo se usará java.
- Cliente del T-arenal instalado en cada una de las PCs que se utilizarán para la simulación distribuida.

Hardware:

- Se requiere de 256 MB de memoria RAM.
- Procesadores Pentium IV.
- Disco duro de 40 GB (puede variar dependiendo de la cantidad de información a almacenar).

Apariencia o Interfaz externa:

- Interfaz externa amigable, legible, interactiva, fácil de usar, profesional, clara y sencilla.

Seguridad:

- 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.

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 esta centrado en ello y en específico

en el diseño de las interfaces que harán posible el intercambio de datos de modo que resulte fácil para el usuario el uso del sistema.

Soporte:

- El sistema debe estar bien documentado de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.

Funcionalidad:

- Cada uno de los módulos del sistema debe proporcionar funcionalidades propias que satisfagan las necesidades establecidas y actuar bajo determinadas condiciones especificadas.

Mantenibilidad:

- El sistema permite implementar cambios, ya sea cualquier corrección, mejora o adaptación del software, por ejemplo: adicionar un nuevo módulo, sin efectos inesperados.

Rendimiento:

- El sistema debe ser capaz de formular respuestas lo más rápido posible. Para hacer más rápida la obtención de los resultados de las simulaciones se hace uso de la plataforma de cálculo distribuido, posibilitando agilidad en la obtención de los resultados.

Instalación:

- La instalación del sistema debe caracterizarse por su facilidad, claridad y sencillez. El sistema debe permitir la interacción con los demás módulos que componen la plataforma.

Portabilidad:

- El sistema debe funcionar en cualquier sistema operativo sobre el cual se haya instalado la máquina virtual de Java 1.5 o superior.

Confiabilidad:

- 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.

Ayuda y Documentación:

- El sistema constará con una ayuda en línea donde esté presente la documentación básica que posibilite comprender su funcionamiento y las funcionalidades generales a tener en cuenta para garantizar la utilización del mismo de manera eficiente.
- 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.

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 una de los componentes 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 (Modelo Vista Controlador).
- Para la implementación de esta segunda versión se reutilizará código implementado en la primera 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.

2.4. Diseño de las vistas

En este epígrafe se van a diseñar cada una de las vistas propuestas en la sección representación arquitectónica: vista de casos de uso, vista lógica, vista de despliegue y vista de implementación, además van a ser descritas cada una de ellas logrando describir la arquitectura del sistema.

2.4.1. Vista de casos de uso

Esta vista facilita la comprensión del sistema representando la relación actor- casos de uso, de ahí que sea de suma importancia conocer estos 2 nuevos conceptos: actor y casos de uso.

Actor del Sistema

Los actores de un sistema representan terceros fuera del sistema que interactúa con este u otros sistemas o hardware externo que se relacionan o interactúan con dicho sistema, no necesariamente tiene que ser una persona. Cada actor juega un rol determinado al interactuar con el sistema y diferentes usuarios pueden asumir el mismo rol de un actor. [33]

El sistema cuenta con un actor:

Nombre del Actor:	Descripción:
Investigador	Es el rol que representa la persona que interactúa con el sistema. Elabora el modelo gráfico del sistema biológico, estima sus parámetros, procesa sus datos, realiza las simulaciones y realiza los estudios correspondientes.

Tabla 3. Descripción del Actor del Sistema.

Caso de Uso del Sistema

Cada forma en que los actores usan un sistema se representa con un caso de uso, los casos de uso son "fragmentos" de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa un caso de uso especifica una secuencia de acciones que el sistema debe llevar a cabo, interactuando con sus actores donde se obtiene un resultado de valor para los actores. [33]

En la primeras iteraciones se eligieron unos pocos de casos de uso que van a ayudar en el diseño de la arquitectura (casos de uso arquitectónicamente significativos).

Se muestra la lista de estos casos de uso del sistema agrupados por módulos.

Módulo Modelación:

1. Generar el modelo matemático.
2. Guardar el modelo matemático del sistema biológico en formato MathML.
3. Guardar la información del sistema biológico desde el formato SBML.

Módulo Editor de Ecuaciones:

4. Editar Ecuaciones.
5. Exportar MathML.
6. Gestionar Biblioteca de expresiones.

Módulo Estimación de Parámetros:

7. Gestionar los datos de los experimentos.
8. Gestionar modelo matemático.
9. Gestionar ecuaciones de ligadura.
10. Transformar datos experimentales en formato del SED.
11. Gestionar el proceso de estimación.
12. Estimar los parámetros.

13. Gestionar los resultados obtenidos.

Módulo Simulación:

14. Gestionar Preferencias.
15. Cargar el modelo matemático.
16. Definir valores necesarios para realizar simulaciones.
17. Gestionar Simulación.

Módulo Análisis:

18. Mostrar Dinámicas de Población.
19. Realizar Clústers.
20. Realizar Clasificaciones.
21. Definir reglas y grupos de reglas.

Una vez que se conocen cuáles son los actores y casos de usos del sistema podemos pasar al diseño de la vista de casos de usos. Esta vista se organizó en paquetes permitiendo lograr una mayor comprensión de la misma, cada paquete contiene funcionalidades afines de la aplicación y en su interior cuenta con los casos de usos correspondientes más significativos así como su relación con el actor del sistema. La unión de todos y cada uno de los diagramas asociados a cada paquete complementan las funcionalidades más importantes del sistema.

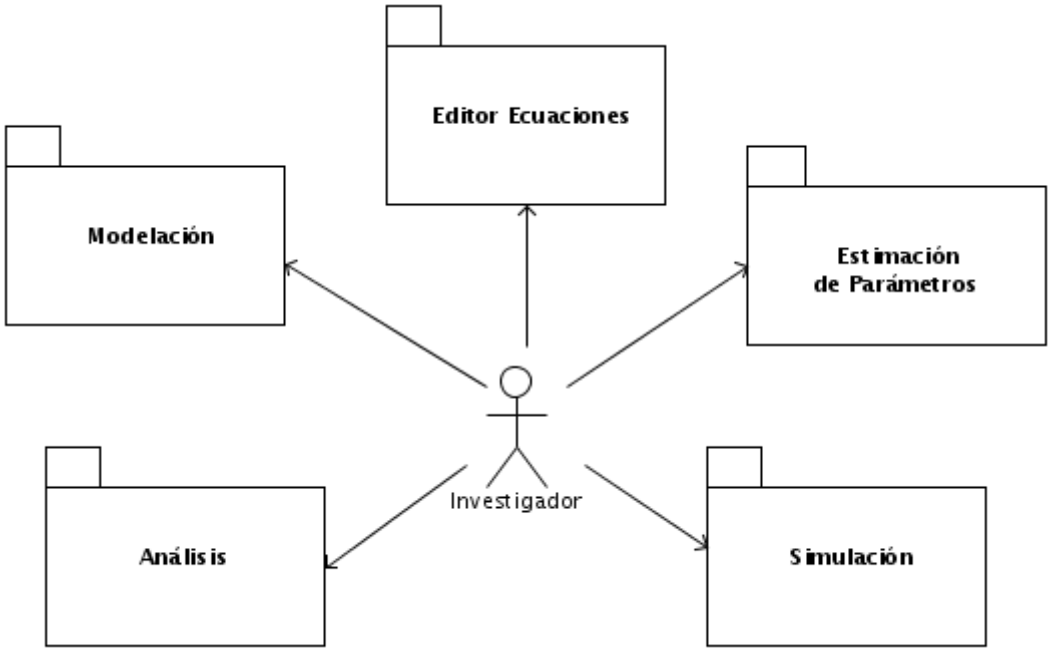

Figura 3. Vista de Casos de Uso.

A continuación se representan los diagramas de casos de usos correspondientes a cada uno de los paquetes representados en la Vista de Casos de Uso, así como la descripción de los casos de uso arquitectónicamente significativos asociados a cada diagrama.

Paquete Modelación:

Descripción de los Casos de Uso.

Generar el modelo matemático: Permite al investigador generar el modelo matemático a partir del modelo gráfico. El caso de uso es iniciado por el investigador cuando desea guardar el modelo matemático a alguno de los formatos definidos en el sistema, luego de haber especificado la ruta donde se deberá guardar el sistema genera el modelo matemático.

Guardar el modelo matemático del sistema biológico en formato MathML: Permite al investigador guardar el modelo del sistema biológico en formato MathML. El caso de uso es iniciado por el investigador cuando desea guardar el modelo matemático a partir del modelo gráfico, después de haber seleccionado la ruta donde será guardado, el sistema genera el modelo matemático en el formato MathML.

Guardar la información del sistema biológico desde el formato SBML: Permite al investigador guardar la información del sistema biológico en formato SBML. El caso de uso es iniciado por el investigador cuando el desea guardar el modelo matemático a partir del modelo gráfico, después de haber seleccionado la ruta donde será guardado, el sistema genera el modelo matemático en el formato SBML.

Diagrama de Casos de Uso.

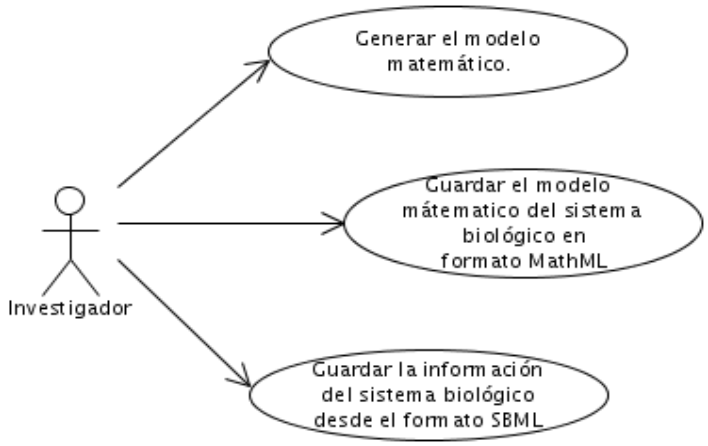

Figura 4. Diagrama de casos de uso correspondiente al paquete Modelación.

Paquete Editor de Ecuaciones:

Descripción de los Casos de Uso.

Editar Ecuaciones: El investigador puede solicitar crear una nueva ecuación o modificar una que ya ha sido generada anteriormente, el sistema muestra la información necesaria para cualquiera que haya sido la petición, en caso de crear el sistema verifica que la ecuación creada no exista en la BD, en caso contrario actualiza los datos.

Exportar MathML: En este caso de uso el investigador puede exportar las ecuaciones creadas anteriormente a un formato MathML, para ello debe seleccionar la ecuación que desea exportar, se deben tener ecuaciones definidas en la Base de Datos.

Gestionar Biblioteca de expresiones: Brinda la posibilidad al investigador de incluir de forma rápida las expresiones más utilizadas en el proceso de edición de sus modelos. El caso de uso se inicia cuando el investigador solicita agregar una expresión a la biblioteca, el sistema permite decidir el tipo de categoría a la que pertenece, mostrando las categorías, o permitiendo crear una nueva, antes se debe haber editado una expresión bien formulada.

Diagrama de Casos de Uso.

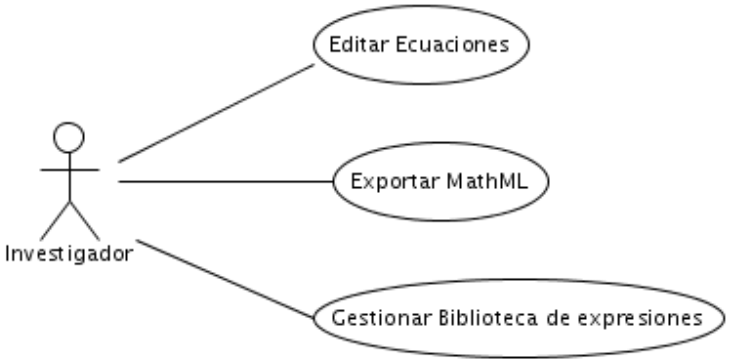

Figura 5. Diagrama de casos de uso correspondiente al paquete Editor de Ecuaciones.

Paquete Estimación de Parámetros:

Descripción de los Casos de Uso.

Gestionar los datos de los experimentos: El investigador puede, seleccionar, modificar o eliminar los datos del experimento. El caso de uso se inicia cuando el investigador solicita, eliminar, seleccionar o

modificar los datos de laboratorio de un experimento dado.

Gestionar modelo matemático: El caso de uso permite insertar un nuevo modelo matemático, visualizar la información de otros ya existentes, eliminarlos y modificarlos. El caso de uso comienza cuando el investigador desea gestionar los datos de un modelo matemático, se introduce un nuevo SED o se escoge uno que ya existe, este puede modificarse o no según desee.

Gestionar ecuaciones de ligadura: El caso de uso permite insertar nuevas ecuaciones de ligadura, visualizar la información de otras ya existentes, eliminarlas y modificarlas. El caso de uso comienza cuando el investigador desea gestionar los datos de las ecuaciones de ligadura.

Transformar datos experimentales en formato del SED: Permite transformar los datos experimentales en el formato de las variables del modelo matemático. El caso de uso comienza cuando el investigador transforma los datos de laboratorio en el formato de las variables que se obtienen del modelo matemático. Para eso se transforma los datos de laboratorio utilizando las ecuaciones de ligadura y se obtiene los valores en el formato de las variables del SED.

Gestionar el proceso de estimación: El investigador selecciona los rangos, valores, asistente matemático y método numérico necesarios para realizar la estimación de parámetros. El caso de uso se inicia cuando un investigador entra los valores de las condiciones iniciales, los valores de los parámetros que no varían, el rango de los valores de los parámetros que varían. Se selecciona el asistente matemático a utilizar y el método numérico y se define los valores de tiempo de integración.

Estimar los parámetros: El caso de uso estimar parámetros permitirá conocer cuáles son los valores de las variables que más coinciden con los datos de laboratorio. El caso de uso comienza cuando el investigador desea estimar los parámetros y solicita al sistema que mediante una función objetiva ya determinada calcule la estimación.

Gestionar los resultados obtenidos: El investigador solicita los resultados obtenidos y el sistema le devuelve los mejores resultados. El caso de uso comienza cuando el investigador desea gestionar los datos obtenidos en la estimación. Solicita los datos obtenidos al sistema y este devuelve una cantidad determinada de los mejores resultados.

Diagrama de Casos de Uso.

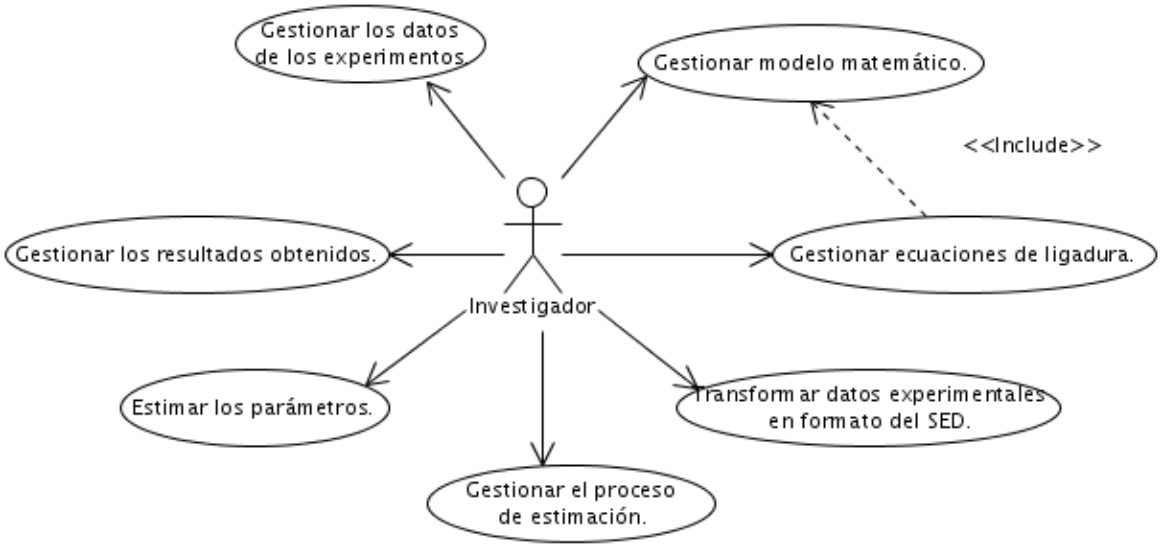

Figura 6. Diagrama de casos de uso correspondiente al paquete Estimación de Parámetros.

Paquete Simulación:

Descripción de los Casos de Uso.

Gestionar Preferencias: El Investigador, define opciones que brinda el sistema para realizar la simulación. El caso de uso se inicia cuando el actor debe decidir los términos en que se realizará la simulación, además de la herramienta matemática, método numérico a utilizar, máquinas que participarán en el proceso de simulado, y configuración local de la conexión y servidor T-arenal a utilizar. Como precondición se debe tener conexión a la Base de datos y que exista en esta un modelo matemático.

Cargar el modelo matemático: El Investigador, selecciona el modelo matemático, el caso de uso comienza cuando el investigador selecciona dicho modelo matemático sobre el cual se realizarán las simulaciones posteriores. Debe existir un modelo matemático credo con anterioridad.

Definir valores necesarios para realizar simulaciones: El Investigador define los valores necesarios para poder realizar simulaciones. El caso de uso comienza cuando el actor define los valores de: condiciones iniciales, parámetros y tiempo de integración. Anteriormente se debe haber cargado un modelo matemático.

Gestionar Simulación: El objetivo de este caso de uso es resolver un modelo matemático que describa a un sistema biológico, el caso de uso comienza cuando el Investigador manda a simular el sistema biológico, chequea el estado de la simulación, decide si detenerla o proseguir con la misma y por

último almacena los resultados obtenidos. Anteriormente se deben haber seleccionado las preferencias, además de haber introducido los parámetros y cargado el modelo matemático, finalmente se almacenan los resultados en la Base de Datos.

Diagrama de Casos de Uso.

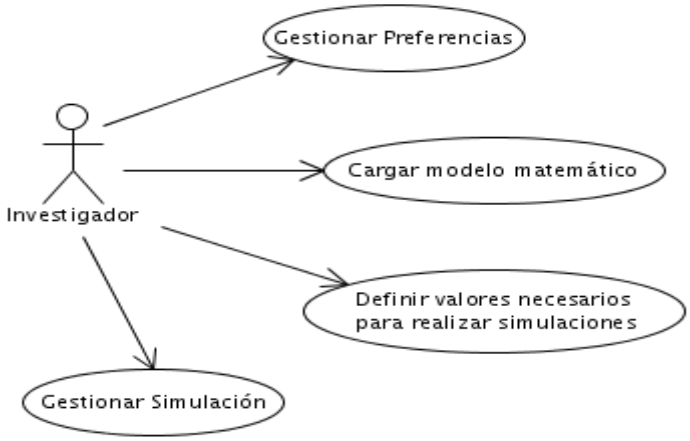

Figura 7. Diagrama de casos de uso correspondiente al paquete Simulación.

Paquete Análisis:

Descripción de los Casos de Uso.

Mostrar Dinámicas de Población: Analizar las salidas gráficas de las simulaciones que son las dinámicas de población, aquí se muestra el comportamiento de las poblaciones en diferentes intervalos de tiempo. El caso de uso se inicia cuando el investigador desea analizar las dinámicas de población correspondientes a un grupo de simulaciones de las cuales se guarda la serie temporal completa. El sistema muestra como resultado una gráfica donde se pueden hacer variaciones a los datos y a las propiedades gráficas de las mismas.

Realizar Clústers: Realizar el análisis de clustering a las simulaciones que están guardadas en la Base de Datos de la cual se almacena la serie temporal completa. El caso de uso se inicia cuando el investigador interactúa con la aplicación para realizar análisis de clustering. Como resultado el sistema muestra una gráfica con las simulaciones agrupadas por Clústers según el algoritmo seleccionado.

Realizar Clasificaciones: Clasificar simulaciones a partir de modelos generados de simulaciones previamente clasificadas. El caso de uso se inicia cuando el investigador interactúa con la aplicación para clasificar simulaciones. Como resultado el sistema muestra una gráfica con las simulaciones clasificadas partiendo de un modelo generado de simulaciones previamente clasificadas.

Definir reglas y grupos de reglas: Definir reglas y grupos reglas, así como las ecuaciones lógicas correspondientes a las mismas. El caso de uso se inicia cuando el investigador decide definir reglas, para el cual debe crear las reglas y los grupos de reglas correspondientes a las mismas, es decir, el nombre del grupo y el nombre de las reglas que pertenecen a ese grupo, así como definir las ecuaciones lógicas pertenecientes a cada regla.

Diagrama de Casos de Uso.

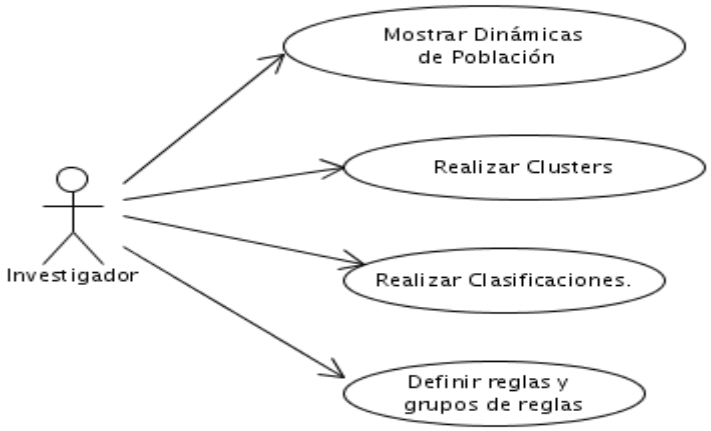

Figura 8. Diagrama de casos de uso correspondiente al paquete Análisis.

2.4.2. Vista lógica

Para la representación de esta vista se puede estructurar el sistema tanto en subsistemas como en paquetes de diseño. Es de suma importancia comprender estos dos conceptos para el entendimiento de esta vista.

Paquete de diseño: son un mecanismo de organización de elementos que subdividen el modelo en otros más pequeños que colaboran entre sí. Constituyen una colección de clases, realizaciones de CU, diagramas y otros paquetes. Son usados para agrupar elementos del modelo de diseño relacionados con propósitos organizacionales, si se necesita una semántica de comportamiento se deben usar subsistemas de diseño [34]

Subsistema de diseño: Una parte del sistema que encapsula comportamientos y expone un conjunto de interfaces, es un sistema por derecho propio cuya operación es independiente de los servicios provistos por otros subsistemas. Las interfaces visibles externamente y sus comportamiento son referidos como especificaciones del subsistema. Internamente, un subsistema es una colección de elementos del modelo (clases del diseño y otros subsistemas) que entienden la interfaz y el

comportamiento de la especificación del subsistema. [34]

Luego de comprender la diferencia de estos 2 conceptos y con el fin de organizar y facilitar el diseño, se realizó una división del sistema en: 5 subsistemas de diseño como partes lógicas coherentes y 2 clases interfaces que brindan el subsistema editor de ecuaciones y el subsistema simulación y que son usadas por el subsistema análisis.

Se establecen relaciones de dependencia entre los subsistemas debido a que existen funcionalidades propias de algunos que son utilizadas por otros y en el resto de los casos se utiliza la información asociada a otros subsistemas directamente desde la base de datos.

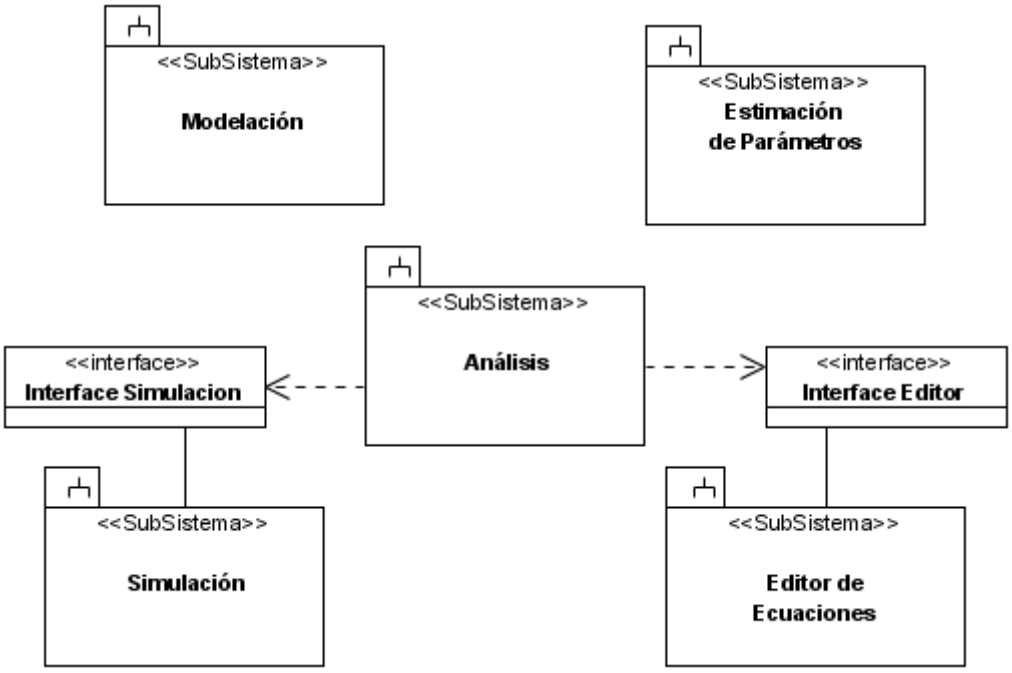

Figura 9. Vista Lógica de BioSyS.

Descripción de paquetes y subsistemas de diseño que conforman la vista lógica.

- Subsistema Modelación: Este subsistema es el responsable de generar las ecuaciones matemáticas estocásticas correspondientes al modelo gráfico estructurado por el investigador. Permite crear un modelo gráfico a partir de los estudios previos y genera automáticamente las ecuaciones asociadas a un modelo matemático el cual puede ser expresado a formato SBML o formato MathML, dicho modelo matemático es guardado en un fichero el cual podrá ser sobrescrito en caso de que el modelo gráfico fuere modificado. Dicho fichero será gestionado

por el editor de ecuaciones.

- Subsistema Editor de Ecuaciones: Este subsistema es el encargado de gestionar todo lo referente a la edición de las ecuaciones matemáticas. Permite editar (crear o modificar) ecuaciones asociadas a un modelo matemático y exportarlas al estándar de representación de fórmulas matemáticas, MathML. Incluye una biblioteca de expresiones que permite a los investigadores conservar los fragmentos de fórmula que más utilizan. Brinda mecanismos que auxilian al investigador en el proceso de análisis de homogeneidad dimensional de los sistemas de ecuaciones planteados.

- Subsistema Estimación de Parámetros: se encarga de estimar los parámetros en un sistema de ecuaciones diferenciales, utilizando datos de laboratorio y seleccionando un modelo matemático. Transformando los datos de laboratorio y el modelo matemático se obtienen una serie de datos a los que posteriormente se les realiza la estimación de parámetros. El resultado de esta combinación será de utilidad para el simulador.

- Subsistema Simulación: este subsistema se encarga de la gestión de simulaciones, dado un modelo matemático lo resuelve devolviendo una lista de soluciones. Permite cargar los modelos matemáticos, realizar las operaciones de forma local o distribuida, así como chequear el progreso de los cálculos. El resultado de las simulaciones será gestionado por el analizador de series temporales o paquete análisis.

- Subsistema Análisis: subsistema asociado al módulo de Análisis de series temporales, se encarga fundamentalmente del análisis de los resultados luego de generadas las simulaciones del modelo matemático, este análisis se puede hacer por diferentes vías: dinámicas de poblaciones, clustering, análisis por reglas y clasificaciones, el software Weka apoya estos tipos de análisis. Desde este subsistema se pueden editar ecuaciones y gestionar simulaciones asociadas a un modelo matemático.

- Interface Editor: clase interfaz brindada por el subsistema editor de ecuaciones y utilizada por subsistema de análisis para acceder a determinadas funcionalidades del editor que complementan el buen funcionamiento de este y para ello debe re-implementarlas.

- Interface Simulación: clase interfaz brindada por el subsistema simulación y utilizada por subsistema de análisis para acceder a determinadas funcionalidades del simulador que complementan el buen funcionamiento de este y para ello debe re-implementarlas.

La estructuración de los paquetes y subsistemas debe seguir un diseño estructurado en 3 capas, presentación (clases interfaces), negocio (clases controladoras) y acceso a datos (clases entidades), solo en algunos casos es innecesario acceder al controlador para llegar al modelo (administrador de los datos), por lo que se propone seguir como estructura el patrón Modelo-Vista-Controlador ambos patrones arquitectónicos usados para las estructuración de los paquetes y subsistemas de diseño.

2.4.3. Vista de despliegue

Esta vista constituye la visión física del sistema, representa un grafo de nodos unidos por conexiones de comunicación. A continuación se definen algunos conceptos importantes para la comprensión de esta vista.

Nodo: Un nodo es un objeto físico en tiempo de ejecución que representa un recurso computacional, generalmente con memoria y capacidad de procesamiento. [35]

Servidores de Objetos: proporcionan objetos distribuidos que pueden accederse a distancia, los cuales son tratados por el programador como si estuvieran en su computadora local. Esta tecnología libera a los programadores de la programación de bajo nivel basada en protocolos requerida para acceder a otras computadoras de una red. [35]

Objeto distribuido: es aquel que reside en una computadora, normalmente un servidor, en un sistema distribuido. Otras computadoras del sistema pueden enviar mensajes a este objeto como si residiera en su propia computadora. El software del sistema se hará cargo de: localizar el objeto, recoger los datos que se requieren para el mensaje y enviarlos a través del medio de comunicación que se utiliza para el sistema. [35]

Servidores de Base de Datos: son computadoras que almacenas grandes colecciones de datos estructuradas, llevan a cabo todo el procesamiento donde el cliente envía las consultas y el servidor la lee, la interpreta y visualiza una respuesta en un objeto de salida. [35]

Protocolo de comunicación JDBC: JDBC es una especificación de un conjunto de clases y métodos de operación que permiten a cualquier programa Java acceder a sistemas de bases de datos de forma homogénea. La aplicación de Java debe tener acceso a un driver JDBC adecuado. Este driver es el que implementa la funcionalidad de todas las clases de acceso a datos y proporciona la comunicación entre el API JDBC y la base de datos real.

Protocolo de comunicación RMI: Esta es la tecnología asociada al lenguaje de programación de Java. Es un enfoque Java puro en el que solo los programas escritos en ese lenguaje se pueden comunicar con un objeto RMI distribuido. [35]

El sistema será desplegado de la siguiente manera:

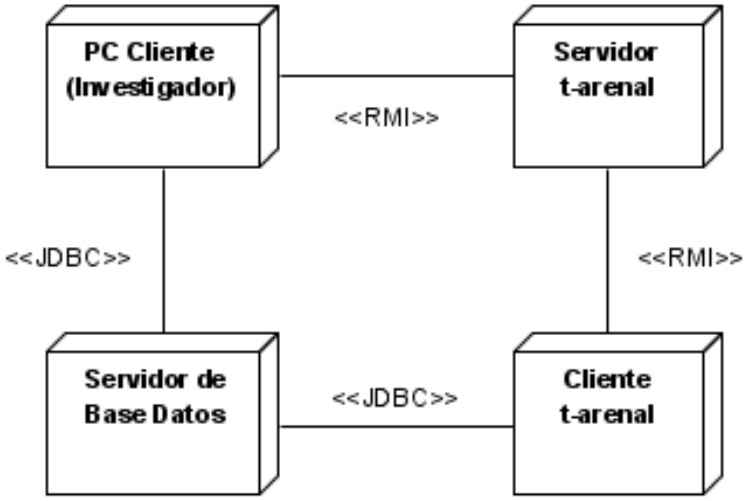

Figura 10. Vista de Despliegue del Sistema.

Descripción de los nodos que conforman el diagrama de despliegue.

Nodo PC Cliente: representa la máquina cliente asignada al investigador, que en este caso es el cliente del sistema, permite gestionar las peticiones del cliente, estará conectada a un servidor de base de datos por medio de JDBC y al T-arenal por medio de RMI.

Nodo T-arenal Server: representa un servidor de objetos o Servidor T-arenal, recibe las peticiones del cliente, las divide en estaciones de trabajo y se las envía a sus máquinas clientes por RMI. Por ser una plataforma de cálculo distribuido permite una mayor agilidad en los resultados.

Nodo T-arenal Cliente: representa las máquinas que están a disposición del T-arenal (servidor de objetos), se representó con solo una PC pero en realidad no se sabe la cantidad de PC Clientes que se van a utilizar, estas son las encargadas de realizar el trabajo enviado por él y para ello necesitan tener instalada de manera opcional la herramienta de cálculo MatLab, en caso de no estar el MatLab, se usa Java. Luego de realizar el trabajo lo envía particionado al servidor de base de datos por JDBC.

Nodo Servidor de BD: representa un servidor de base de datos utilizado para el almacenamiento de los datos de la aplicación y para lograr la conexión del sistema con la base de datos se utiliza JDBC como

protocolo de comunicación asociado al ORM Hibernate. Se encarga también de organizar la información enviada por la PC clientes del T-arenal.

2.4.4. Vista de implementación

Los diagramas de despliegue y componentes conforman lo que se conoce como vista de implementación al desplegar los componentes a construir, su organización y dependencia entre nodos físicos en los que funcionará la aplicación. El diagrama de despliegue ya ha sido representado en el anterior sub-epígrafe, solo queda la representación del diagrama de componentes para la conformación de esta vista.

El diagrama de componentes va a ser representado en términos de subsistemas de implementación para ser usado en la estructuración de la vista de implementación. Muestra un conjunto de elementos tales como componentes, subsistemas de implementación y sus relaciones de dependencia.

Componente: 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 software puede ser desde una subrutina de una librería matemática, hasta una clase, una base de datos, un archivo DLL, un JavaBeans, o incluso una aplicación que pueda ser usada por otra aplicación por medio de una interfaz especificada. [36]

Subsistema de Implementación: Una colección de componentes y otros subsistemas de implementación usados para estructurar el modelo de implementación y dividirlos en pequeñas partes que pueden ser integradas y probadas de forma separada, incluyen dependencias y otras informaciones. Ejemplos de subsistemas de implementación: un paquete en Java, un proyecto en Visual Basic, un paquete en el Rational, etc. [36]

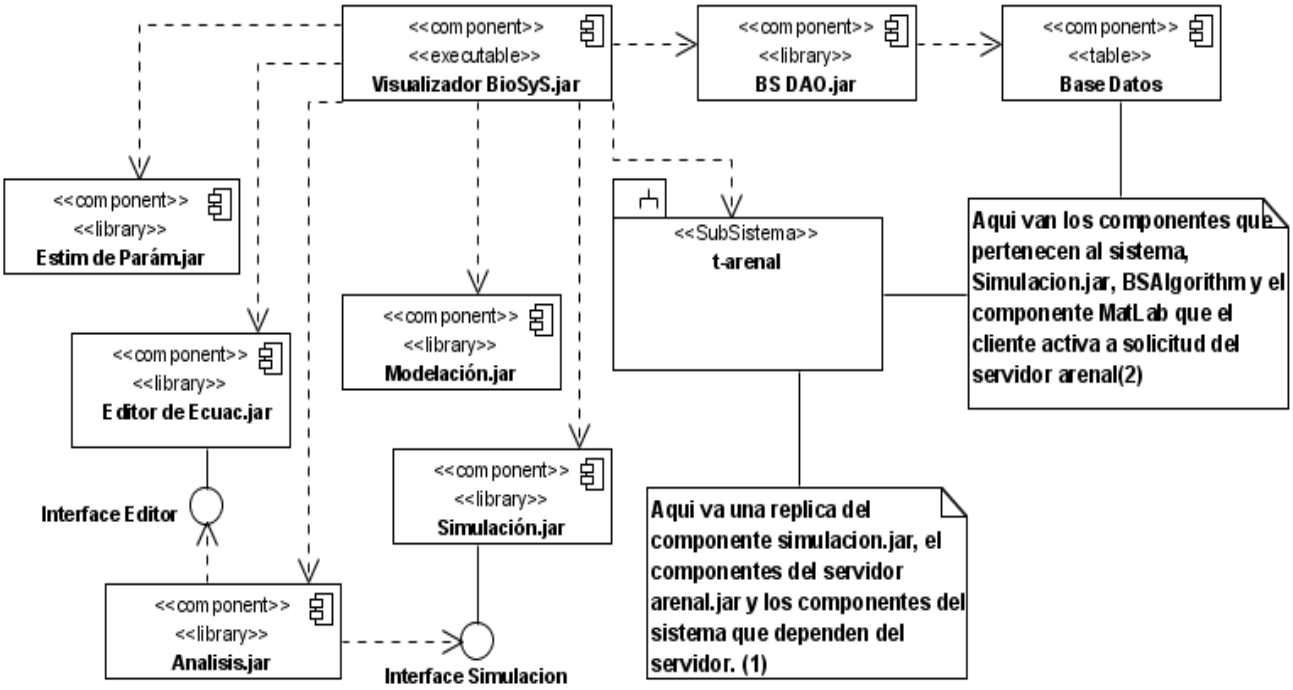

Figura 11. Diagrama de Componentes.

En este diagrama de componentes se establecen relaciones de dependencia entre componentes y entre componentes y subsistemas de implementación, entre el componente ejecutable BioSyS.jar y cada uno de los componentes.jar, entre el componente Analisis.jar y los componentes Editor de Ecuaciones.jar y Simulacion.jar y entre el componente BioSyS.jar y el subsistema T-arenal.

Existen componentes que no fueron representados en el diagrama y son justamente aquellos que van a estar ubicados dentro del subsistema T-arenal.jar, que serían: una réplica del componente Simulación.jar para el caso en que las simulaciones seas del tipo distribuidas, este componente solo se diferencia del que está en la PC cliente en su ubicación, además el componente ejecutable T-arenal.jar y los componentes BSAlgorithm y BSDataManager que representan clases que heredan de 2 clases brindadas por el componente ejecutable T-arenal.

Para una mayor comprensión de lo antes explicado ver figura 12, muestra los componentes que están ubicados en el subsistema T-arenal.

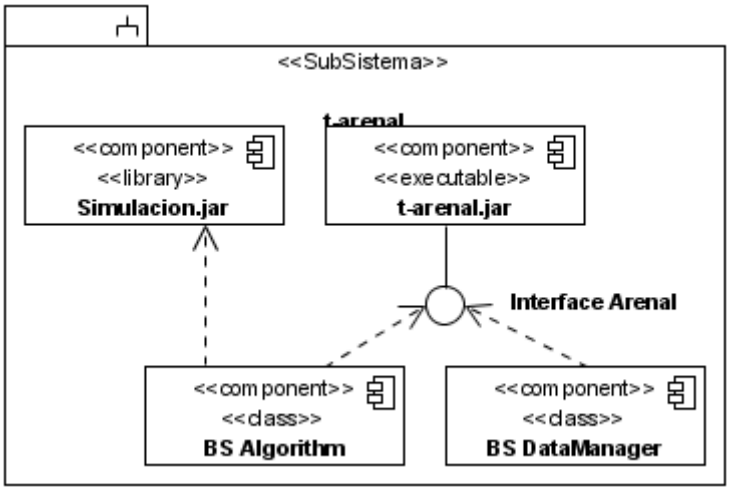

Figura 12. Representación del comentario1 del diagrama de componente.

En esta figura (12) se puede ver como una vez que se solicita la realización de simulaciones distribuidas, el sistema establece conexión con el componente T-arenal.jar, este genera por medio de una interface un nuevo Algorithm y un nuevo DataManager cada vez que se hace una solicitud de trabajo a la plataforma de cálculo distribuido, en este caso especificados con BS de Simulador Biológico, para diferenciarlos de los demás que ya han sido creados para la realización de otros trabajos, luego el BSAlgorithm necesita del componente Simulación.jar para la realización del mismo y una vez que ocurre todo ese proceso el BSAlgorithm es enviado junto al componente Simulacion.jar a la PC cliente del t-arenal donde se ejecuta el MatLab si se solicita su uso, en caso contrario se usará java, y terminado el trabajo el BSAlgorithm envía el resultado del mismo al servidor de BD.

Descripción de los componentes y subsistemas de implementación representados en el diagrama de componentes.

Componente Visualizador BioSyS.jar: permite mantener el control del sistema y la gestión de las funcionalidades del sistema por medio de interfaces, representa la cara principal del sistema, ya que las clases que contiene son las que permiten que se logre un buen funcionamiento del sistema, por sí solo no contiene funcionalidad, depende totalmente de los componentes.jar para lograr el resultado esperado.

Componentes.jar: representan cada una de las funcionalidades del sistema, es decir los .jar correspondientes a cada uno de los módulos del sistema, la comunicación entre ellos se establece por medio de interfaces, así como con la interfaz principal.

Subsistema T-arenal: representa otra aplicación que va a ser usada por el sistema y que necesita del subsistema Simulación.jar para lograr brindar las funcionalidades con que fue destinado su uso, para ello utiliza una clase DataManager que es la encargada de fragmentar el trabajo y enviárselo junto a la clase Algorithm encargada de realizar el trabajo a la maquinas clientes del T-arenal, las cuales se levantan el MatLab en caso que sea necesario.

Componente DataSourse DAO: representa aquellos objetos de acceso a datos (Patrón DAO).

Componente Base Datos: representa la base de datos física, a ella aceden todos los subsistemas por medio del API de Java JDBC, protocolo de comunicación que permite el acceso a los datos asociado al ORM Hibernate.

Las interface: representan clases que contienen operaciones que son brindadas por los componentes simulación.jar y editor de ecuaciones.jar y utilizadas por el componente de análisis.jar, este componente necesita re-implementar estas funcionalidades para hacer uso de ella y complementar las suyas propias.

Luego de haber conformado el diagrama de componentes y el diagrama de despliegue, solo queda conformar la vista de implementación asignando los subsistemas de implementación y componentes software a los nodos físicos. Aclarar que en esta vista no van a ser representados tampoco los componentes pertenecientes al subsistema T-arenal, para ello ver la figura 12.

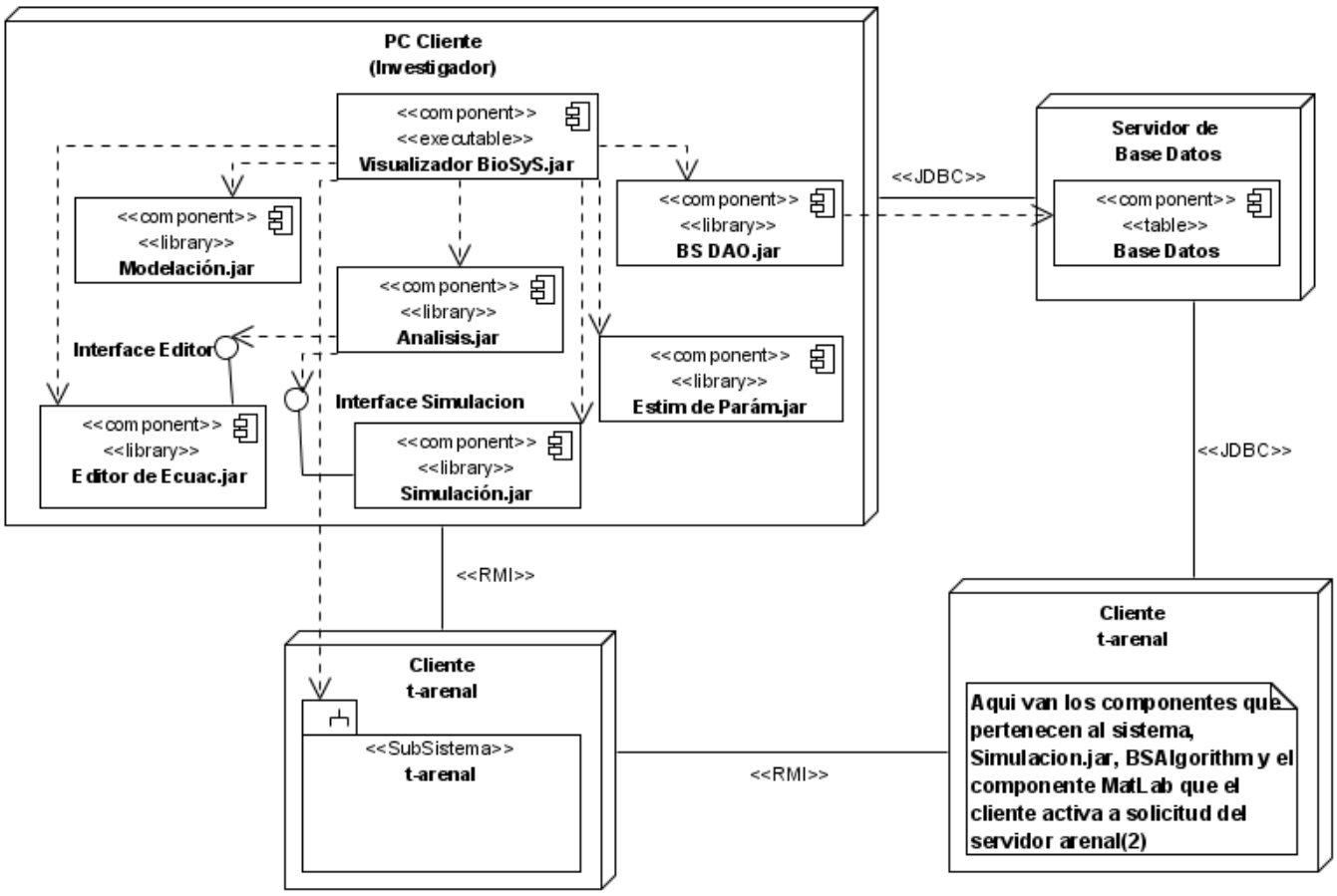

Figura 13. Vista de Implementación.

2.5. Conclusiones del Capítulo

El análisis de las metas y restricciones arquitectónicas permitió identificar los elementos críticos o sensibles en cuanto a diseño/implementación indispensables para el buen funcionamiento del sistema. La representación de la arquitectura del sistema a través de las 4 vistas arquitectónicas propuestas por el modelo “4+1” de Philippe Kruchten permitieron lograr una visión general del mismo y su evolución. Con el diseño de las 4 vistas propuestas por la metodología OpenUp/Basic se logró proporcionar una mayor comprensión del sistema a los desarrolladores que lo construirán.

Capítulo 3

EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

CAPÍTULO 3: EVALUACIÓN DE LA ARQUITECTURA PROPUESTA

Introducción

El capítulo persigue como principal objetivo evaluar la arquitectura propuesta y para ello se hace un estudio de los modelos de calidad, centrando atención en el que ha sido adoptado para arquitecturas software, además se estudiaron las técnicas y los métodos de evaluación de arquitecturas software, se define también qué es evaluación, porqué es importante evaluar la arquitectura y cuál es el momento propicio para ello.

En el mundo actual cada vez se exigen soluciones software de mayor calidad y con menores costes y tiempos de desarrollo. La arquitectura de software posee un gran impacto sobre la calidad de un sistema, la garantía de una arquitectura correcta cumple un papel fundamental en el éxito general del proceso de desarrollo, además del cumplimiento de los atributos de calidad del sistema. De ahí que la evaluación constituya un punto de gran importancia, errores en ella, pueden traer consigo que el proyecto fracase.

3.1. ¿Qué es una evaluación?

La evaluación es un estudio de factibilidad que pretende detectar posibles riesgos, así como buscar recomendaciones para contenerlos. La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Tiene como objetivo saber si la arquitectura puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders.

3.2. ¿Por qué evaluar una Arquitectura Software?

Sobran justificaciones para evaluar una arquitectura. La evaluación permite mejorar la visión de los procesos críticos y validar las decisiones de diseño que se tomaron. Permite valorar los atributos no funcionales (disponibilidad, desempeño, seguridad, interoperabilidad, mantenibilidad, presupuesto) sin esperar a que el software se construya [37], así como tomar algunas decisiones, por ejemplo: si se puede seguir el proyecto con las áreas de debilidad dadas en la evaluación o si hay que reforzar la AS o si hay que comenzar de nuevo toda la AS. Sirve para prevenir todos los posibles desastres de un diseño que no cumple con los requerimientos de calidad y para saber que tan adecuada es la AS

diseñada para el sistema. [38] Cuanto más temprano se encuentre un problema en un proyecto de software, mejor. Realizar una evaluación de la arquitectura es la manera más económica de evitar desastres.

3.3. ¿Cuándo es recomendable evaluar la arquitectura?

Aunque es posible realizarla en cualquier momento existen dos posibilidades distintas: **temprana** y **tardía**. [38]

Si se va realizar una evaluación temprana no es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo. Resulta interesante resaltar el planteamiento de Bosch quien plantea que es posible efectuar decisiones sobre la arquitectura a cualquier nivel, puesto que se pueden imponer distintos tipos de cambios arquitectónicos, producto de una evaluación en función de los atributos de calidad esperados. Es necesario destacar que tanto Bosch como Kazman establecen que mientras mayor es el nivel de especificación, mejores son los resultados que se obtienen de la evaluación.

La evaluación tarde consiste en realizar la evaluación de la arquitectura cuando ésta se encuentra establecida y la implementación se ha completado, es decir luego de la fase de implementación justo en el momento de la adquisición de un sistema ya desarrollado. Se considera muy útil la evaluación del sistema en este momento, puesto que puede observarse el cumplimiento de los atributos de calidad asociados al sistema, y cómo será su comportamiento general.

3.4. ¿Quiénes están involucrados en la evaluación?

Están involucrados en el proceso de evaluar la arquitectura de un sistema:

- Equipo de Evaluación.
- Stakeholders.

3.5. ¿Qué resultado produce la evaluación de una Arquitectura?

La evaluación de una arquitectura no produce resultados cuantitativos, solo ayuda a encontrar debilidades en el sistema. Produce una lista priorizada de los atributos de calidad requeridos para la arquitectura que se esta evaluando y los riesgos y no riesgo así como escenarios de negocio evaluados de acuerdo a los atributos de calidad.

3.6. ¿Qué beneficios aporta la evaluación de una Arquitectura?

Evaluar la arquitectura de un sistema trae consigo varios beneficios:

- Financieros.
- Fuerza una mejora a la documentación de la arquitectura.
- Recolecta los fundamentos y las decisiones arquitectónicas tomadas (las evidencias).
- Detecta problemas de forma temprana.
- Valida los requerimientos. Debido a que se evalúa si la arquitectura cumple con los requerimientos siempre sales discusiones sobre estos.
- Prioriza Requerimientos.
- Mejora la arquitectura.

3.7. Modelos de Calidad

Los modelos calidad resultan de utilidad para efectuar la medición del nivel de complejidad de un sistema software, han sido establecidos debido a la organización y descomposición de los atributos de calidad en efectos de la evaluación de la calidad arquitectónica.

Desde 1970 hasta ahora han sido propuestos varios modelos de calidad, algunos de ellos son: McCall (1977), Dromey (1996), FURPS (1987), ISO/IEC 9126 (1991) e ISO/IEC 9126 adaptado para arquitecturas de software, propuesto por Losavio (2003).

Se centra atención en el modelo ISO/IEC 9126 adaptado para arquitecturas de software en el año 2003 por ser un modelo que se basa en los atributos de calidad que se relacionan directamente con la arquitectura: funcionalidad, confiabilidad, eficiencia, mantenibilidad y portabilidad.

3.7.1. Modelo ISO/IEC 9126

Este modelo fue adoptado para arquitecturas de software, categoriza los atributos de calidad en seis características (Funcionalidad, Fiabilidad, Usabilidad, Eficiencia, Mantenibilidad y Portabilidad) y cada una de estas se dividen a su vez en sub-características.

Descripción de cada uno de estos atributos de calidad.

Funcionalidad: Se refiere a la capacidad del software de proveer un conjunto de funciones que permitan cumplir con unos requerimientos o satisfacer unas necesidades implícitas. Estas funciones se

deben ejecutar bajo ciertas condiciones.

- Adecuación: Capacidad del software para cumplir los requisitos del usuario.
- Corrección: Referida a la capacidad del software para llevar a cabo sus funcionalidades de manera consistente.
- Interoperabilidad: Capacidad del software de interactuar con otros sistemas específicos.
- Seguridad: Capacidad del producto de software de proteger su información y datos así como la de controlar el acceso no autorizado al mismo.
- Conformidad: La capacidad del producto de software de proporcionar un conjunto de funciones para las tareas especificadas.

Fiabilidad: Capacidad de que el software mantenga un nivel especifico de desempeño cuando es usado bajo ciertas condiciones.

- Madurez: Capacidad del producto de software para evitar fallas en los resultados del software.
- Tolerancia a fallos: Capacidad del producto de software para mantener un nivel de desempeño en caso de fallas del software o infracción en una interfaz específica.
- Recuperación: Capacidad del producto del software para restablecer su nivel de desempeño así como recuperación de datos en caso de ocurrir alguna falla.

Usabilidad: La capacidad del software de que sea compresible, de fácil aprendizaje, que su uso sea atractivo al usuario cuando es usado bajo ciertas condiciones.

- Comprensibilidad: Capacidad del producto del software de ser entendible y como pueda ser usado para una tarea en particular.
- Aprendibilidad: Capacidad que tiene el producto de software para permitirle al usuario aprender a utilizarlo.
- Operabilidad: Capacidad del producto de software que permite ser operable y controlable.
- Atractividad: Capacidad del producto del software de ser atractivo al usuario.

Eficiencia: Es la capacidad del software para proporcionar una ejecución apropiada, relativo a la cantidad de recursos usados bajo ciertas condiciones.

- Comportamiento temporal: Capacidad del producto de software para proporcionar un apropiado tiempo de respuesta y de proceso cuando es ejecutada una función bajo ciertas condiciones.
- Usabilidad de recursos: La capacidad que tiene el producto de software de usar apropiadamente cantidad y tipos de recursos cuando ejecuta una función bajo ciertas condiciones.

Mantenibilidad: La capacidad que tiene el software a ser modificado. Estas modificaciones pueden ser correcciones, mejoras o adaptaciones del software a cambios en el ambiente y a requerimientos y especificaciones funcionales.

- Analizabilidad: Capacidad que tiene el producto de software a ser diagnosticado, por causas de fallas o para identificar partes a ser modificadas.
- Cambiabilidad: Capacidad que tiene el producto de software para permitir modificaciones especificas a ser implementadas
- Estabilidad: Capacidad que tiene el producto de software de evitar efectos inesperados por modificaciones sobre el software.
- Facilidad de pruebas: Capacidad que tiene el software de permitir validar las modificaciones realizadas.

Portabilidad: La capacidad que tiene el software a ser transferido a otros ambientes. El ambiente puede ser la organización, hardware o software.

- Adaptabilidad: Capacidad que tiene el producto de software de adaptarse a diferentes ambientes sin la aplicación de acciones u otros medios que permitan llevar cabo esta tarea.
- Instalabilidad: Capacidad que tiene el producto de software a ser instalado en un ambiente especifico.
- Coexistencia: Capacidad que tiene el producto de software de coexistir con otro con otros productos independientes en un ambiente en común.
- Remplazabilidad: Capacidad que tiene el producto de software de ser usado en lugar de otro producto de software para el mismo propósito en el mismo.

Todas estas características de calidad pueden ser consideradas para llevar acabo el proceso de evaluación de la calidad de un producto de software.

3.8. Técnicas de Evaluación de Arquitectura de Software

Según Bosch los atributos de calidad de la arquitectura deben ser evaluados durante el proceso de diseño de la misma y afirma que implementar el sistema y luego establecer valores para los atributos de calidad del mismo seria destinar gran cantidad de recursos y esfuerzo en el desarrollo de un sistema que no satisface los requerimientos de calidad. Basado en su criterio plantea las técnicas de evaluación basada en escenarios, basada en simulación, basada en modelos matemáticos y basada en experiencia.

3.8.1. Evaluación basada en escenarios

De acuerdo con Kazman, un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con éste. Ejemplo: Un desarrollador se enfocará en el uso de la arquitectura para efectos de su construcción o predicción de su desempeño. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta, proveen un vehículo que permite concretar y entender atributos de calidad.

Entre las ventajas de su uso están:

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

Las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes:

1. Utility Tree, propuesto por Kazman: es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software.
2. Profiles, propuestos por Bosch: conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos.

3.8.2. Evaluación basada en simulación

La evaluación basada en simulación según Bosch utiliza una implementación de alto nivel de la arquitectura de software. Consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. El propósito de este método es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Desde el momento justo que estas implementaciones estén disponibles, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

El proceso de evaluación basada en simulación sigue los siguientes pasos (Bosch, 2000):

1. Definición e implementación del contexto.
2. Implementación de los componentes arquitectónicos.
3. Implementación del perfil.
4. Simulación del sistema e inicio del perfil.
5. Predicción de atributos de calidad.

La certeza de los resultados de la evaluación depende de la exactitud del perfil utilizado para evaluar el atributo de calidad y de la precisión con la que el contexto del sistema simula las condiciones del mundo real.
Entre los instrumentos asociados a las técnicas de evaluación basadas en simulación, se encuentran los lenguajes de descripción arquitectónica y los modelos de colas.

3.8.3. Evaluación basada en modelos matemáticos

La evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Según Bosch, permite una evaluación estática de los modelos de diseño arquitectónico, y constituyen una alternativa a la simulación, dado que evalúan el mismo tipo de atributos.
El proceso de evaluación basada en modelos matemáticos sigue los siguientes pasos (Bosch, 2000):

1. Selección y adaptación del modelo matemático.
2. Representación de la arquitectura en términos del modelo.
3. Estimación de los datos de entrada requeridos.
4. Predicción de atributos de calidad.

Instrumentos para las técnicas de evaluación de arquitecturas de software basada en modelos matemáticos, las Cadenas de Markov y los Reliability Block Diagramas.

3.8.4. Evaluación basada en experiencia

En muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evasión de decisiones erradas de diseño, independientemente que estas experiencias se basan en evidencia anecdótica.
Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

En términos de los instrumentos asociados a las técnicas de evaluación basadas en experiencia, se encuentran la Intuición y Experiencia, Tradición y Proyectos similares.

3.9. Métodos de Evaluación de Arquitecturas de Software según Kazman

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema, en la búsqueda

de conflictos que puede presentar una arquitectura, y sus soluciones. Por esta razón, resulta conveniente estudiar los métodos de evaluación de arquitecturas de software.

3.9.1. SAAM (Software Architecture Analysis Method)

El Método de Análisis de Arquitecturas de Software o *Software Architecture Analysis Method* (SAAM) 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. [39]

Es un método de evaluación de arquitectura basado en escenarios, permite evaluar una arquitectura o evaluar y comparar varias. Si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Si se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones. [39]

SAAM contempla 6 pasos:

1. Desarrollo de escenarios.
2. Descripción de la arquitectura.
3. Clasificación y asignación de prioridad de los escenarios.
4. Evaluación individual de los escenarios indirectos.
5. Evaluación de la interacción entre escenarios.
6. Creación de la evaluación global. [41]

3.9.2. ATAM (Architecture Trade-off Analysis Method)

El nombre del método ATAM (Método de Análisis de Acuerdos de Arquitectura) 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. Está inspirado en tres áreas distintas: los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. [39] Apoya a los involucrados en el proyecto a entender las consecuencias de las decisiones arquitectónicas respecto a los atributos de calidad (requisitos no funcionales como: seguridad, disponibilidad, performance, facilidad de mantenimiento, etc.) del sistema. Esta basado en cuestionarios y escenarios, que permiten evaluar qué tan bien un atributo de

calidad es soportado por la arquitectura y (permitiendo reconocer las decisiones de arquitectura que afectan a más de un atributo de calidad). Cuando el atributo de calidad Modificabilidad es el de mayor interés es conveniente su aplicación.

ATAM comprende nueve pasos, agrupados en cuatro fases:

Fase 1: Presentación.

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

Fase 2: Investigación y análisis.

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

Fase 3: Pruebas.

7. Lluvia de ideas y establecimiento de prioridad de escenarios.

Análisis de los enfoques arquitectónicos.

Fase 4: Reporte.

8. Presentación de los resultados. [39]

3.9.3. ARID (Active Reviews for Intermediate Designs)

ARID es un método conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. ARID es un híbrido entre Active Design Review (ADR) y ATAM, descrito anteriormente. ADR es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software.

Es utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. [39] Los roles involucrados en este método de evaluación son: Equipo de verificación, Arquitecto y Stakeholders y se centra en la factibilidad de la arquitectura.

Comprende 9 pasos agrupados en 2 fases:

Fase 1: Actividades Previas

1. Identificación de los encargados de la revisión.

2. Preparar el informe de diseño.
3. Preparar los escenarios base.
4. Preparar los materiales.

Fase 2: Revisión.

5. Presentación del ARID.
6. Presentación del diseño.
7. Lluvia de ideas y establecimiento de prioridad de escenarios.
8. Aplicación de los escenarios.
9. Resumen. [39]

3.9.4. CBAM (Cost-Benefit Analysis Method)

El método de análisis de costos y beneficios introduce las llamadas estrategias arquitectónicas, que consisten en posibles opciones para la resolución de conflictos entre atributos de calidad presentes en una arquitectura. Constituye un marco de referencia que no toma decisiones por los involucrados en el desarrollo del sistema sino que por el contrario, ayuda en la elicitación y documentación de los costos, beneficios e incertidumbre, y provee un proceso de toma de decisiones racional. Entrega bases para el análisis económico de proyectos de desarrollo sistemas basados en una evaluación a la arquitectura.

El método CBAM abarca los siguientes aspectos:

- Selección de escenarios
- Evaluación de los beneficios de los atributos de calidad
- Cuantificación de los beneficios de las estrategias arquitectónicas
- Cuantificación de los costos de las estrategias arquitectónicas implicaciones de calendario
- Cálculo del nivel de deseabilidad
- Toma de decisiones

3.9.5. Método de Diseño y Uso de Arquitecturas de Software propuesto por Bosch

Plantea en su método de diseño de arquitecturas de software, que el proceso de evaluación debe ser visto como una actividad iterativa, que forma parte del proceso de diseño, también iterativo. Una vez que la arquitectura es evaluada, pasa a una fase de transformación, asumiendo que no satisface todos los requerimientos. Luego, la arquitectura transformada es evaluada de nuevo. [39]

El proceso de evaluación propuesto por Bosch se divide en dos etapas:

Etapa I

1. Selección de atributos de calidad.
2. Definición de los perfiles.
3. Selección de una técnica de evaluación.

Etapa II

4. Ejecución de la evaluación.
5. Obtención de resultados. [41]

3.10. Evaluando la arquitectura de software propuesta

El objetivo principal de evaluar la arquitectura del sistema es verificar que el desarrollo del mismo termine en un producto que cumpla con los requerimientos del cliente y la calidad total del software. Para ello se decide utilizar el método ATAM, método basado en escenarios que permite hacer evaluaciones sin tener que haber definido toda la arquitectura, así como también permiten describir la forma en la que el sistema puede crecer, responder a cambios, e integrarse con otros sistemas. Para llevar a cabo este proceso de evaluación del software fueron consideradas varias de las características de calidad propuestas por el modelo ISO/IEC 9126 adaptado para arquitecturas de software, propuesto por Losavio en el año 2003. Se sugiere remitirse a los epígrafes *modelos de calidad* y *evaluación basada en escenarios* para el entendimiento de la aplicación de este método.

Aplicación del método ATAM

Para la aplicación de este método se identificaron las metas del negocio que motivaban el desarrollo del proyecto y los objetivos que tienen que ver con la arquitectura resultando de mayor relevancia las características de calidad que se especifican en la Tabla 4. Para cada escenario se hizo una priorización en base a dos dimensiones:

1) importancia del escenario en relación al éxito del sistema.
2) grado de dificultad para el logro del escenario.

Esta priorización se hizo siguiendo la escala siguiente: alto (A), medio (M) y bajo (B) (ver Tabla 4) y para un mayor entendimiento de esta tabla ver Leyenda.

Escenarios
FI1. Realización de múltiples simulaciones haciendo uso del T-arenal (A, B).
FS1. Se trata de producir intrusión no autorizada al sistema, un usuario (no administrador) trata de ingresar al sistema y este la rechaza (A, B).
PA1. Reemplazo del sistema operativo sin tener que realizar grandes cambios en el código (A, B).
MM1. Adicionar un nuevo modulo al sistema (A, B). **MM**2. Migración del sistema gestor de base de datos, ya sea de forma temporal o definitiva (A, B).

Tabla 4: Árbol de Utilidad.

Leyenda:

FI: Funcionalidad-Interoperabilidad.

FS: Funcionalidad-Seguridad de Acceso.

PA: Portabilidad-Adaptabilidad.

MM: Mantenibilidad-Modificabilidad.

Ahora serán analizaron los escenarios de acuerdo al árbol de utilidad, estableciendo en cada uno de ellos las posibles decisiones arquitectónicas a tomar así como la respuesta lograda por el sistema.

Código del Escenario: FI1	Realización de múltiples simulaciones haciendo uso del T-arenal.
Característica-SubCaracterística	Funcionalidad-Interoperabilidad.
Decisión Arquitectónica	Instalación del MatLab en caso que se necesite su uso. Control del progreso de las simulaciones. Lectura directa a la BD. Creación de un componente en simulación que vaya actualizando en la BD de BioSyS.
Respuesta	Se realizan las simulaciones por el T-arenal.

Tabla 5. Escenario FI1.

Código del Escenario: FS1	Se trata de producir intrusión no autorizada al sistema, un usuario (no administrador) trata de ingresar al sistema y este

	la rechaza.
Característica-SubCaracterística	Funcionalidad- Seguridad de Acceso.
Decisión Arquitectónica	Establecer antes que nada conexión con el sistema gestor de BD. Si el usuario no se autentica contra el gestor entonces es usuario no autorizado y no tiene acceso al sistema.
Respuesta	Deniega el acceso al sistema.

Tabla 6. Escenario FS1.

Código del Escenario: PA1	Reemplazo del sistema operativo, y no tener que realizar cambios en el código.
Característica-SubCaracterística	Portabilidad-Adaptabilidad.
Decisión Arquitectónica	Se escogió como lenguaje de programación Java y como sistema gestor de base de datos PostgreSQL.
Respuesta	El sistema corre sobre cualquier sistema operativo donde corra la máquina virtual de Java.

Tabla 7. Escenario PA1.

Código del Escenario: MM1	Adicionar un nuevo modulo al sistema.
Característica-SubCaracterística	Mantenibilidad-Modificabilidad.
Decisión Arquitectónica	Utilización del estilo basado en componentes, permite que sean adicionados tantos componentes o módulos como se estime conveniente, sería solo adicionarlo a la capa de negocio y conectarlo con la presentación del sistema (BioSyS).
Respuesta	Se adiciona una nueva funcionalidad al sistema.

Tabla 8. Escenario MM1.

Código del Escenario: MM2	Migración del sistema gestor de base de datos
Característica-SubCaracterística	Mantenibilidad-Modificabilidad.
Decisión Arquitectónica	Creación de una capa de acceso a datos. Uso del Framework Hibernate. Utilización del patrón DAO (la implementación de las interface DAO abstraen la capa de negocio de la de acceso a los datos).
Respuesta	Se hacen cambios solamente en la capa de acceso. Cambiar el driver. Cambiar específicamente el tipo de sistema gestor con el que se quiere conectar.

Tabla 9. Escenario MM2.

Las características de calidad propiciadas a través del conjunto de decisiones arquitectónicas que se tomaron fueron, en primer lugar la Portabilidad mediante la Adaptabilidad y en segundo lugar la Mantenibilidad mediante la Modificabilidad. Igualmente se propició la Funcionalidad a través de las decisiones relativas a la Seguridad de Acceso.

3.11. Conclusiones del Capítulo

Se decidió realizar la evaluación arquitectónica del sistema utilizando el método de evaluación ATAM. Se pudo observar que la utilización del mismo es adecuada para la evaluación de este tipo de arquitectura, ya que el método permite una evaluación temprana, no es necesario que toda la arquitectura este definida para ser evaluada. El comportamiento conseguido en la evaluación es válido para soportar futuros cambios que puedan surgir en la arquitectura del sistema. Se pudo comprobar que la característica de calidad más deseada en una arquitectura basada en componentes es la Mantenibilidad, además permitió constatar que las características de calidad que propicia una arquitectura de software basada en componentes, son la Portabilidad y Mantenibilidad.

CONCLUSIONES

Con la realización del presente trabajo se diseñó la arquitectura del Simulador de Sistemas Biológicos (BioSyS), para lograr este resultado se hicieron búsquedas bibliográficas referentes al tema de la arquitectura del software, se hizo un estudio de las tecnologías actuales en materia de arquitectura, de los temas relacionados con estilos y patrones, de los lenguajes de descripción arquitectónica y de los métodos de evaluación de arquitecturas. Posteriormente fue seleccionado el estilo y patrones arquitectónicos a utilizar lo cual permitió la estructuración lógica del diseño, se diseñaron las vistas arquitectónicas del sistema permitiendo lograr una visión general y una mayor comprensibilidad del sistema, se describió la arquitectura del sistema con la generación de un artefacto crítico (Cuaderno de Arquitectura) que orienta a los desarrolladores que van a construir el sistema y que es utilizado para tomar decisiones arquitectónicas importantes. La evaluación del diseño arquitectónico propuesto permitió comprobar que la arquitectura cumple con los requisitos indispensables para el cumplimiento de la calidad del sistema. Con el cumplimiento de cada uno de estos objetivos se ha cumplido el principal objetivo de este trabajo, proponer una arquitectura evaluada para BioSyS.

Se puede decir que la arquitectura diseñada es reusable ya que permite la creación de nuevos simuladores modulares con características semejantes, en un tiempo de desarrollado relativamente corto sin tener que hacer cambios significativos a la arquitectura y además es flexible permitiendo la agregación o eliminación de nuevas funcionalidades al sistema.

RECOMENDACIONES

Luego de concluido este trabajo se recomienda:

- El refinamiento constante de la arquitectura propuesta a un nivel adecuado de detalle durante todo el ciclo de desarrollo del sistema.
- Poner en práctica el diseño arquitectónico propuesto.
- Ir incorporando los patrones de diseño utilizados en la construcción del sistema, en el documento de la arquitectura.

REFERENCIAS BIBLIOGRÁFICAS

[1] The System Biology Markup Language. [Online] http://www.sbml.org.

[2] **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* s.l. : Version1.0, Marzo 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx .

[3] **Daccach T, José Camilo.** Articulo DELTA # 15.[Online] 11 22, 2007. http://www.deltaasesores.com/prof/PRO015.html..

[4] **CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL.** Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

[5] **Reynoso, Carlos and Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires: Version 1.0, Marzo 2004. http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

[6] **CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL.** *Arquitecturas de Software, Guías de Estudio.* Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

[7] **Yorio, Darío.** Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles. [Online]http://postgrado.info.unlp.edu.ar/Carrera/Magister/Ingenieria%20de%20Software/Tesis/Yorio.pdf.

[8] **CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL.** Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf .

[9] **Lagos Torres, Manuel.** Introducción al diseño con patrones. [Online]http://www.elrincondelprogramador.com/default.asp?id=29&pag=articulos/leer.asp.

[10] Patrones y Patrones de Diseño (i). [Online] http://www.info-ab.uclm.es/asignaturas/42579/pdf/04-Capitulo4a.pdf .

[11] **Visconti, Marcello and Astudillo, Hernán.** Fundamentos de Ingeniería de Software, Patrones de Diseño. [Online]http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

[12] **CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL.** Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf .

[13] **Reynoso, Carlos and Kiccillof, Nicolás.** Lenguajes de Descripción de Arquitectura. [Online] Marzo 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx#top.

[14] **Hernández Orallo, Enrique.** El Lenguaje Unificado de Modelado (UML).

[Online]http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

[15] **Mendoza Sánchez, María A.** Metodologías De Desarrollo De Software. [Online]http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html

[16] Sitio de la metodología OpenUp. [Online] http://10.34.20.5:5800/OpenUP/.

[17] Embarcadero Lanza ER/Studio 7.5. [Online] http://www.software.net.mx/desarrolladores/minegocio/noticias/comercial/embarcaderoxml.htm .

[18] Unified Modeling Language. [Online]http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html.

[19] Visual Paradigm for UML. [Online]http://www.versioncero.com/noticia/210/visual-paradigm-for-uml.

[20] Lenguaje Java. [Online] http://www.slideshare.net/Angelus8/lenguaje-de-programacion-java.

[21] El Lenguaje C++. [Online] http://www.zator.com/Cpp/E1_2.htm .

[22] Manual Tutorial de PHP. [Online] http://www.superhosting.cl/manuales/manual-tutorial-de-php.html

[23] Desarrollo de aplicaciones multiplataforma con NetBeans IDE. [Online]http://www.sun.com/emrkt/innercircle/newsletter/latam/0207latam_feature.html.

[24] **Gutierrez, Juan.** El entorno de desarrollo Eclipse. [Online]http://www.uv.es/~jgutierr/MySQL_Java/TutorialEclipse.pdf.

[25] Manual.es/Part1/Compilando la Fuente. [Online]http://wiki.blender.org/index.php/Manual.es/PartI/Compiling_the_sources.

[26] **García, Luis.** Sistema de control de versiones: SUBVERSION. [Online] enero 2008. http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=548.

[27] Sistema Gestor de Base de Datos PostgreSQL. [Online] http://www.http-peru.com/postgresql.php.

[28] MySQL. [Online] www.netpecos.org/docs/mysql_postgres/x57.html.

[29] Qué es Oracle. [Online] http://www.desarrolloweb.com/articulos/840.php .

[30] Qué es un framework? [Online] septiembre 2006. http://jordisan.net/blog/2006/que-es-un-framework/ .

[31] Frameworks para Java. [Online] http://micomandero.wordpress.com/2007/05/25/frameworks-para-java/.

[32] Sitio de la metodología OpeUp. [Online] http://10.34.20.5:5800/OpenUP/.

[33] Sitio de la Asignatura Ingeniería de Software 1. *Conferencia 3_ FT Requerimientos.* [Online] http://teleformacion.uci.cu/mod/resource/view.php?id=8865.

[34] Sitio de la Asignatura Ingeniería de Software 2. *Material_de_Apoyo_Conferencia_Diseño.* [Online]http://teleformacion.uci.cu/mod/resource/view.php?id=21363.

[35] **Pressman, Roges S.** *Ingeniería de Software un Enfoque Práctico.* s.l. : 5ta Edicion, 2005. http://biblioteca.uci.cu/bives/titdigitales.htm#igs .

[36] Sitio de la Asignatura Ingeniería de Software 2. *Conferencia4. FT Implementación,.* [Online] http://teleformacion.uci.cu/mod/resource/view.php?id=22199 .

[37] La importancia de la arquitectura en el desarrollo de software con calidad. [Online] febrero 17, 2005. http://www.eafit.edu.co/NR/rdonlyres/223A8F47-27B5-4EB8-B695-4097F745D701/0/Arquitectura.pdf.

[38] **Icedo Ojeda, Rosa Virginia and Trebejo Vargas, Jorge Moisés.** [Online] Mayo 2003. http://www.cimat.mx/~ricedo/IngSW/SoftwareArchitectureAssesment.pdf.

[39] **Peralta, Arturo and Palomino, Martin.** Calidad en Sistemas Basados en Componentes. *Universidad de Castilla- La Mancha.* [Online] 01 2, 2008. http://alarcos.inf-cr.uclm.es/doc/cmsi/trabajos/Arturo%20Peralta%20-%20CSBC%20-%20Doc.pdf .

BIBLIOGRAFÍA

The System Biology Markup Language [Online]http://www.sbml.org.

Reynoso, Carlos Billy. *Introducción a la Arquitectura de Software.* s.l. : Version1.0, Marzo 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.mspx .

Daccach T, José Camilo. Articulo DELTA # 15 [Online] 11 22, 2007. http://www.deltaasesores.com/prof/PRO015.html..

CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL. Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

Reynoso, Carlos and Kiccillof, Nicolás. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* Buenos Aires: Version 1.0, Marzo 2004. http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL. *Arquitecturas de Software, Guías de Estudio.* Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

Yorio, Darío. Identificación y Clasificación de Patrones en el Diseño de Aplicaciones Móviles. [Online]http://postgrado.info.unlp.edu.ar/Carrera/Magister/Ingenieria%20de%20Software/Tesis/Yorio.pdf.

CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL. Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf .

Lagos Torres, Manuel. Introducción al diseño con patrones. [Online]http://www.elrincondelprogramador.com/default.asp?id=29&pag=articulos/leer.asp.

[10] Patrones y Patrones de Diseño (i). [Online] http://www.info-ab.uclm.es/asignaturas/42579/pdf/04-Capitulo4a.pdf .

Visconti, Marcello and Astudillo, Hernán. Fundamentos de Ingeniería de Software, Patrones de Diseño. [Online]http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

CAMACHO, ERIKA, CARDESO, FABIO and NUÑEZ, GABRIEL. Arquitecturas de Software, Guías de Estudio. [Online] Abril 2004. http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf .

Reynoso, Carlos and Kiccillof, Nicolás. Lenguajes de Descripción de Arquitectura. [Online] Marzo 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/lenguaje.mspx#top.

Hernández Orallo, Enrique. El Lenguaje Unificado de Modelado (UML). [Online]

http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.

Mendoza Sánchez, María A. Metodologías De Desarrollo De Software. [Online]http://www.informatizate.net/articulos/metodologias_de_desarrollo_de_software_07062004.html

Sitio de la metodología OpenUp. [Online] http://10.34.20.5:5800/OpenUP/.

Embarcadero Lanza ER/Studio 7.5. [Online] http://www.software.net.mx/desarrolladores/minegocio/noticias/comercial/embarcaderoxml.htm .

Unified Modeling Language [Online]http://gidis.ing.unlpam.edu.ar/personas/glafuente/uml/uml.html.

Visual Paradigm for UML [Online]http://www.versioncero.com/noticia/210/visual-paradigm-for-uml.

Lenguaje Java. [Online] http://www.slideshare.net/Angelus8/lenguaje-de-programacion-java.

El Lenguaje C++. [Online] http://www.zator.com/Cpp/E1_2.htm .

Manual Tutorial de PHP. [Online] http://www.superhosting.cl/manuales/manual-tutorial-de-php.html

Desarrollo de aplicaciones multiplataforma con NetBeans IDE. [Online]http://www.sun.com/emrkt/innercircle/newsletter/latam/0207latam_feature.html.

Gutierrez, Juan. El entorno de desarrollo Eclipse. [Online]http://www.uv.es/~jgutierr/MySQL_Java/TutorialEclipse.pdf.

Manual.es/Part1/Compilando la Fuente. [Online]http://wiki.blender.org/index.php/Manual.es/PartI/Compiling_the_sources.

García, Luis. Sistema de control de versiones: SUBVERSION. [Online] enero 2008. http://observatorio.cnice.mec.es/modules.php?op=modload&name=News&file=article&sid=548.

Sistema Gestor de Base de Datos PostgreSQL. [Online]http://www.http-peru.com/postgresql.php.

MySQL [Online] www.netpecos.org/docs/mysql_postgres/x57.html.

Qué es Oracle. [Online] http://www.desarrolloweb.com/articulos/840.php .

Qué es un framework? [Online] septiembre 2006. http://jordisan.net/blog/2006/que-es-un-framework/ .

Frameworks para Java. [Online] http://micomandero.wordpress.com/2007/05/25/frameworks-para-java/.

Sitio de la metodología OpeUp. [Online] http://10.34.20.5:5800/OpenUP/.

Sitio de la Asignatura Ingeniería de Software 1. *Conferencia 3_ FT Requerimientos.* [Online] http://teleformacion.uci.cu/mod/resource/view.php?id=8865.

Sitio de la Asignatura Ingeniería de Software 2. *Material_de_Apoyo_Conferencia_Diseño.* [Online]http://teleformacion.uci.cu/mod/resource/view.php?id=21363.

Pressman, Roges S. *Ingeniería de Software un Enfoque Práctico.* s.l. : 5ta Edicion, 2005. http://biblioteca.uci.cu/bives/titdigitales.htm#igs .

Sitio de la Asignatura Ingeniería de Software 2. *Conferencia4. FT Implementación,.* [Online] http://teleformacion.uci.cu/mod/resource/view.php?id=22199 .

La importancia de la arquitectura en el desarrollo de software con calidad. [Online] febrero 17, 2005.

http://www.eafit.edu.co/NR/rdonlyres/223A8F47-27B5-4EB8-B695-4097F745D701/0/Arquitectura.pdf.

Icedo Ojeda, Rosa Virginia and Trebejo Vargas, Jorge Moisés. [Online] Mayo 2003. http://www.cimat.mx/~ricedo/IngSW/SoftwareArchitectureAssesment.pdf.

Peralta, Arturo and Palomino, Martin. Calidad en Sistemas Basados en Componentes. *Universidad de Castilla- La Mancha.* [Online] 01 2, 2008. http://alarcos.inf-cr.uclm.es/doc/cmsi/trabajos/Arturo%20Peralta%20-%20CSBC%20-%20Doc.pdf .

Patrones GoF, http://www.info-ab.uclm.es/asignaturas/42579/pdf/04-Capitulo4a.pdf

Noel Moreno Lemus, Tesis en opción al grado de Máster en Bioinformática, BioSyS: Software para la simulación y análisis de sistemas biológicos, junio 2007.

Gilberto Arias Naranjo, Tesis en opción al grado de Máster en Bioinformática, Editor de ecuaciones para la Plataforma de Simulación de Sistemas Biológicos, 2008.

GLOSARIO DE TÉRMINOS

A continuación se detallan todos aquellos términos que se consideran de interés a la hora de comprender el trabajo.

Algorithm: clase ubicada en el servidor de cálculo distribuido encargada de ejecutar las unidades de trabajo recibidas y las envía al servidor de base de datos.

API JDBC: define una forma estándar para que el código Java se comunique con bases de datos relacionales.

DataManager: clase ubicada en el servidor de cálculo distribuido que recibe el trabajo solicitado por el cliente, genera unidades de trabajo se las envía a sus maquinas clientes junto a la clase Algorithm.

DAO (Data Access Object): patrón de diseño de acceso a datos, es una solución al problema del diferencial de impedancia entre un programa de aplicación orientado a objetos y una base de datos relacional, empleando únicamente la interfaz de programación (API) nativa del manejador de base de datos, o algún otro sustituto como el ODBC, JDBC, DBI. El propósito del patrón DAO es, en pocas palabras, abstraer y encapsular todos los accesos a la fuente de datos.

GUI: constructor de interfaz gráfica.

Herramienta ORM (Mapeo Objeto-Relacional): ayuda a reducir la llamada diferencia de impedancia Objeto-Relacional.

T-arenal: Sistema de Cómputo Distribuido programado en Java, se basa en el paradigma de la POO y utiliza RMI para el envío de mensajes.

MatLab: es una potente herramienta de cálculo. Puede ser utilizada en computación matemática, modelado y simulación, análisis y procesado de datos, visualización y representación de gráficos y desarrollo de algoritmos.

Modificabilidad: Capacidad que tiene un producto software que permite que una determinada modificación sea implementada.

Plug - in: componentes conectados.

Sistemas Biológicos: conjunto de órganos y estructuras análogas que trabajan en conjunto para cumplir alguna función en el ser vivo.

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

Weka: herramienta de aprendizaje automático y minería de datos (data mining), escrita en lenguaje Java, se adapta a cualquier entorno, posee un diseño arquitectónico basado en componentes independientes y su código fuente esta disponible.