

Universidad de las Ciencias Informáticas

Facultad 6



**Integración de un Sistema de Almacenamiento Distribuido al Portal de Servicios
Bioinformáticos**

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

Autores: Jorge Humberto Armas Hernández
Alberto Hernández Herrera

Tutores: MSc. Longendri Aguilera Mendoza
Ing. Mónica Teresa Llorente Quesada

La Habana, junio de 2011

“Año 53 de la Revolución”

DECLARACIÓN DE AUTORÍA

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

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

Jorge Humberto Armas Hernández

Autor

MSc. Longendri Aguilera Mendoza

Tutor

Alberto Hernández Herrera

Autor

Ing. Mónica Teresa Llorente Quesada

Tutora

DATOS DE CONTACTO

Longendri Aguilera Mendoza

Profesor Asistente.

Licenciado en Ciencia de la Computación por la Universidad de la Habana.

Máster en Bioinformática por el Instituto Superior de Tecnología y Ciencias Aplicadas (InSTEC)

Miembro del Grupo de Bioinformática de la UCI.

Mónica Teresa Llorente Quesada

Instructor

Ingeniera en Ciencias Informáticas

Dpto. Técnicas de Programación. Facultad 6

DEDICATORIA

Jorge Humberto Armas Hernández

Dedico este trabajo de manera muy especial a mis abuelos Jorge y Humberto que dejaron de estar entre nosotros hace varios años.

A mi Mom que es la mejor madre del mundo y a mi papá que siempre ha sido mi guía y ejemplo a seguir; a los dos gracias por todo su esfuerzo y amor que sin ellos no hubiese sido posible llegar hasta aquí.

A mi hermanita que ya ha dejado de ser una niña para convertirse en una mujer, pero siempre será mi nené linda, espero esto le sirva de motivación para este mismo camino que ahora le toca a ella recorrer.

A mis abuelas que ojalá las pueda disfrutar muchos años más porque son muy especiales para mi.

A mi bebe que el destino quiso que nos volviéramos a encontrar para pasar muchos momentos felices a su lado, por apoyarme todos estos años y darme todo su amor.

A todos mis tíos, en especial a mis tío César y tía Idalmis que me han acogido como un hijo y han tenido que cargar conmigo unos cuantos años.

A todos mis primos y en general a toda mi familia que es bastante grande y especial.

Alberto Hernández Herrera

AGRADECIMIENTOS

Jorge Humberto Armas Hernández

A Mónica y Longendri por apoyarnos y darnos confianza para lograr llegar hasta el final.

A Alberto con el que compartí todo este trabajo y sin él esto no hubiese sido posible.

A Randy y Yendri que desde el primer año formamos el trío de los Pepillos y son uno de los mejores recuerdos que me llevo de la universidad, ojalá podamos seguir viéndonos frecuentemente.

A todos mi amigos y amigas que son unos cuantos y no quiero mencionar nombres para que no se me quede ninguno, los llevaré siempre conmigo.

A todos los que de una manera u otra han colaborado para cumplir con el objetivo de este trabajo.

Alberto Hernández Herrera

RESUMEN

El presente trabajo de diploma tiene como objetivo integrar al Portal de Servicios Bioinformáticos de la Universidad de las Ciencias Informáticas, un Sistema de Archivos Distribuidos (SAD), que utilice el espacio disponible de las estaciones de trabajo. Se desarrolló una interfaz Web que permite la gestión de los datos almacenados en el SAD y una fachada que posibilita la comunicación entre estos dos componentes. Durante el desarrollo de este sistema se utilizó la metodología de software OpenUP, el Lenguaje de Modelado Unificado (UML) y Visual Paradigm for UML para el modelado del sistema. Se desplegó el Sistema de Archivos Distribuidos haciendo uso de la herramienta Hadoop DFS. Como Entorno de Desarrollo Integrado se utilizó NetBeans IDE 6.9 y Java como lenguaje de programación. Junto a estos elementos se incorporan el framework Spring y Portlet como componente Web gestionado por el contenedor de portlets Liferay, que a su vez es un Sistema de Gestión de Contenidos. Todos estos elementos giran alrededor del patrón de diseño Modelo Vista Controlador.

PALABRAS CLAVE

Bioinformática

Portal de Servicios Bioinformáticos

Almacenamiento Distribuido

Sistema de Archivos Distribuidos

Hadoop

Web

Fachada

Portlet

TABLA DE CONTENIDOS

DECLARACIÓN DE AUTORÍA I

DATOS DE CONTACTO II

DEDICATORIA III

AGRADECIMIENTOS IV

RESUMEN V

TABLA DE CONTENIDOS VI

TABLA DE FIGURAS IX

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 4

1. Definiciones previas 4

1.1. Sistemas Distribuidos 4

1.2. Sistemas de Archivos Distribuidos 5

1.3. Arquitectura de los Sistemas Distribuidos 7

1.3.1. Arquitectura Cliente-Servidor 7

1.3.2. Modelo Múltiples Servidores 8

1.3.3. Modelo Servidores Proxy y Cachés 9

1.3.4. Arquitectura Procesos de Igual a Igual 10

2. Estudio valorativo 11

2.1. Sistemas de Archivos Distribuidos existentes 11

2.1.1. Sistema de Almacenamiento Masivo para Archivos Pequeños (MSFSS) 11

2.1.2. Sistema de Archivos de Google 12

2.1.3. GridExpand 13

2.1.4. Sistema de Archivos Distribuidos Hadoop 14

2.2. Metodología de desarrollo 15

2.2.1. OpenUP 16

2.3. Herramientas y entorno de desarrollo 16

2.3.1. Redmine 16

2.3.2. Visual Paradigm for UML 6.4 17

2.3.3. NetBeans IDE 6.9 17

2.3.4. Karmasphere Studio 18

2.3.5. Portlet 18

2.3.6. Liferay 19

2.3.7. Spring 19

2.3.8. ExtJS 4.0 20

2.4. Lenguajes 20

2.4.1.	Lenguaje Unificado de Modelado (UML)	21
2.4.2.	Java	21
2.5.	Conclusiones del capítulo.....	21
CAPÍTULO 2:	CARACTERÍSTICAS DEL SISTEMA.....	24
1.	Modelo de Dominio.	24
1.1.	Conceptos del dominio.....	24
1.2.	Diagrama de clases del Modelo de Dominio.....	24
2.	Especificación de requisitos.	25
2.1.	Requisitos funcionales.....	25
2.2.	Requisitos no funcionales.....	26
3.	Modelado del sistema.	27
3.1.	Actores del sistema.....	27
3.2.	Diagrama de Casos de Uso del sistema.....	28
3.3.	Especificación de los Casos de Uso.....	29
3.3.1.	Descripción del Caso de Uso: Administrar Estructura Molecular	29
3.3.2.	Descripción del Caso de Uso: Gestionar Metadatos de Estructura Molecular	33
3.3.3.	Descripción del Caso de Uso: Establecer Valores a Metadatos de Estructura Molecular	35
3.4.	Conclusiones del capítulo.....	35
CAPÍTULO 3:	ANÁLISIS Y DISEÑO DEL SISTEMA	37
1.	Modelo de análisis.	37
1.1.	Diagramas de clases del análisis.	37
2.	Arquitectura del sistema.....	38
3.	Modelo de diseño.....	40
3.1.	Diagramas de clases del diseño.....	41
3.2.	Diagramas de secuencia.	45
3.2.1.	Caso de Uso Administrar Estructura Molecular.	45
3.2.2.	Caso de Uso Gestionar Metadatos de Estructura Molecular.	46
4.	Descripción de las clases principales del sistema.	48
4.1.	Conclusiones del capítulo.....	49
CAPÍTULO 4:	IMPLEMENTACIÓN Y PRUEBA.....	50
1.	Modelo de implementación.....	50
1.1.	Diagrama de componentes.	50
1.2.	Diagrama de despliegue.....	52
2.	Integración del Sistema de Archivos Distribuidos con la interfaz Web.	53
3.	Validación de la integración.	54

3.1. Casos de Prueba.....	56
3.1.1. Caso de Uso Administrar Estructura Molecular.	56
3.1.2. Caso de Uso Gestionar Metadatos de Estructura Molecular.	56
4. Conclusiones del capítulo.	57
CONCLUSIONES.....	58
RECOMENDACIONES.....	59
REFERENCIAS BIBLIOGRÁFICAS.....	60
BIBLIOGRAFÍA.....	61
GLOSARIO DE TÉRMINOS.....	63
ANEXOS.....	65

TABLA DE FIGURAS

<i>Figura 1: Modelo Carga/Descarga.</i>	6
<i>Figura 2: Modelo Acceso Remoto.</i>	7
<i>Figura 3: Arquitectura Cliente-Servidor.</i>	8
<i>Figura 4: modelo Múltiples Servidores.</i>	9
<i>Figura 5: modelo Servidores Proxy y Cachés.</i>	10
<i>Figura 6: arquitectura Procesos de Igual a Igual</i>	11
<i>Figura 7: Diagrama de clases del Modelo de Dominio.</i>	25
<i>Figura 8: Diagrama de Casos de Uso del Sistema.</i>	29
<i>Figura 9: DCA CU Administrar Estructura Molecular.</i>	37
<i>Figura 10: DCA CU Gestionar Metadatos de Estructura Molecular.</i>	38
<i>Figura 11: Vista de la Arquitectura.</i>	39
<i>Figura 12: DCD Subsistema FacadeSAD.</i>	42
<i>Figura 13: DCD CU Administrar Estructura Molecular.</i>	43
<i>Figura 14: DCD CU Gestionar Metadatos de Estructura Molecular.</i>	44
<i>Figura 15: DCD CU Establecer Valores a Metadatos de Estructura Molecular.</i>	44
<i>Figura 16: DS CU Administrar Estructura Molecular. Escenario: Guardar Estructura Molecular.</i>	45
<i>Figura 17: DS CU Administrar Estructura Molecular. Escenario: Eliminar Estructura Molecular.</i>	46
<i>Figura 18: DS CU Gestionar Metadatos de Estructura Molecular. Escenario: Mostrar</i>	47
<i>Figura 19: DS CU Gestionar Metadatos de Estructura Molecular. Escenario: Modificar</i>	47
<i>Figura 20: Diagrama de Componentes: Componente FacadeSAD.</i>	51
<i>Figura 21: Diagrama de Componentes: Componente Portlet Interfaz Web</i>	52
<i>Figura 22: Diagrama de Despliegue del Sistema.</i>	53
<i>Figura 23: Código de integración entre Interfaz Web y FacadeSAD.</i>	54
<i>Figura 24: Código de integración entre Interfaz Web y FacadeSAD.</i>	54
<i>Figura 25: Interfaz Web - Escenario Subir Estructura Molecular.</i>	65
<i>Figura 26: Interfaz Web - Escenario Mostrar Metadatos de Estructuras Moleculares</i>	66

INTRODUCCIÓN

Hoy día el mundo de la informática enfrenta una eminente preocupación con el crecimiento acelerado del universo digital. Estudios realizados revelan que la información digital generada acerca de las personas sobrepasa la cantidad de información creada por ellas mismas, la Sombra Digital como fue nombrado este fenómeno se incrementa cada día aumentando consigo el universo digital.

En el 2007 la información que se creó, capturó o replicó, superó la capacidad de almacenamiento disponible. Esta expansión acelerada está dada fundamentalmente por el crecimiento de las ventas de las cámaras digitales, la televisión digital, la computación en la nube, las redes sociales, investigaciones científicas, entre otros.

Una de las ramas de la ciencia que no está ajena a este fenómeno es la Bioinformática, que surgió debido a la presencia de grandes volúmenes de datos biológicos; la misma se encarga del estudio, comprensión y análisis de estos datos utilizando herramientas informáticas. Su protagonismo está dado por dos razones principales: la enorme cantidad de datos de origen biológico sólo puede ser analizada utilizando computadoras, la mayoría de los datos no son de fácil comprensión y se procesan utilizando sofisticados algoritmos computacionales.

Uno de los retos más importantes que afronta hoy la Bioinformática es la enorme cantidad de datos biológicos que se encuentran disponibles en repositorios biológicos para ser procesados. Por ejemplo: se conoce la secuencia de miles de millones de nucleótidos, de millones de proteínas, la estructura tridimensional de más de 40 mil complejos moleculares (proteínas y ácidos nucleico) y el genoma completo de los humanos, ratones, mosquitos, entre otros. Además, todos estos grandes repositorios se van enriqueciendo con nuevas entidades en el transcurso de los años.

En la Universidad de las Ciencias Informáticas (UCI) está presente la Bioinformática como una rama de investigación y producción de software. Razón por la cual, a pesar de las exigencias de recursos computacionales que tienen las aplicaciones en esta área, se ha decidido poner a disposición de algunos centros científicos del país, un Portal de Servicios Bioinformáticos que integre varios programas existentes de gran importancia en el procesamiento de datos biológicos, así como otros que se desarrollan en la propia universidad.

La UCI es el centro a nivel nacional con el mayor número de computadoras personales disponibles en una red local, cuenta con más de 6000 estaciones de trabajo distribuidas por toda la universidad. En aras de utilizar todo este potencial de cómputo, se han realizados trabajos anteriores que ponen a disposición los recursos computacionales presentes en la institución para satisfacer las demandas de cálculo que exigen varios proyectos Bioinformáticos.

En materia de almacenamiento, también se han dado pasos de avance. En investigaciones previas se ha llegado a especificar un Sistema de Almacenamiento Distribuido (SAD) teniendo en cuenta las características de la red y las estaciones de trabajo presentes en el entorno UCI. El sistema fue modelado mediante las Redes de Petri Coloridas y luego validado a través de simulaciones para evaluar su desempeño.

El Portal tendrá la necesidad de contar con mayor capacidad de almacenamiento, pues un usuario común tiene la posibilidad de subir al Portal los archivos que serán procesados y se corre el riesgo de que no exista suficiente espacio disponible en el servidor para su almacenamiento.

Teniendo en cuenta la situación anterior se plantea como **problema científico a resolver**: ¿Cómo aumentar la capacidad de almacenamiento que brinda el Portal de Servicios Bioinformáticos de la UCI para lograr una mejor prestación de los diferentes servicios ofertados a la comunidad científica?

Para dar solución al problema antes mencionado el presente trabajo se plantea como **objetivo general**: Integrar al Portal de Servicios Bioinformáticos de la UCI un Sistema de Archivos Distribuidos que utilice el espacio disponible de las estaciones de trabajo, para almacenar los datos biológicos.

Desglosándose en los siguientes **objetivos específicos**:

- Desplegar un Sistema de Archivos Distribuidos que utilice el espacio disponible de las estaciones de trabajo, para almacenar datos biológicos.
- Desarrollar una interfaz Web para el Portal de Servicios Bioinformáticos que permita gestionar los datos almacenados en el Sistema de Archivos Distribuidos.
- Integrar el Sistema de Archivos Distribuidos desplegado con la interfaz Web.
- Validar la integración de la interfaz Web con el Sistema de Archivos Distribuidos al Portal de Servicios Bioinformáticos.

En relación con el problema y los objetivos se define como **objeto de estudio de la investigación** los

Sistemas de Archivos Distribuidos y el **campo de acción** la gestión de los datos almacenados en un Sistema de Archivos Distribuidos a través de una interfaz Web.

Para realizar con éxito este trabajo se proponen las siguientes **tareas de la investigación**:

1. Estudio y selección de las herramientas a utilizar para el desarrollo del SAD y la interfaz Web.
2. Especificación de los requerimientos funcionales y no funcionales de la interfaz Web.
3. Diseño de la interfaz Web.
4. Implementación de la interfaz Web.
5. Despliegue del Sistema de Archivos Distribuidos.
6. Integración del Sistema de Archivos Distribuidos y la interfaz Web.
7. Validación de los resultados de la integración.

Con el objetivo de organizar y estructurar el trabajo se han definido los siguientes capítulos:

Fundamentación teórica

Conceptos y definiciones importantes sobre los Sistemas de Archivos Distribuidos y sus distintos tipos de arquitectura, así como un estudio valorativo para determinar cuál integrar al Portal de Servicios Bioinformáticos; además de las principales características sobre las metodologías, tecnologías y herramientas de desarrollo utilizadas.

Características del sistema

En este capítulo el lector podrá conocer las principales características del sistema, Modelo de Dominio, requisitos funcionales y no funcionales, además del modelado del sistema.

Análisis y diseño del sistema

Vistas y descripciones de los principales diagramas y arquitectura del sistema.

Implementación y prueba

Vistas y descripciones del Modelo de Implementación y la integración entre el Sistema de Archivos Distribuidos con la interfaz Web. Detalles de las pruebas realizadas y sus resultados.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En la presente sección se abordarán conceptos y definiciones importantes que ayudarán al lector a comprender mejor la investigación, se plasmará un estudio valorativo sobre algunos Sistemas de Archivos Distribuidos existentes que pudieran utilizarse para integrarlo al Portal de Servicios Bioinformáticos, además de las principales características de la metodología, tecnologías y herramientas de desarrollo utilizadas.

1. Definiciones previas.

Debido a que el objetivo general de este trabajo es integrar al Portal de Servicios Bioinformáticos un Sistema de Archivos Distribuidos que permita utilizar el espacio disponible de las estaciones de trabajo para almacenar los datos biológicos. Se pretende esclarecer algunos puntos de vital importancia para llevar a cabo esta integración, tales como: Sistemas Distribuidos, Sistemas de Archivos Distribuidos, sus arquitecturas y características más relevantes.

1.1. Sistemas Distribuidos.

Un Sistema Distribuido es una colección de componentes independientes conectados a través de una red, mediante la cual se comunican y se coordinan para aparecer ante el usuario del sistema como un único componente (1).

Los Sistemas Distribuidos se desarrollaron con el objetivo de compartir recursos, mostrando entre sus principales características:

Confiabilidad: los Sistemas Distribuidos son más confiables que los centralizados, porque si dejara de funcionar uno de sus componentes, el resto del sistema seguiría funcionando sin afectaciones.

- **Velocidad:** un Sistema Distribuido puede alcanzar mayor poder de cómputo que uno centralizado.
- **Distribución inherente:** las actividades pueden distribuirse a través de varios componentes.
- **Transparencia:** una determinada tarea o actividad puede procesarse en cualquier componente sin que el usuario tenga conocimiento. Existen varios tipos de transparencia:

- **Transparencia en localización:** el usuario no sabe dónde está ubicado realmente el recurso que está utilizando.
- **Transparencia de paralelismo:** varias actividades pueden ocurrir paralelamente sin el conocimiento del usuario.

1.2. Sistemas de Archivos Distribuidos.

Los Sistemas de Archivos Distribuidos están formados por componentes, encargados de almacenar y procesar los datos y atender a las peticiones de los usuarios. Brindan la posibilidad de obtener acceso mediante la red a archivos almacenados en nodos remotos, con un desempeño y fiabilidad casi del mismo modo que si fueran locales. Además de las características de los Sistemas Distribuidos, un Sistema de Archivos Distribuidos debe presentar otras características deseables como:

- **Disponibilidad:** el sistema debe ser capaz de responder a las peticiones de los usuarios en todo momento.
- **Tolerancia a fallos:** algunos componentes del Sistema de Archivos Distribuidos podrían fallar y el sistema debe ser capaz de gestionar estas fallas para mantener la disponibilidad de los datos.
- **Réplicas:** es posiblemente la principal estrategia en cuanto la tolerancia a fallos; los datos se replican por los componentes del Sistema de Archivos Distribuidos encargados de almacenar los datos y en caso de fallar alguno de estos componentes, los datos que éste almacenaba se encontrarán disponibles en otros componentes.
- **Escalabilidad:** el sistema debe ser flexible para poder aumentar su capacidad de almacenamiento y cómputo, agregando nuevos componentes al sistema sin que afecte su rendimiento.

Por lo general un Sistema de Archivos Distribuidos cuenta con dos tipos de componentes, los servidores de archivos y los servidores de direcciones. Los primeros se encargan de las operaciones en los archivos individuales, como la lectura y escritura, mientras que los segundos se encargan de crear y administrar directorios, añadir y eliminar archivos de los directorios.

Un aspecto importante del modelo de archivos está relacionado con los permisos de modificación o no modificación de los archivos, después de su creación. Existen algunos sistemas que permiten únicamente

crear y leer. Una vez creado un archivo no se puede modificar. Estos facilitan el ocultamiento y duplicación de los archivos, eliminan todos los problemas asociados con la actualización de todas las copias de un archivo cada vez que éste se modifique.

Cuando se accede a un archivo remoto el uso sobre éste se realiza utilizando uno de los dos modelos de administración, modelo Carga/Descarga o el modelo de Acceso Remoto. En el modelo Carga/Descarga el servicio de archivo solo proporciona la lectura del mismo. Se transfiere todo un archivo de uno de los servidores de archivos al cliente solicitante. La escritura transfiere todo un archivo en sentido contrario del cliente al servidor. Los archivos se pueden almacenar en memoria o en disco local según sea necesario.

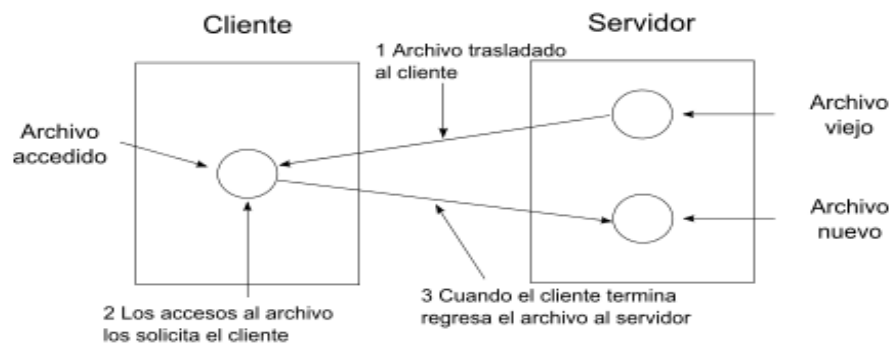


Figura 1: Modelo Carga/Descarga.

El servicio de archivos de Acceso Remoto proporciona un gran número de operaciones para abrir y cerrar archivos, leer y escribir parte del archivo, moverse a través de éste, examinar y modificar los atributos de archivos, entre otras.

Mientras en el modelo Carga/Descarga el servicio de archivos solo proporciona el almacenamiento físico y la transferencia, en este caso el sistema de archivos se ejecuta en los servidores y no en los clientes. Su ventaja es que no necesita cuantioso espacio por parte de los clientes, a la vez que elimina la necesidad de transferir archivos completos, cuando solo se necesita una parte de ellos.

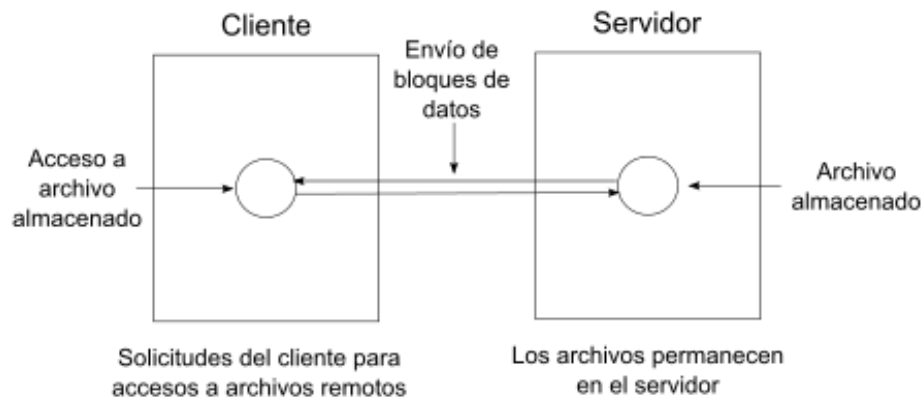


Figura 2: Modelo Acceso Remoto.

1.3. Arquitectura de los Sistemas Distribuidos.

La ubicación de componentes y la asignación de responsabilidades son muy importantes en el diseño de Sistemas Distribuidos, de ellos depende la seguridad y el desempeño de los recursos. Cuando se va a utilizar una arquitectura se debe tener en cuenta cómo se comporta en cuanto a las características deseables de los Sistemas de Archivos Distribuidos. Los principales tipos de arquitecturas en un Sistema de Archivos Distribuidos en cuanto a la ubicación y asignación de responsabilidades de sus componentes son: Cliente-Servidor, de la cual se derivan modelos como: Múltiples Servidores y Servidores Proxy y Cachés, el otro tipo de arquitectura es la de Procesos de Igual a Igual. Estas arquitecturas son descritas a continuación.

1.3.1. Arquitectura Cliente-Servidor

Es la arquitectura más conocida por ser la más utilizada. La Figura 3 muestra la estructura sobre la que interaccionan los procesos clientes con los procesos del servidor.

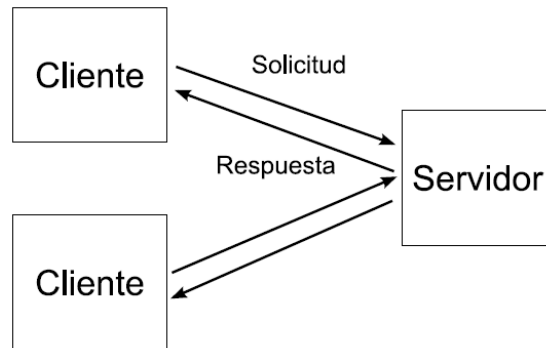


Figura 3: Arquitectura Cliente-Servidor.

En los Sistemas de Archivos Distribuidos esta arquitectura se manifiesta del siguiente modo: los clientes son los componentes encargados de almacenar los archivos y responder a las peticiones del componente servidor, que a su vez se encarga de gestionar tareas como:

- Determinar y tener control de en qué clientes se van a almacenar los archivos en cada momento, para cuando se solicite el archivo, esté disponible, de esta forma se satisface la característica deseable de la disponibilidad de los archivos.
- Establecer de qué manera se van a replicar los archivos.
- Tener constancia de las entradas y salidas de los clientes en el sistema, para la gestión de fallos y la escalabilidad del sistema.

Dentro de las desventajas de esta arquitectura se tiene que si el servidor fallara, el sistema colapsaría. Una de las alternativas a este problema, es el modelo Múltiples Servidores.

A continuación se describen algunos modelos que se derivan de esta arquitectura.

1.3.2. Modelo Múltiples Servidores

Este modelo se emplea como posible solución a la desventaja de la arquitectura Cliente-Servidor, mencionada anteriormente. En este modelo los servicios se implementan como procesos ubicados en diferentes computadoras que interactúan entre sí cuando es necesario para proporcionar el servicio al cliente, como se puede observar en la Figura 4. Los procesos que integran el servicio pueden estar distribuidos o replicados en varias máquinas. El uso de la replicación en este modelo se utiliza para

aumentar el desempeño y la disponibilidad para mejorar la tolerancia a fallos. Los servidores se distribuyen las tareas entre ellos y tienen una copia del estado del sistema, en caso de fallar alguno de estos componentes, el sistema continuaría funcionando (2).

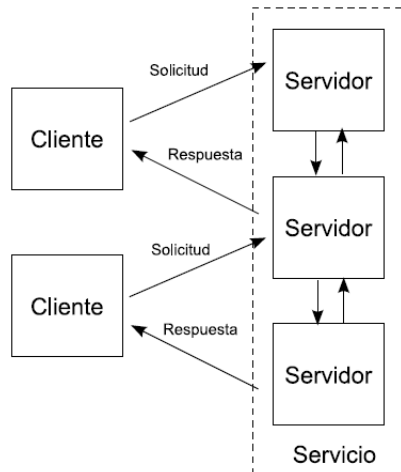


Figura 4: modelo Múltiples Servidores.

Como desventaja del empleo de este modelo se puede señalar que es una solución costosa, además de existir una sobrecarga en la red por la constante sincronización entre los servidores.

1.3.3. Modelo Servidores Proxy y Cachés

Una caché es un almacén de objetos de datos utilizados recientemente, que se encuentra más próximo que los objetos en sí. Al recibir un objeto nuevo en una computadora se añade al almacén de la caché, reemplazando, si fuera necesario, algunos objetos existentes.

Cuando se necesita un objeto en un proceso cliente, el servicio caché comprueba la caché y proporciona una copia actualizada del objeto al cliente; si no, se buscaría una copia actualizada. Las cachés pueden estar en cada cliente o en servidores proxy. Los servidores proxy proporcionan una caché compartida de recursos a las máquinas clientes de uno o más sitios como se muestra en la Figura 5 (2).

En un Sistema de Archivos Distribuidos los servidores proxy serían los encargados de almacenar los archivos que se están utilizando, para que el acceso sea más rápido y directo; también pudiera tener copias de los archivos más solicitados y de los utilizados recientemente.

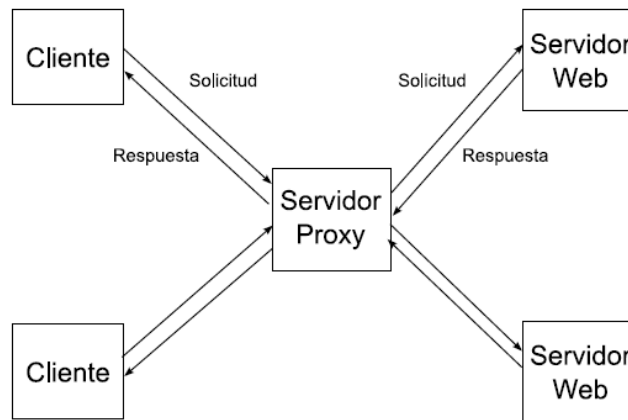


Figura 5: modelo Servidores Proxy y Cachés.

Como desventaja de este modelo se puede señalar que es una opción más costosa y que la eficiencia depende de las estrategias que se utilicen para los archivos que van a tener copias en los servidores proxy.

1.3.4. Arquitectura Procesos de Igual a Igual

La arquitectura de Procesos de Igual a Igual o punto a punto ha ido ganándose un espacio dentro de las arquitecturas que utilizan los Sistemas de Archivos Distribuidos. Su principal característica es que no existe diferencia entre componentes servidores y clientes, cuando se aplica en su forma “pura”, pues existen otros modelos que se derivan de esta arquitectura en los cuales se hace uso de servidores. Todos los componentes del sistema realizan la misma función, tratar de mantener comunicación con la mayoría de los componentes posibles, para en caso de alguna falla, el sistema continúe funcionando. Esto constituye una mejora con respecto a la arquitectura Cliente-Servidor, presenta mayor robustez en cuanto a la tolerancia a fallos.

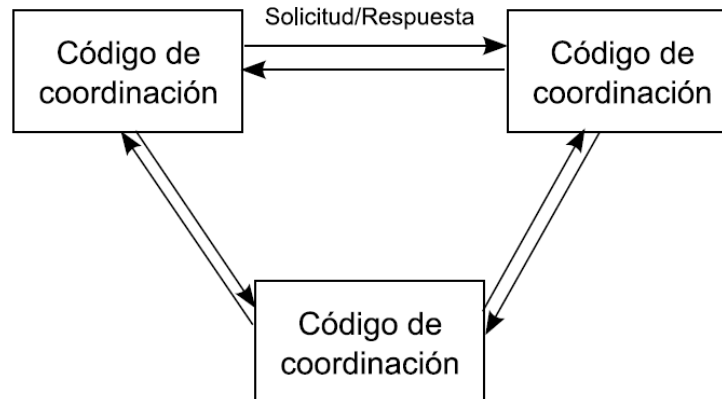


Figura 6: arquitectura Procesos de Igual a Igual

Como desventaja se puede señalar que su implementación es muy compleja, quizás esta sea una de las razones por la que se encuentren menos Sistemas de Archivos Distribuidos con este tipo de arquitectura.

2. Estudio valorativo.

En esta sección se presenta un análisis de los diferentes Sistemas de Archivos Distribuidos estudiados para el desarrollo de este trabajo, las principales características de la metodología y herramientas de desarrollo utilizadas, previamente establecidas por el Grupo de Arquitectura del Portal de Servicios Bioinformáticos.

2.1. Sistemas de Archivos Distribuidos existentes.

En este punto se presentan las principales características de los Sistemas de Archivos Distribuidos estudiados durante esta investigación, con el objetivo de determinar el más conveniente para integrarlo al Portal de Servicios Bioinformáticos.

2.1.1. Sistema de Almacenamiento Masivo para Archivos Pequeños (MSFSS).

Sistema de Almacenamiento Masivo para Archivos Pequeños (MSFSS, por sus siglas en inglés) es un sistema realizado por Lihua Yu, está dirigido al almacenamiento masivo de pequeños archivos y basado en la arquitectura Cliente-Servidor, específicamente el modelo Servidores Proxy y Cachés. Uno de los

aspectos más importantes de este sistema es que para evitar los cuellos de botella, se separa el manejo de los metadatos, de la transferencia de los archivos (3).

Algunas de las ventajas que ofrece son:

- **Migración de datos:** consiste en migrar los datos a servidores que mejoren el rendimiento y servicio del sistema.
- **Caché de archivos:** consiste en almacenar aquellos archivos accedidos frecuentemente en una caché, que permita un acceso más rápido a estos archivos.
- **Replicación:** consiste en mantener copias de los archivos en los diferentes servidores de almacenamiento registrados en el sistema, con el fin de mantener la consistencia de los archivos ante el fallo de un servidor.

Los principales componentes de este sistema son:

- **Interfaz de Sistema de Archivos:** (FSI, por sus siglas en inglés). Es una Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) que ofrece los servicios de almacenamiento a los clientes.
- **Máster:** es el componente medular del sistema que se encarga de la administración de los archivos almacenados en los diferentes servidores registrados en el sistema.
- **Servidor de Metadatos:** (MDS, por sus siglas en inglés). Servidor que se encarga de almacenar los metadatos de los archivos almacenados.
- **Nodos de Almacenamiento:** (SN, por sus siglas en inglés). Son los servidores de almacenamiento registrados en el sistema.

Como desventaja para su uso en esta investigación se puede señalar que el sistema no es flexible en cuanto al tamaño de los archivos, siendo óptimo su desempeño para archivos pequeños y no para archivos grandes.

2.1.2. Sistema de Archivos de Google.

El Sistema de Archivo de Google (GFS, por sus siglas en inglés) desarrollado por Google Inc. Para su

propio uso, está formado por tres componentes:

- **Máster.** Se encarga de la administración de los archivos y de los metadatos de los archivos.
- **Servidores de Segmentos.** Son los servidores de almacenamiento, los cuales se ejecutan sobre un sistema operativo Linux.
- **Clientes.** Son quienes se benefician de los servicios dados por el sistema.

Una de las virtudes de este sistema es que los archivos a ser almacenados en el sistema son divididos en bloques llamados segmentos. Cada segmento está identificado por un nombre de 64 bits asignado por el Máster al momento de ser creado. Estos segmentos son almacenados en los Servidores de Segmentos, mientras que el Máster crea réplicas y las guarda en diferentes servidores de segmentos, almacenando el metadato creado para el archivo almacenado (4).

Éste es uno de los Sistemas de Archivos Distribuidos de mayor prestigio en el mundo, pero tiene como desventaja que es de licencia propietaria por Google.

2.1.3. GridExpand.

Sistema Distribuido desarrollado por J.M. Pérez, F. García, J. Carretero, A. Calderón y J. Fernández donde se integran servidores de almacenamiento disponibles en Internet. Esto se logra gracias a la efectiva integración de los protocolos y los servicios ofrecidos por los servidores.

La arquitectura general del sistema GridExpand cuenta con los siguientes componentes:

- El servidor GridExpand, quien se encarga de la administración de los metadatos y de los archivos a almacenar.
- Los servidores que se encuentran alojados en diferentes redes conectadas a Internet.
- Clientes del sistema, los cuales forman el último componente.

La implementación de este sistema consiste en dividir los archivos en pequeños archivos y almacenarlos en los distintos servidores registrados al sistema, siendo esto transparente para el cliente. La comunicación entre los clientes y los servidores es por medio de una API la cual es ocupada por las

diferentes aplicaciones usadas por los clientes (5).

Como desventaja para su uso en esta investigación se identificó que el sistema no hace un uso óptimo de la topología de la red. Al estar diseñado para servidores que se encuentran en internet, no tiene en cuenta las localizaciones de los servidores y clientes. Por lo que no es capaz de determinar qué recursos son los más apropiados para almacenar determinados archivos o desde que recursos copiar los archivos que se están solicitando, afectando esto su rendimiento.

2.1.4. Sistema de Archivos Distribuidos Hadoop.

Hadoop fue creado por Doug Cutting en el año 2002. Con el tiempo se ha convertido en una de las principales herramientas para el análisis y almacenamiento de grandes cantidades de datos. El Sistema de Archivos Distribuidos Hadoop (HDFS, por sus siglas en inglés) es un Sistema de Archivos Distribuidos para ejecutarse en variedades de hardware. Éste tiene muchas similitudes con otros Sistemas de Archivos Distribuidos existentes, aunque, algunas diferencias pueden ser significantes. HDFS es muy bueno en cuanto a tolerancia a fallos y está diseñado para poder desplegarse en hardware de bajo costo. A continuación se describen las principales características de HDFS.

- **Detección de fallo de hardware:** el desplegar un HDFS puede consistir en cientos o miles de estaciones de trabajo, cada una almacenando partes de los datos del Sistema de Archivos Distribuidos. Por lo que existen una gran cantidad de componentes y cada uno de estos tiene una determinada probabilidad de fallar, lo que significa que algún componente en cierto momento estará en estado no funcional. Por ello, que la detección de fallos sea rápida y se realice una recuperación automática, son los principales objetivos de la arquitectura de HDFS.
- **Modelo de coherencia simple:** las aplicaciones que se ejecutan en HDFS necesitan escribir-solo uno-leer-varios, modelo de acceso a archivos. Los archivos una vez creados y escritos necesitan ser cerrados para modificarlos. Esto simplifica la coherencia de acceso a datos.
- **Portabilidad a través de hardware y software heterogéneos:** HDFS ha sido diseñado para ser fácil de llevar de una plataforma a otra y a variedades de hardware.
- **Namenodes y Datanodes:** HDFS tiene una arquitectura amo/esclavo. Un cluster HDFS consiste en una estación de trabajo llamada *Namenode*, que es un servidor que maneja el espacio de

nombres del sistema de archivos y regula el acceso de los usuarios a estos. Por otro lado pueden existir un gran número *Datanodes*, usualmente uno por estaciones de trabajo, en los que se almacenan los datos.

- **Réplica de datos:** HDFS está diseñado para almacenar los datos a través de todas las estaciones de trabajo. Para esto cada archivo es almacenado como una secuencia de bloques; todos los bloques de un archivo tienen el mismo tamaño, excepto el último que puede ser menor. Los bloques de un archivo son replicados para la tolerancia a fallos. Una aplicación puede especificar el número de réplicas a un archivo, esto puede ser definido en el momento de creación del archivo y cambiado posteriormente.

HDFS es uno de los Sistemas de Archivos Distribuidos de mayor prestigio en la actualidad, en este punto se puede destacar que es utilizado en compañías de mucho prestigio como Amazon, Facebook y Yahoo; además en abril del 2008 rompió el récord de ordenar un terabyte de datos convirtiéndose en el sistema más rápido, con 209 segundos (6).

2.2. Metodología de desarrollo.

Existen numerosas metodologías para el proceso de desarrollo de software. Por una parte están aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto. La otra aproximación es centrarse las dimensiones relacionadas con el factor humano o el producto. Éstas son conocidas como metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas (7).

La metodología que se emplea en esta investigación define las fases, actividades y artefactos que se deben generar durante el ciclo de vida del software. Teniendo en cuenta su aplicación a un proceso de

desarrollo basado en líneas de productos de software. Está basada en la metodología OpenUP.

2.2.1. OpenUP.

OpenUP es una metodología ágil que aplica un enfoque iterativo e incremental dentro de su estructura del ciclo de vida del proyecto y que puede adaptarse para desarrollar diversos tipos de proyectos (8).

Es un proceso mínimo y suficiente, lo que significa que solo el contenido fundamental y necesario es incluido. Por lo tanto no provee lineamientos para todos los elementos que se manejan en un proyecto pero tiene los componentes básicos que pueden servir de base a procesos específicos. La mayoría de los elementos de OpenUP están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances.

Dentro de los principales objetivos de OpenUP se encuentran:

- Colaborar para sincronizar intereses y compartir conocimiento. Este principio promueve prácticas que impulsan un ambiente de equipo saludable, facilitan la colaboración y desarrollan un conocimiento compartido del proyecto.
- Equilibrar las prioridades para maximizar el beneficio obtenido por los interesados en el proyecto. Este principio promueve prácticas que permiten a los participantes de los proyectos desarrollar una solución que maximice los beneficios obtenidos y que cumpla con los requisitos y restricciones del proyecto.
- Centrarse en la arquitectura de forma temprana para minimizar el riesgo y organizar el desarrollo.
- Desarrollo evolutivo para obtener retroalimentación y mejoramiento continuo. Este principio promueve prácticas que permiten a los equipos de desarrollo obtener retroalimentación temprana y continua de los participantes del proyecto, permitiendo demostrarles incrementos progresivos en la funcionalidad.

2.3. Herramientas y entorno de desarrollo.

2.3.1. Redmine

Redmine es una flexible aplicación Web para la gestión de proyectos, implementada usando el lenguaje

Ruby y el framework Rails, es una aplicación de código abierto bajo la licencia GNU Licencia Pública General (GPL, por sus siglas en inglés). Dentro de sus principales características se encuentran:

- Soporte de múltiples proyectos.
- Flexibilidad en el control de acceso basado en roles.
- Sistema de seguimientos a diversos temas.
- Soporte a diagrama de Gantt y calendario.
- Noticias, documentos y administración de archivos.
- Notificaciones por correo electrónico.
- Documentación por proyectos.
- Foros por proyectos.
- Integración con SVN, CVS, Git, Mercurial, Bazaar y Darcs.
- Soporte de autenticación LDAP.
- Soporte plurilingüe.
- Soporte a la mayoría de los gestores de base de datos.

2.3.2. Visual Paradigm for UML 6.4

Visual Paradigm for UML es una herramienta de Ingeniería de Software Asistida por Computadora (CASE, por sus siglas en inglés) que da soporte al modelado visual con Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) y ofrece un entorno de creación de diagramas para UML, con un diseño centrado en casos de uso y enfocado al negocio, que generan un software de mayor calidad. Permite modelado visual del análisis y el diseño, incluyendo sus diagramas, posibilidad de configuración de estilos y formatos de la documentación, que genera automáticamente la herramienta.

2.3.3. NetBeans IDE 6.9

NetBeans IDE es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso.

- **Búsqueda rápida:** se pueden realizar búsquedas de términos, símbolos y tipos, que pueden ser clasificados.
- **Gestión de plugin:** se pueden adicionar, eliminar o actualizar una serie de funcionalidades gracias a este mecanismo de gestión de plugin.
- **Personalización de proyectos:** establecer propiedades y dependencias entre los proyectos y hacer uso de librerías, guardar la configuración de distintos proyectos que se encuentran en diferentes fases.
- **Plantillas y ejemplos de aplicaciones:** comenzar proyectos desde ejemplos o plantillas existentes de diferentes tipos, como: aplicaciones Web, móviles, en C, C++, Java, entre otros.
- **Base de datos y servicios:** integración de acceso a la base de datos y los servicios Web, se pueden iniciar y detener los servicios, así como adicionarlos a un proyecto.

Para esta investigación se utiliza la versión 6.9 de NetBeans IDE.

2.3.4. Karmasphere Studio

Kamasphere Studio es un plugin para el Entorno de Desarrollo Integrado Netbeans, que permite crear servicios de Hadoop, tanto Map Reduce como HDFS. Incorpora las bibliotecas de clases necesarias para crear proyectos de este tipo, así como el autocompletado y depurado de código.

2.3.5. Portlet

Un Portlet es un componente Web gestionado por un contenedor que tras la petición de un usuario genera y presenta contenidos dinámicos de forma identificable, como componentes de contenidos en la interfaz de usuario del portal. Estas mini-aplicaciones Web interactivas se integran en los portales añadiendo cada vez más funcionalidades. También permiten la personalización, la presentación y la gestión de la seguridad; además proporcionan un objeto PortletPreferences para almacenar las preferencias de los usuarios. Estas preferencias son almacenadas en una base de datos persistente, así se encontrarán disponibles cada vez que el contenedor de portlets se reinicie.

El contenido generado por los portlets se denomina fragmento. Estos fragmentos son código XHTML, HTML, WML, entre otros. Los fragmentos agregados resultantes de la operación de varios portlets

constituyen un documento que se traduce en la interfaz del portal, los portlets al igual que los contenidos tienen su propio ciclo de vida.

2.3.6. Liferay

Liferay es un contenedor de portlets y a la vez un Sistema de Gestión de Contenidos (CMS, por sus siglas en inglés) de código abierto hecho en java. Se puede ejecutar en la mayoría de los servidores de aplicaciones y contenedores de servlets, bases de datos y en múltiples plataformas. Permite a los usuarios mover elementos diferentes en todo el portal con solo arrastrarlos, además de contar con una interfaz amigable. Por lo general las tareas que consumen mucho tiempo como la modificación de un diseño de página, añadir nuevas aplicaciones y contenidos, se pueden llevar a cabo relativamente fácil, sin tener que actualizar la página.

Dentro de sus principales características se encuentran:

Fiabilidad

Está configurado para entornos que requieren de la tolerancia a fallos y balance de carga. Además permite escalar un sistema eficientemente a medida que crece.

Rendimiento

Ha sido optimizado para funcionar con los principales servidores de aplicaciones y bases de datos, ofreciendo un rendimiento óptimo en carga. Configurado a medida para cubrir las necesidades críticas de velocidad y escalabilidad de las instalaciones en las grandes empresas.

2.3.7. Spring

Spring es un framework de código abierto para el desarrollo de aplicaciones en la plataforma Java, también existe una versión para la plataforma .NET, Spring .NET. Dentro de las ventajas que ofrece Spring, se puede destacar que facilita la manipulación de los objetos que usen EJBs o no, reduce la proliferación de Singletons, elimina la necesidad de usar variados tipos de ficheros de configuración, mejora la práctica de programación, permite el uso o no de EJBs, realizando el mismo tipo de funciones sin ellos. Una ventaja de Spring es su modularidad, pudiendo usar algunos de los módulos sin

comprometerse con el uso del resto. Está enfocado en el manejo de objetos de negocio, dentro de una arquitectura en capas.

2.3.8. ExtJS 4.0

El framework ExtJs comenzó por ser una extensión de la popular librería Interfaz de Usuario Yahoo (YUI por sus siglas en inglés). Luego los desarrolladores de ExtJs crearon una comunidad de código abierto, uniendo los conocimientos de varios programadores Web, al punto que en la actualidad es una de las librerías más poderosas para el desarrollo de interfaces de usuarios Web en todo el mundo.

Las principales características de ExtJs son:

- Provee funcionalidades sencillas de utilizar para hacer páginas con apariencia de sistemas de escritorio, que pueden incluir ventanas, tablas y vistas muy divertidas capaces de interactuar en diferentes navegadores de la misma forma, sin necesidad de realizar ningún cambio.
- Interactúa con el usuario y el navegador, vía el “Manejador de Eventos”, respondiendo a las órdenes del usuario a través del teclado y el ratón.
- Capacidad de comunicación con el servidor paralelamente, sin necesidad de tener que refrescar la página. Esto es posible usando la tecnología AJAX.

2.3.9. PostgreSQL.

PostgreSQL es un potente sistema gestor de bases de datos de código abierto. Cuenta con más de 15 años de desarrollo activo y una arquitectura probada que se ha ganado una sólida reputación. Soporta almacenamiento de objetos binarios de gran tamaño, como imágenes, sonidos y vídeo. Cuenta con características avanzadas tales como: control de concurrencia, punto en el tiempo de recuperación, replicación asincrónica, transacciones anidadas, copias de seguridad en caliente y un planeador de consultas sofisticadas. Es compatible con conjuntos de caracteres internacionales, codificaciones de caracteres multibyte y Unicode. Es altamente escalable, tanto en la enorme cantidad de datos que puede manejar y en el número de usuarios concurrentes que puede acomodar. Incluye un marco que permite a los desarrolladores definir y crear sus propios tipos de datos personalizados, junto con el apoyo a las funciones y los operadores que definen su comportamiento.

2.4. Lenguajes.

2.4.1. Lenguaje Unificado de Modelado (UML).

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés). Para comprender qué es el UML, basta con analizar cada una de las palabras que lo componen, por separado.

- Lenguaje: el UML es, precisamente, un lenguaje. Lo que implica que éste cuenta con una sintaxis y una semántica. Por lo tanto, al modelar un concepto en UML, existen reglas sobre cómo deben agruparse los elementos del lenguaje y el significado de esta agrupación.
- Modelado: el UML es visual. Mediante su sintaxis se modelan distintos aspectos del mundo real, que permiten una mejor interpretación y entendimiento de éste.
- Unificado: unifica varias técnicas de modelado en una única (9).

Debido a que el UML proviene de técnicas orientadas a objetos, se crea con la fuerte intención de que éste permita un correcto modelado orientado a objetos.

2.4.2. Java.

Java es un lenguaje simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portátil, de alto desempeño, dinámico y de hilos múltiples.

El compilador de Java genera bytecodes para la Máquina Virtual de Java (JVM, por sus siglas en inglés), en vez de código nativo de máquina. Para ejecutar un programa de Java, se debe usar el intérprete de Java para interpretar los bytecodes compilados. Como los bytecodes de Java no dependen de plataforma, los programas de Java se pueden ejecutar en cualquier plataforma en la que la JVM se haya instalado. Esta es una de las principales características de Java y muy importante para sistemas como los Sistemas de Archivos Distribuidos, donde se puedan adicionar estaciones de trabajo sin que el sistema operativo sea un problema. Además de esta característica de ser un lenguaje multiplataforma, Java cuenta con grandes cantidades de bibliotecas para casi cualquier tipo de software (10).

2.5. Conclusiones del capítulo.

Después de haber realizado un estudio de los Sistema de Archivos Distribuidos seleccionados, se tuvieron en cuenta diferentes puntos con la intención de elegir el indicado para integrarlo al Portal de Servicios

Bioinformáticos:

- **Robustez:** respuesta a prueba de fallos, capacidad del sistema de responder a alguna petición de un usuario, aún cuando algunas estaciones de trabajo no estén funcionando.
- **Escalabilidad:** manejo de altas y bajas de las estaciones de trabajo, capacidad de la herramienta para permitir extender el número de estaciones de trabajo, para así aumentar la cantidad de espacio y poder de cómputo.
- **API:** debido a que la herramienta se debe integrar al Portal de Servicios Bioinformáticos, es muy importante que cuente con una API que se encuentre bien documentada y probada, para poder acceder a sus servicios.
- **Documentación:** cantidad de información, ejemplos, libros, entre otros.
- **Reputación:** proyectos donde se ha aplicado la herramienta y cuáles son las impresiones y resultados de la misma.
- **Licencia de uso:** estado de su licencia de uso, si era necesario invertir recursos para poder utilizarla o si era libre.

Por estas razones se decidió que la herramienta que se iba a integrar al Portal de Servicios Bioinformáticos por sus características, sería Hadoop HDFS.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Durante el desarrollo del presente capítulo se realizará una descripción detallada de las características del sistema que ayudarán al lector a una mejor comprensión. Se detalla el funcionamiento del negocio a través de un Modelo de Dominio, en el que se precisan conceptos asociados su funcionamiento. Se exponen los requisitos funcionales y no funcionales a partir de los cuales se obtendrán los casos de uso y se realiza una descripción de éstos.

1. Modelo de Dominio.

1.1. Conceptos del dominio.

Estructura-Molecular

Archivo digital que contiene información sobre estructuras moleculares, tales como, sus enlaces, tamaños, aminoácidos, distintas estructuras, entre otras.

Investigador

Persona encargada de iniciar el flujo de eventos para gestionar los datos biológicos.

Portal de Servicios Bioinformáticos

Aplicación Web con la que interactúan los investigadores para gestionar los datos biológicos.

1.2. Diagrama de clases del Modelo de Dominio.

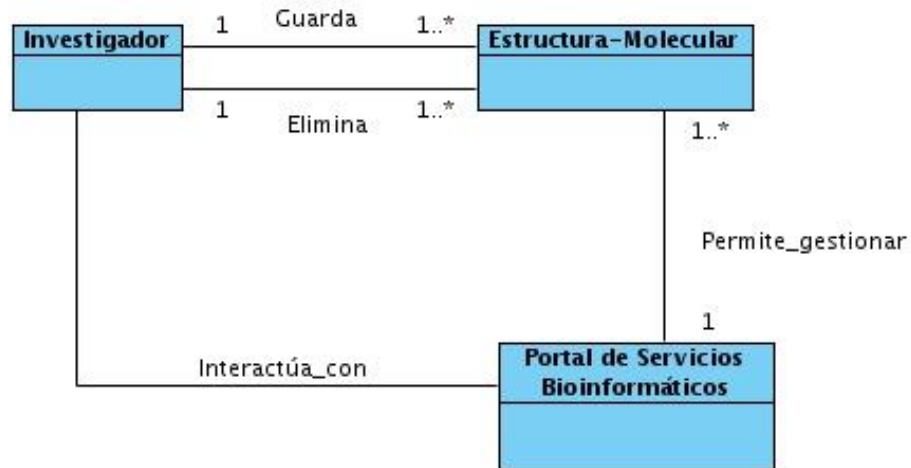


Figura 7: Diagrama de clases del Modelo de Dominio.

2. Especificación de requisitos.

Los requisitos o requerimientos son una condición o capacidad que tiene que ser alcanzada por un sistema o componente de un sistema para satisfacer un contrato, estándar u otro documento impuesto formalmente.

2.1. Requisitos funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. Estos se mantienen invariables sin importar con que propiedades o cualidades se relacionen.

RF1. Guardar estructuras moleculares en el sistema.

RF2. Buscar estructuras moleculares.

RF3. Eliminar estructuras moleculares.

RF4. Establecer valores a metadatos de las estructuras moleculares.

RF5. Modificar valores a metadatos de las estructuras moleculares.

RF6. Mostrar valores de metadatos de las estructuras moleculares.

2.2. Requisitos no funcionales.

Los requisitos no funcionales son propiedades o cualidades con las que el producto debe contar. Son una parte significativa de la especificación y son importantes para que clientes y usuarios puedan valorar las características del producto. Es conocido que si el mismo cumple con las funcionalidades requeridas, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable sea el sistema, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación.

Usabilidad

RNF1. La aplicación debe contar con una interfaz amigable donde el usuario pueda navegar de una sesión a otra en menos de tres clics.

RNF2. Es posible llevar a cabo cualquier tarea dada con sólo usar el teclado, sin hacer uso del ratón.

Fiabilidad

RNF3. Los archivos que describen las estructuras moleculares deben almacenarse íntegramente en el Sistema de Archivos Distribuidos.

RNF4. El sistema debe mantener la integridad de los archivos almacenados.

Soporte

RNF5. El sistema debe permitir aumentar la capacidad de almacenamiento sin que deje de prestar los servicios.

Restricciones del diseño

RNF6. Para el desarrollo del sistema se utilizará **Java** como lenguaje de programación.

RNF7. Se utilizará **Spring** como framework.

RNF8. La aplicación que sirva como interfaz de usuario al sistema debe ser implementada usando la tecnología **Portlet**.

Software

RNF9 Las estaciones de trabajo donde se desplegará el Sistema de Archivos Distribuidos deben contar con la Máquina Virtual de Java en su versión 1.6.

RNF10 Las estaciones de trabajo donde se desplegará el Sistema de Archivos Distribuidos deben contar con el servicio de SSH.

Hardware

RNF11 Las estaciones de trabajo donde se desplegará el Sistema de Archivos Distribuidos deben contar como mínimo con:

- 512 MB de memoria RAM.
- Procesador Pentium IV.

Estándares Aplicables

RNF12 Se utilizará OpenUP como metodología y el lenguaje UML 2.1 para visualizar, especificar, construir y documentar los artefactos del sistema.

3. Modelado del sistema.

3.1. Actores del sistema.

Actor	Descripción
<p>Usuario</p>	<p>Persona encargada de interactuar con la aplicación realizando diferentes acciones.</p> <p>Acciones:</p> <ul style="list-style-type: none"> • Guardar estructuras moleculares. • Buscar estructuras moleculares. • Eliminar estructuras moleculares. • Establecer valores a metadatos de las estructuras moleculares.

	<ul style="list-style-type: none"> • Modificar valores a metadatos de las estructuras moleculares.
SAD-HDFS	<p>Sistema de Archivos Distribuidos que presta servicios para gestionar y almacenar de forma transparente archivos, además de otros servicios de mantenimiento.</p> <p>Servicios:</p> <ul style="list-style-type: none"> • Gestionar archivos (crear, eliminar, modificar, almacenar, obtener). • Replicar archivos.

3.2. Diagrama de Casos de Uso del sistema.

Los diagramas de casos de usos son representaciones gráficas en las que se muestran los requisitos funcionales de un sistema y su relación con el entorno. Los casos de uso son utilizados básicamente en el proceso de modelado de sistemas (Figura 8).

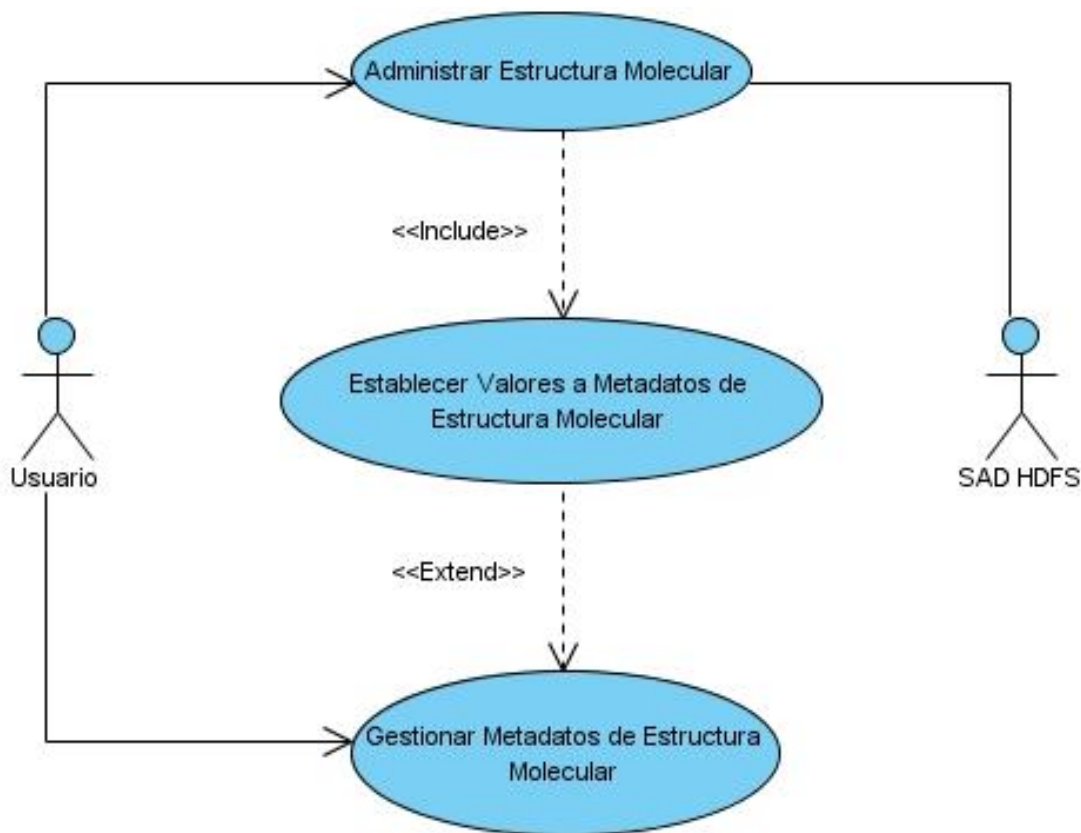


Figura 8: Diagrama de Casos de Uso del Sistema.

3.3. Especificación de los Casos de Uso.

3.3.1. Descripción del Caso de Uso: Administrar Estructura Molecular

Caso de Uso:	Administrar Estructura Molecular	
Actores:	<ul style="list-style-type: none"> • Usuario • SAD-HDFS 	
Resumen:	El caso de uso inicia cuando el actor Usuario selecciona en la interfaz Web una o más estructuras moleculares para ser eliminadas o selecciona la opción de guardar una estructura molecular. Se envían los cambios realizados al SAD-HDFS y termina el caso de uso.	
Escenario	Guardar Estructura Molecular	
Precondiciones:	El archivo que describe la estructura molecular debe encontrarse accesible desde la estación de trabajo a través de la cual el actor Usuario está interactuando con la interfaz Web.	
Postcondiciones	El archivo que describe la estructura molecular debe ser almacenado correctamente.	
Flujo de eventos		
	Actor	Sistema
1	Usuario Selecciona la opción "guardar estructura molecular".	
2		Muestra una interfaz de usuario en la que se selecciona el archivo que describe la estructura molecular que se desea

		guardar, además de un formulario para establecer los valores de los metadatos de la estructura molecular (Ver descripción de caso de uso incluido: Establecer Valores a Metadatos de Estructura Molecular).
3	<p style="text-align: center;">Usuario</p> Selecciona el archivo que desea guardar y establece los valores de los metadatos.	
4		Solicita a SAD-HDFS el servicio de almacenar archivo.
5	<p style="text-align: center;">SAD-HDFS</p> Ejecuta el servicio de almacenar archivo.	
6	<p style="text-align: center;">SAD-HDFS</p> Notifica que el servicio almacenar archivo se llevó a cabo correctamente.	
7		Muestra un mensaje informando que la estructura molecular se ha guardado correctamente.
Flujos alternos		
6 <Error al ejecutar el servicio almacenar archivo>		
6	<p style="text-align: center;">SAD-HDFS</p> Notifica que ha ocurrido un error durante	

	la ejecución del servicio almacenar archivo.	
7		Muestra un mensaje informando que ha ocurrido un error en el momento de guardar la estructura molecular.
Escenario	Eliminar Estructura Molecular	
Precondiciones:	Debe encontrarse almacenada en el sistema al menos una estructura molecular.	
Postcondiciones	Las estructuras moleculares seleccionadas deben ser eliminadas.	
Flujo de eventos		
	Actor	Sistema
	Usuario	
1	Selecciona las estructuras moleculares que desea eliminar y selecciona la opción “eliminar estructuras moleculares”.	
2		Muestra un mensaje de confirmación para eliminar las estructuras moleculares previamente seleccionadas.
	Usuario	
3	Selecciona la opción aceptar en el mensaje de confirmación.	
4	SAD-HDFS	

	Ejecuta el servicio de eliminar archivo.	
5		Solicita a SAD-HDFS el servicio de “eliminar archivo” repetidamente, donde en cada iteración le envía el nombre del archivo que representa la estructura molecular que se desea eliminar.
6	SAD-HDFS Notifica que el servicio eliminar archivo se llevó a cabo correctamente.	
		Muestra un mensaje informando que las estructuras moleculares se han eliminado correctamente.
Flujos alternos		
3 < Selecciona la opción cancelar en el mensaje de confirmación >		
3	Usuario Selecciona la opción cancelar en el mensaje de confirmación.	
4		Muestra la interfaz con las estructuras moleculares previamente seleccionadas.
6 < Error al solicitar el servicio eliminar archivo >		
6	SAD-HDFS Notifica que ha ocurrido un error durante la ejecución del servicio eliminar archivo.	

7	Muestra un mensaje informando las estructuras moleculares que no se eliminaron correctamente.
Referencias	RF1, RF2, RF3, RF4
Prioridad	Crítico

3.3.2. Descripción del Caso de Uso: Gestionar Metadatos de Estructura Molecular

Caso de Uso:	Gestionar Metadatos de Estructura Molecular	
Actor:	Usuario	
Resumen:	El caso de uso puede iniciar cuando el sistema muestra una interfaz Web con un componente en el que se muestran los metadatos de las estructuras moleculares, cuando el actor Usuario selecciona guardar una estructura molecular, o cuando selecciona un metadato para modificar su valor.	
Escenario	Mostrar Metadatos de Estructura Molecular	
Precondiciones:	Deben existir estructuras moleculares almacenadas previamente.	
Flujo de eventos		
	Actor	Sistema
1	Selecciona la opción “mostrar estructuras moleculares”.	
2		Muestra una interfaz de usuario que contiene un componente con los valores de los metadatos de las estructuras moleculares existentes, permitiendo al usuario hacer

		búsquedas filtradas por cada metadato.
3	<p>Usuario</p> <p>Establece los valores a los filtros de búsqueda del componente que muestra los valores de los metadatos de las estructuras moleculares.</p>	
4		Actualiza el componente que muestra los valores de los metadatos de las estructuras moleculares según el filtrado.
Escenario	Modificar Metadatos de Estructura Molecular	
Precondiciones:	Debe encontrarse seleccionado el metadato que se desea modificar.	
Flujo de eventos		
	Actor	Sistema
1	<p>Usuario</p> <p>Modifica el valor del metadato previamente seleccionado.</p>	
2		Establece el nuevo valor del metadato modificado.
Referencias	RF2, RF5, RF6	
Prioridad	Crítico	

3.3.3. Descripción del Caso de Uso: Establecer Valores a Metadatos de Estructura Molecular

Caso de Uso:	Establecer Valores a Metadatos de Estructura Molecular	
Actor:	Usuario	
Resumen:	El caso de uso inicia cuando el actor Usuario selecciona la opción “guardar estructura molecular”, luego se muestra un formulario donde se deben introducir los valores de los metadatos de las estructuras moleculares.	
Precondiciones:	El actor Usuario debe haber seleccionado la opción “guardar estructura molecular”.	
Flujo de eventos		
	Actor	Sistema
1		Muestra un formulario con los campos de los metadatos que se deben establecer.
2	Usuario Introduce en los campos del formulario los valores de los metadatos.	
3		Establece los valores de los metadatos a la estructura molecular.
Referencias	RF1, RF4	
Prioridad	Crítico	

3.4. Conclusiones del capítulo.

Teniendo en cuenta lo abordado durante este capítulo, se puede concluir que la complejidad de la solución propuesta no solo está dada por sus funcionalidades. La comprensión y despliegue de la herramienta SAD-HDFS, ocuparon gran parte del esfuerzo y tiempo dedicado a su desarrollo. Se realizó un estudio del negocio en el que se identificaron los principales conceptos del dominio. Se obtuvieron seis requisitos funcionales agrupados en tres casos de usos del sistema y doce requisitos no funcionales. Se llevó a cabo una descripción de los actores y casos de uso del sistema con el objetivo de poseer una visión detallada para próximas fases del ciclo de vida de desarrollo.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

El presente capítulo tiene como objetivo profundizar en los casos de usos detallándolos de manera que permitan reflejar una vista interna del sistema, descrita con un lenguaje entendible para los desarrolladores. En esta vista se plasman el análisis y diseño de los casos de uso y se determinan las clases necesarias para llevar a cabo las funcionalidades en ellos contenidos.

1. Modelo de análisis.

Durante el análisis se estudian los requisitos que fueron descritos en la captura de requisitos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que ayude a estructurar todo el sistema, incluyendo su arquitectura (11).

1.1. Diagramas de clases del análisis.

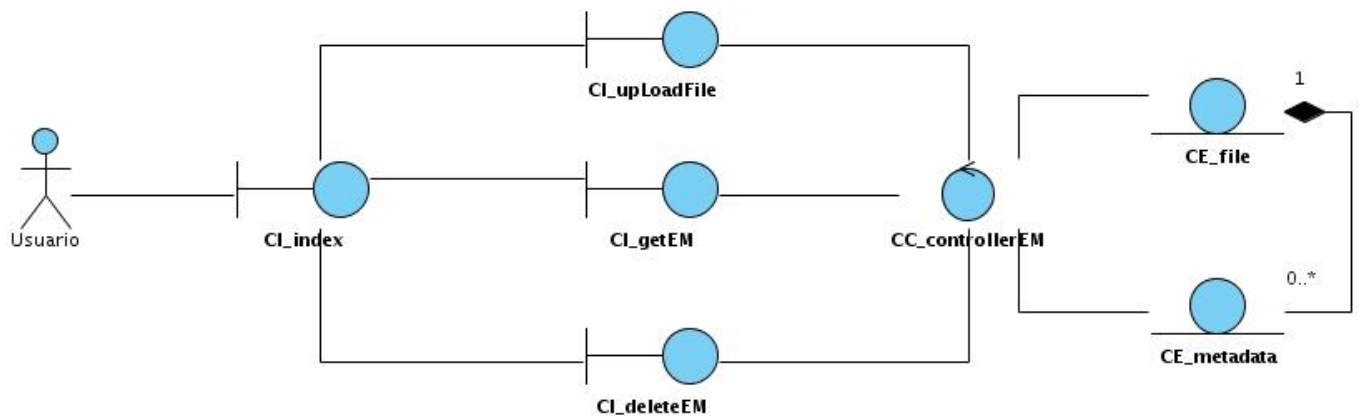


Figura 9: DCA CU Administrar Estructura Molecular.

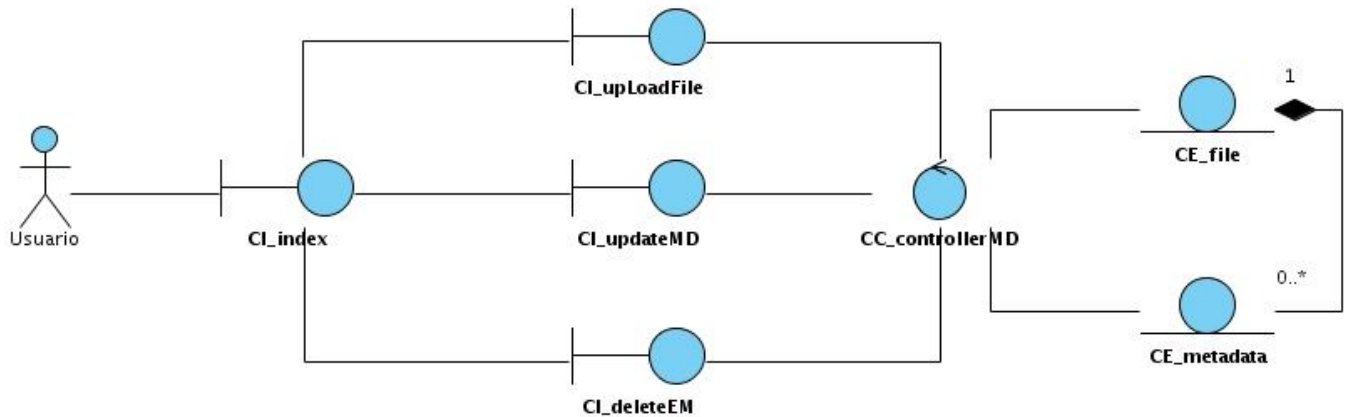


Figura 10: DCA CU Gestionar Metadatos de Estructura Molecular.

2. Arquitectura del sistema.

La Arquitectura de Software es a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se percibe desde el resto del sistema y las formas en que los componentes interactúan. Para la solución propuesta se utilizaron varios patrones de diseño enmarcados dentro del patrón de arquitectura de software Modelo-Vista-Controlador.

Modelo-Vista-Controlador

Vista: Administra la interacción del usuario con el sistema. Captura y comunica la información del usuario, ejecutando el mínimo de procesos y realizando un filtrado previo, para comprobar que no hay errores de formato. Está integrada básicamente por las interfaces de usuario.

Controlador: Es el encargado de recibir las peticiones de los usuarios e iniciar correctamente los procesos pertinentes y residentes en el Modelo, así como enviar las respuestas obtenidas de esa interacción a la Vista. Está integrada básicamente por clases controladoras que administran objetos y entidades fuertemente relacionadas.

Modelo: Está integrado por los objetos de acceso a datos, las abstracción de los datos y los objetos que agrupan funcionalidades complementarias, en fin la lógica del negocio. Es el encargado de almacenar,

adicionar y modificar los datos del negocio así como ejecutar los procesos necesarios para responder a las peticiones realizadas por el Controlador y que son resultado de la interacción con el usuario.

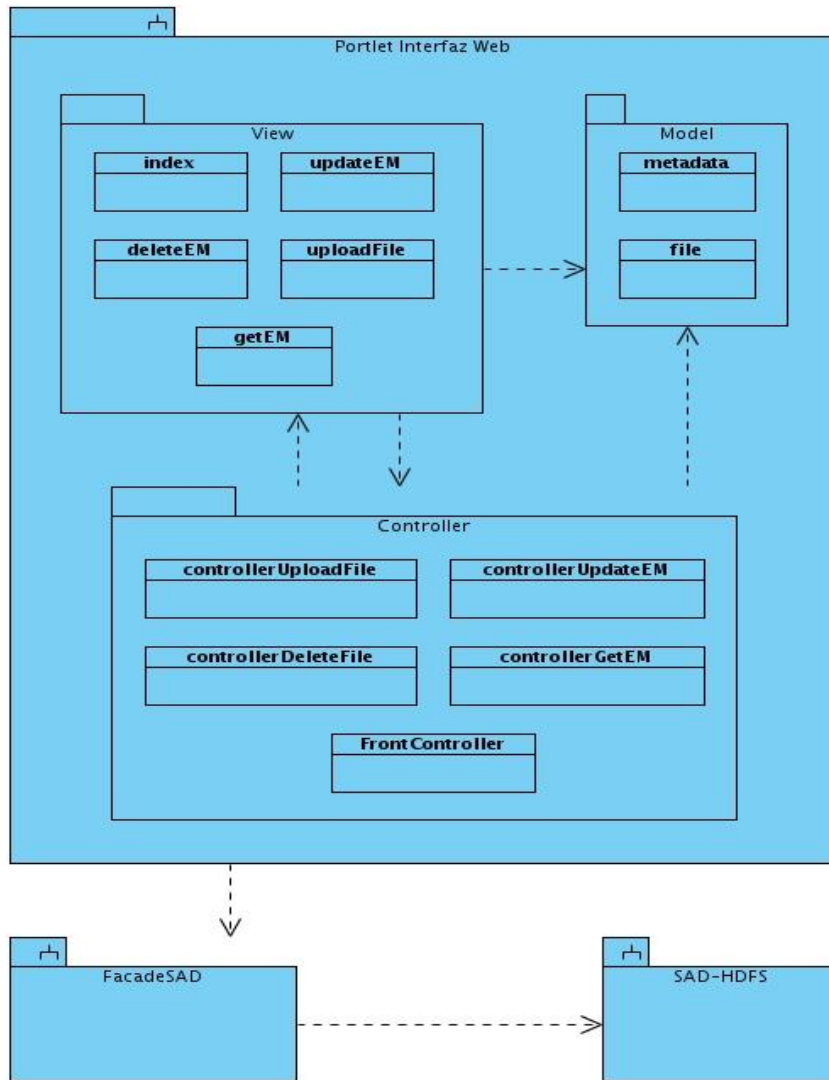


Figura 11: Vista de la Arquitectura.

La clara separación de la capa Modelo, Vista y Controlador brindan la posibilidad de realizar el mantenimiento del sistema de una forma más fácil. Además provee sencillez para crear distintas representaciones de los mismos datos. Facilidad para la realización de pruebas unitarias de los

componentes. Reutilización de los componentes, simplicidad y facilidad para desarrollar prototipos rápidos.

Fábrica

El patrón Fábrica se encuentra en el grupo de patrones de creación. Los patrones de creación, tal como su nombre indica, se dedican a crear objetos. Con ello se consigue disminuir el trabajo de los desarrolladores, elevar la productividad. La ventaja de utilizar este patrón de diseño es que se centraliza la creación de los objetos. El patrón Fábrica implementa la inyección de dependencias, la cual consiste en cargar las propiedades de cada clase mediante su constructor o sus métodos en el momento de iniciar la aplicación. Estos objetos se instancian una vez, se guardan y se comparten por todos los usuarios. Esto posibilita tener ese objeto creado y configurado listo para usarse desde cualquier clase que lo necesite.

Fachada

El patrón Fachada trata de simplificar la comunicación entre dos sistemas o componentes de software, ocultando un sistema complejo detrás de una clase que hace de pantalla o fachada.

La idea principal es la de ocultar todo lo posible la complejidad de un sistema, el conjunto de clases o componentes que lo forman, de forma que solo se ofrezcan determinados punto de entrada al sistema a través de la fachada.

Una ventaja de usar una fachada para comunicar las dos partes o componentes, es aislar los posibles cambios que se puedan producir en alguna de las partes. En el caso del sistema que se propone como solución en esta investigación, el objetivo era crear un subsistema que proporcionara servicios que posibilitaran interactuar con un Sistema de Archivos Distribuidos. Para lograrlo se utilizó éste patrón, donde el subsistema FacadeSAD (Figura 12) brinda los servicios a través de la clase FacadeSAD ocultando de esta forma la complejidad de este subsistema. De esta forma se garantiza el bajo acoplamiento entre el subsistema FacadeSAD y el que consume sus servicios, en este caso, el subsistema Portlet Interfaz Web.

3. Modelo de diseño.

En el diseño se modela el sistema para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Una entrada esencial en el diseño es el resultado del análisis, o sea, el modelo de análisis, que proporciona una comprensión detallada de los requisitos. Además impone una estructura del sistema que se debe conservar lo más fielmente posible cuando se desarrolle el sistema.

Principales propósitos del diseño:

- Crear una entrada apropiada y un punto de partida para actividades de implementación, capturando los requisitos o subsistemas individuales, interfaces y clases.
- Descomponer los trabajos de implementación en partes más manejables.

3.1. Diagramas de clases del diseño.

Un diagrama de clases del diseño representa las clases y subsistemas que serán utilizados dentro del sistema y las relaciones que existen entre ellos. A continuación se representan los principales diagramas de clases del diseño por casos de uso del sistema.

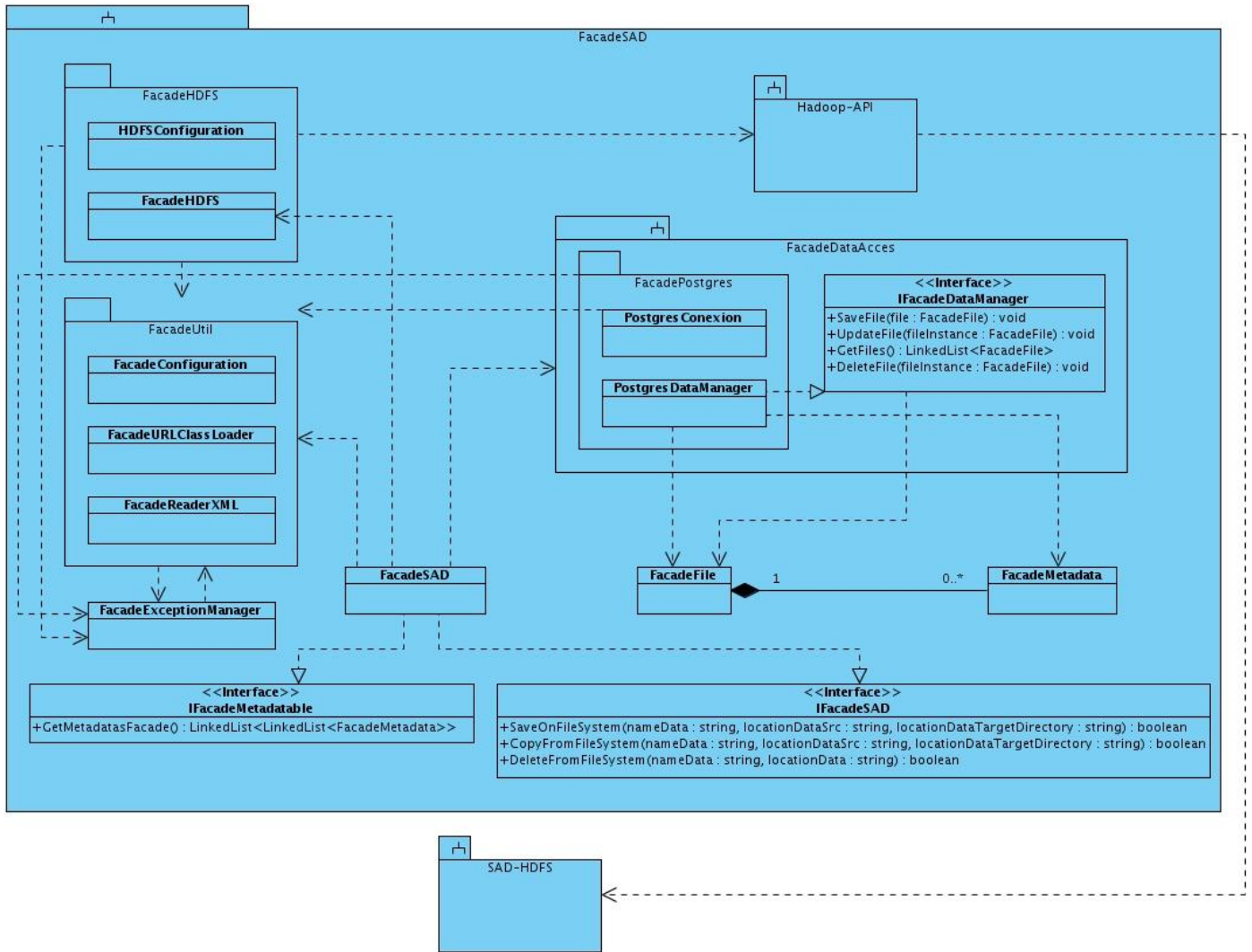


Figura 12: DCD Subsistema FacadeSAD.

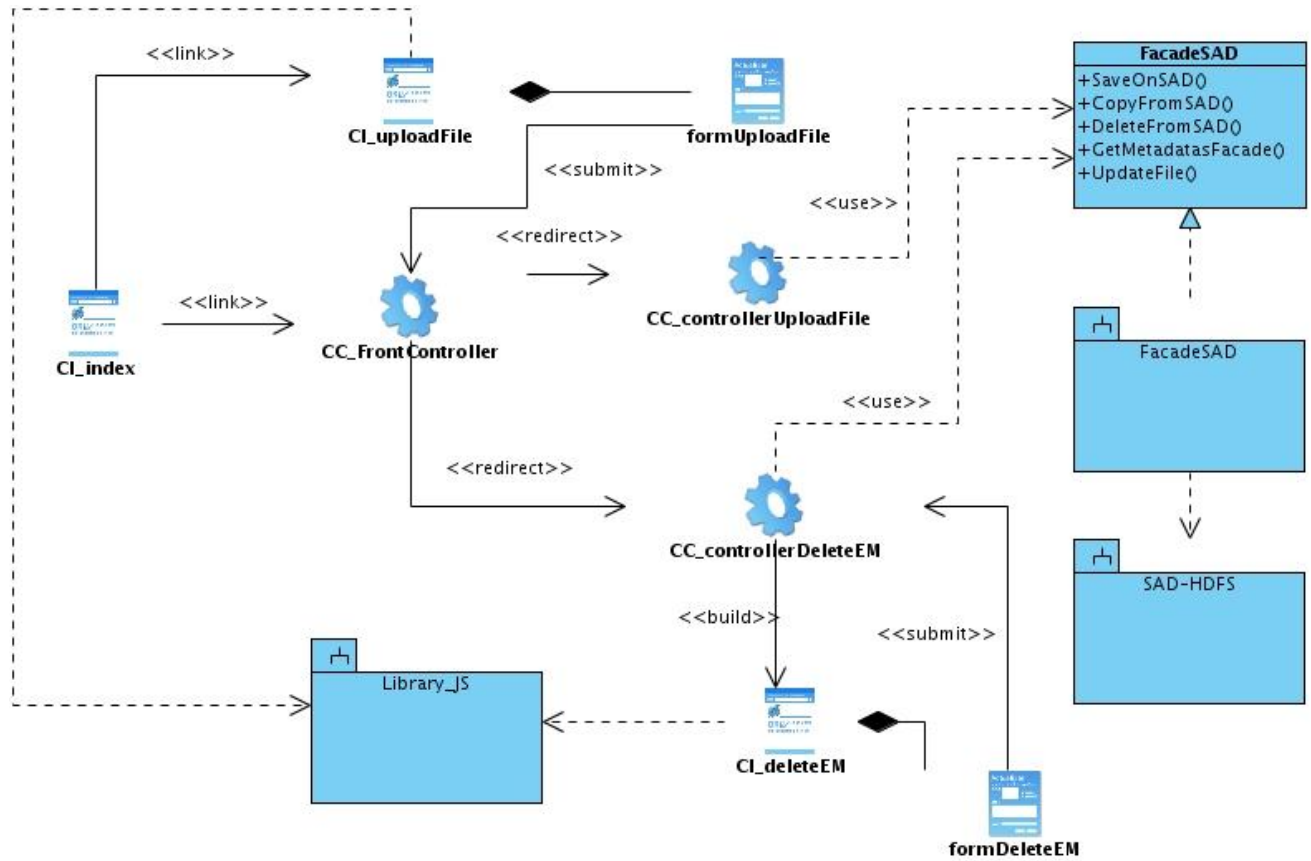


Figura 13: DCD CU Administrar Estructura Molecular.

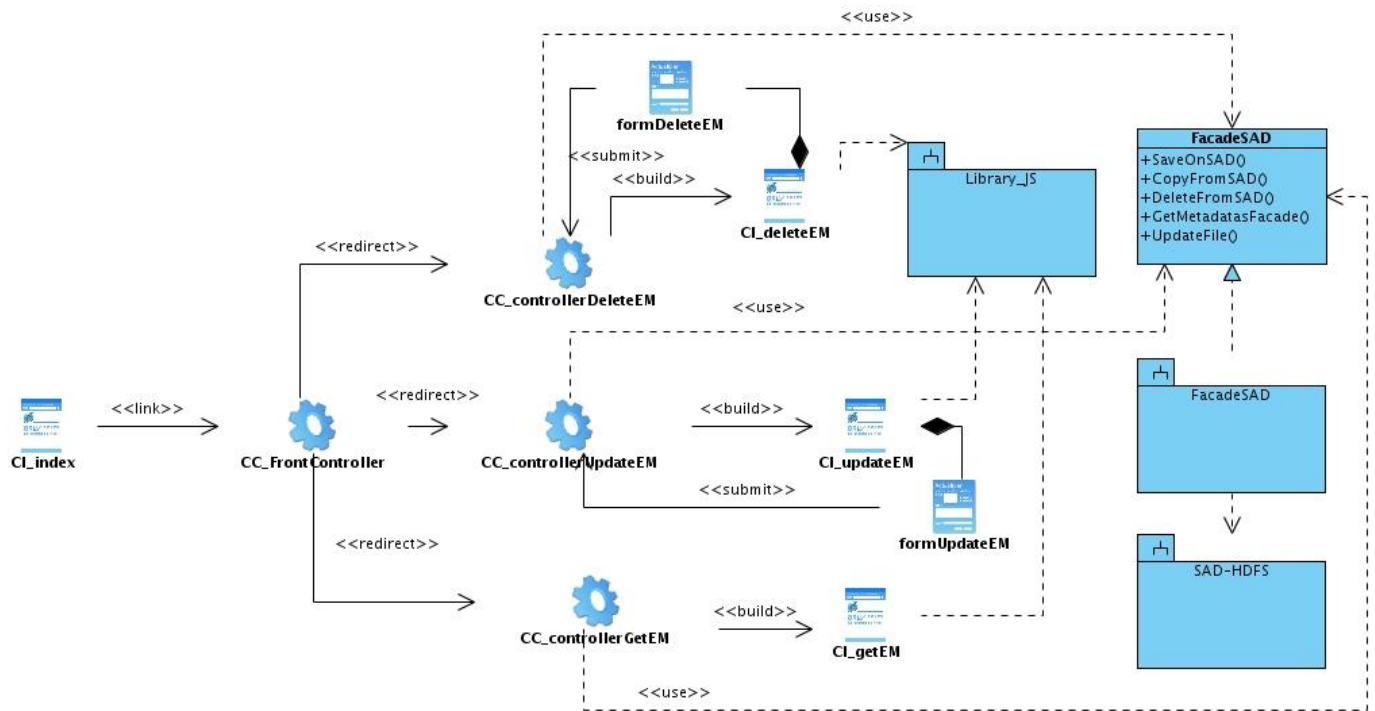


Figura 14: DCD CU Gestionar Metadatos de Estructura Molecular.

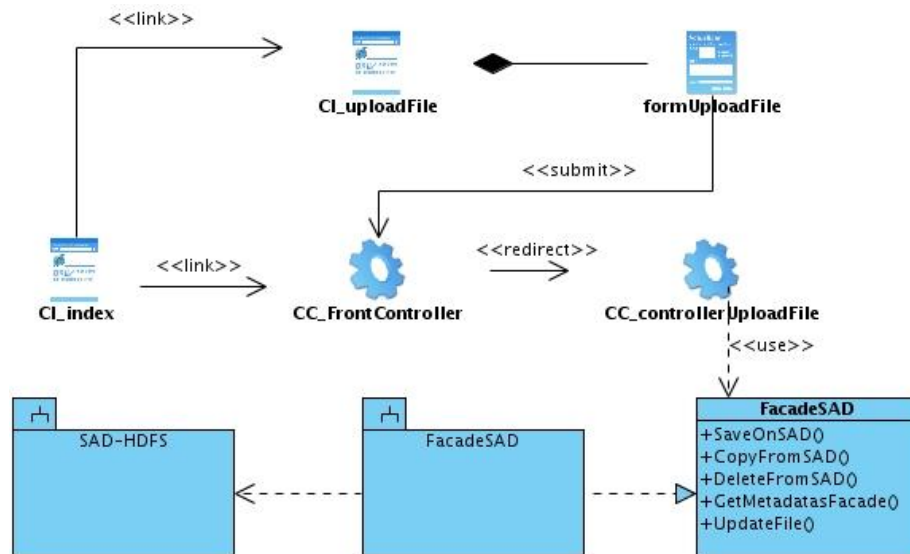


Figura 15: DCD CU Establecer Valores a Metadatos de Estructura Molecular.

3.2. Diagramas de secuencia.

Los diagramas de secuencia de un sistema muestran gráficamente los eventos que originan los actores y su impacto sobre el sistema. Describen las interacciones entre los actores y el sistema, mostrando de forma secuencial los envíos de mensajes entre ellos. A continuación se muestran los principales diagramas de secuencia por casos de uso del sistema.

3.2.1. Caso de Uso Administrar Estructura Molecular.

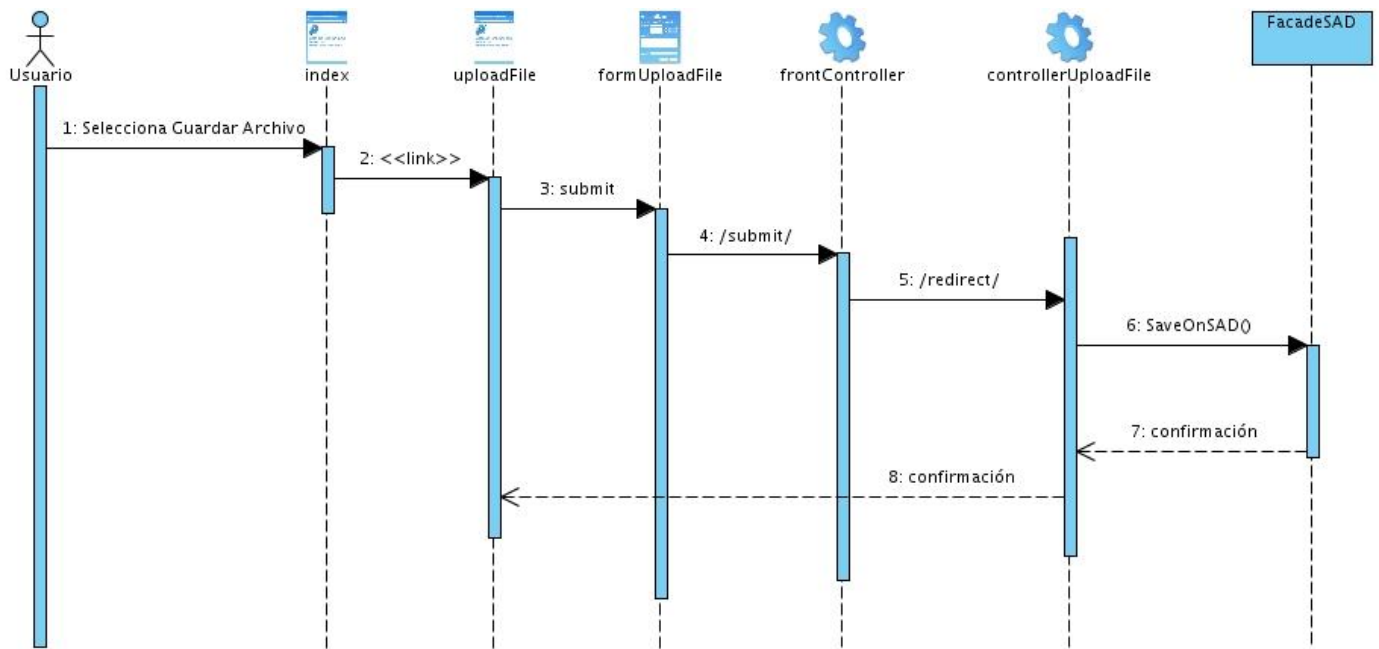


Figura 16: DS CU Administrar Estructura Molecular. Escenario: Guardar Estructura Molecular.

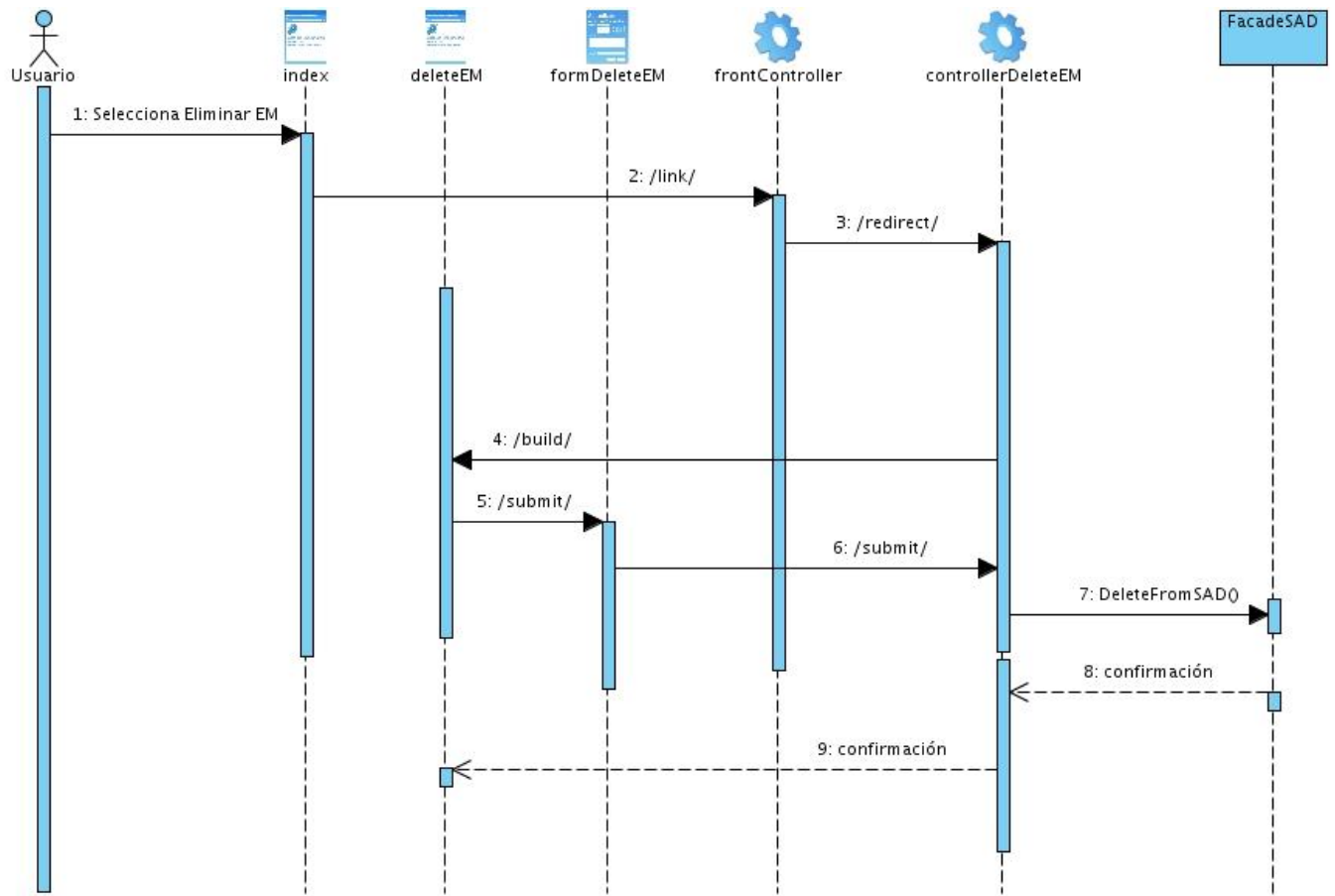


Figura 17: DS CU Administrar Estructura Molecular. Escenario: Eliminar Estructura Molecular.

3.2.2. Caso de Uso Gestionar Metadatos de Estructura Molecular.

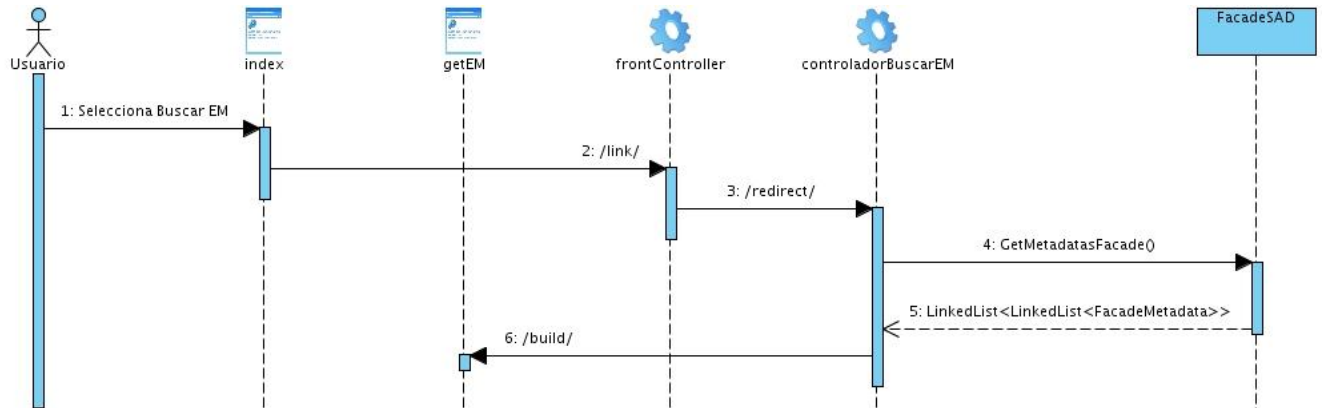


Figura 18: DS CU Gestionar Metadatos de Estructura Molecular. Escenario: Mostrar Metadatos de Estructura Molecular.

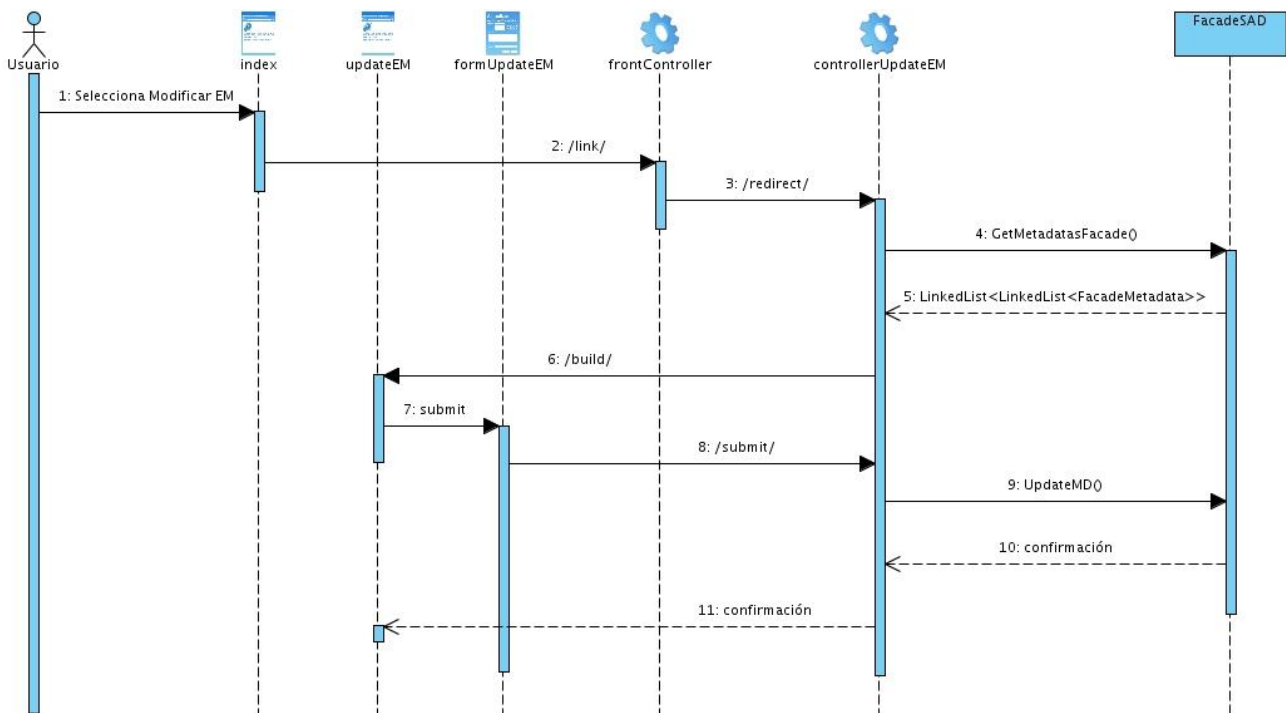


Figura 19: DS CU Gestionar Metadatos de Estructura Molecular. Escenario: Modificar Metadatos de Estructura Molecular.

4. Descripción de las clases principales del sistema.

Nombre: FacadeSAD

Descripción: Es la clase que implementa la interfaz IFacadeSAD para proporcionar los servicios:

- SaveOnSAD(nameData : String): boolean
- CopyFromSAD(nameData : String): boolean
- DeleteFromSAD(nameData : String): boolean

Nombre: FacadeHDFS

Descripción: Es la clase que sirve de intermediaria para la comunicación entre la clase FacadeSAD y el SAD-HDFS.

Nombre: file

Descripción: Es la clase donde se declara el método para guardar un archivo que se utiliza para subir un archivo al servidor. Además posee los atributos nombre, fecha y descripción. .

Nombre: frontController

Descripción: Es la clase encargada de asignar a las interfaces las clases controladoras que van a atender sus peticiones y las vistas que van a mostrar los resultados una vez satisfechas las

peticiones.

4.1. Conclusiones del capítulo.

Con el desarrollo de este capítulo se obtuvieron los diagramas de clases del análisis, clases del diseño y de secuencia de los subsistemas Interfaz Web y FacadeSAD, en los que se puede apreciar el uso de patrones de diseño como el Modelo-Vista-Controlador, Fachada, entre otros. De esta forma se arribó a una idea más cercana de cómo sería la integración entre la interfaz Web y el Sistema de Archivos Distribuidos.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

Con el desarrollo del presente capítulo se pretende mostrar cómo los elementos del Modelo de Diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el diagrama de despliegue. Durante la implementación se deberá solucionar a menudo, pequeños problemas relacionados con el entorno de implementación que no deberían afectar al Modelo de Diseño.

1. Modelo de Implementación.

1.1. Diagrama de componentes.

Los diagramas de componentes permiten modelar la vista estática de un sistema. Se representan como un grafo de componentes de software unidos por medio de relaciones de dependencia (compilación, ejecución) y las interfaces que estos soportan. No es necesario que un diagrama incluya todos los componentes del sistema, normalmente se realizan por partes, por lo cual cada diagrama describe una parte del sistema.

En un diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. El uso más importante de un diagrama de componentes es mostrar la estructura del Modelo de Implementación, específicamente:

- Subsistemas de implementación y sus dependencias.
- Los subsistemas de implementación organizados en capas.

Componente: Es la parte modular de un sistema, desplegable y reemplazable que encapsula implementación, un conjunto de interfaces y proporciona la realización de los mismos. Un componente típicamente contiene clases y puede ser implementado por uno o más artefactos (ficheros ejecutables, binarios, entre otros.). Son las piezas reutilizables de alto nivel a partir de las cuales se pueden construir

sistemas (12). A continuación se muestra los Diagramas de Componentes del Sistema.

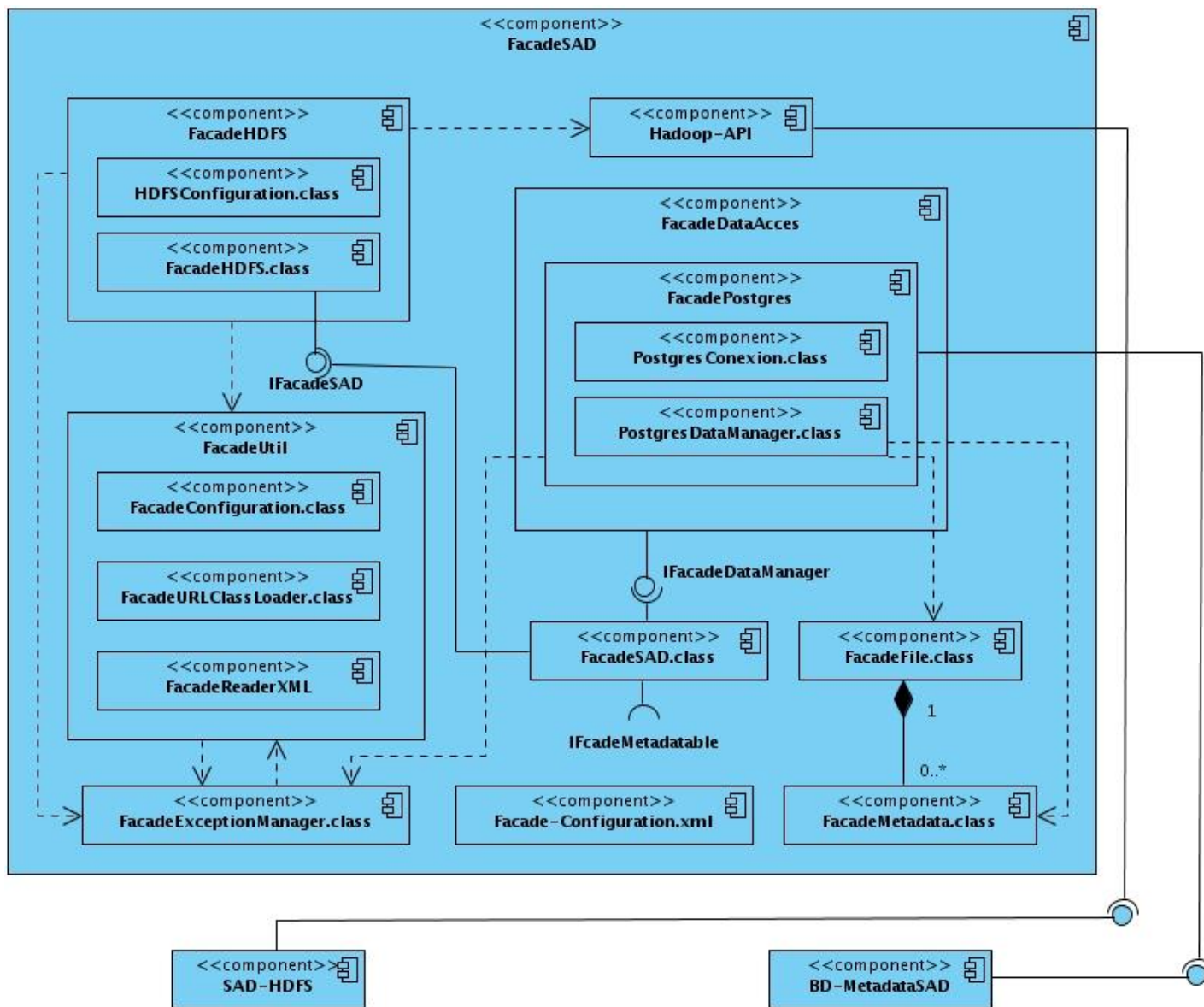


Figura 20: Diagrama de Componentes: Componente FacadeSAD.

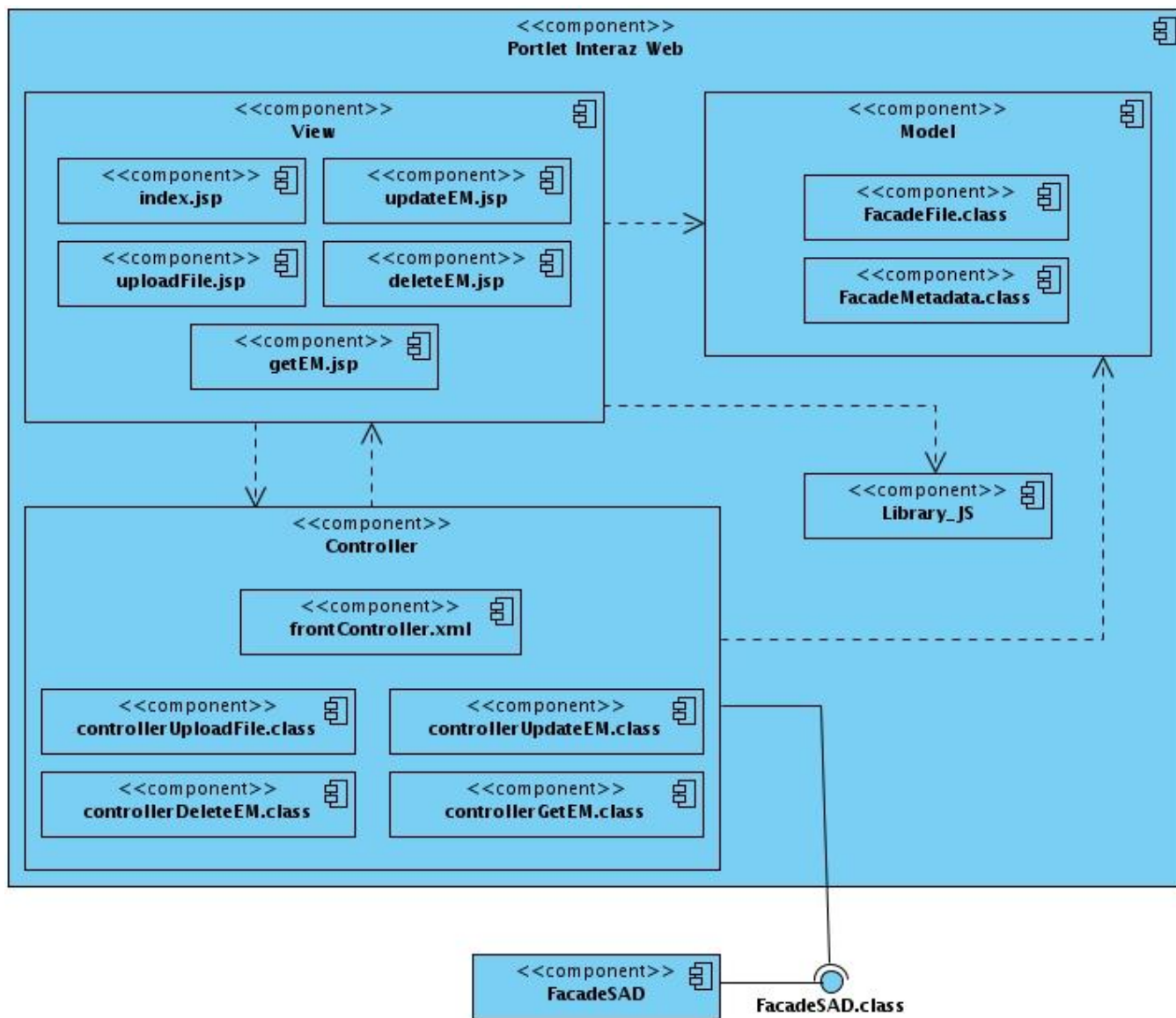


Figura 21: Diagrama de Componentes: Componente Portlet Interfaz Web

1.2. Diagrama de despliegue.

Los diagramas de despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre éstos. Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene algo de memoria y, a

menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representan generalmente un procesador o un dispositivo sobre el que se pueden desplegar los componentes (12).

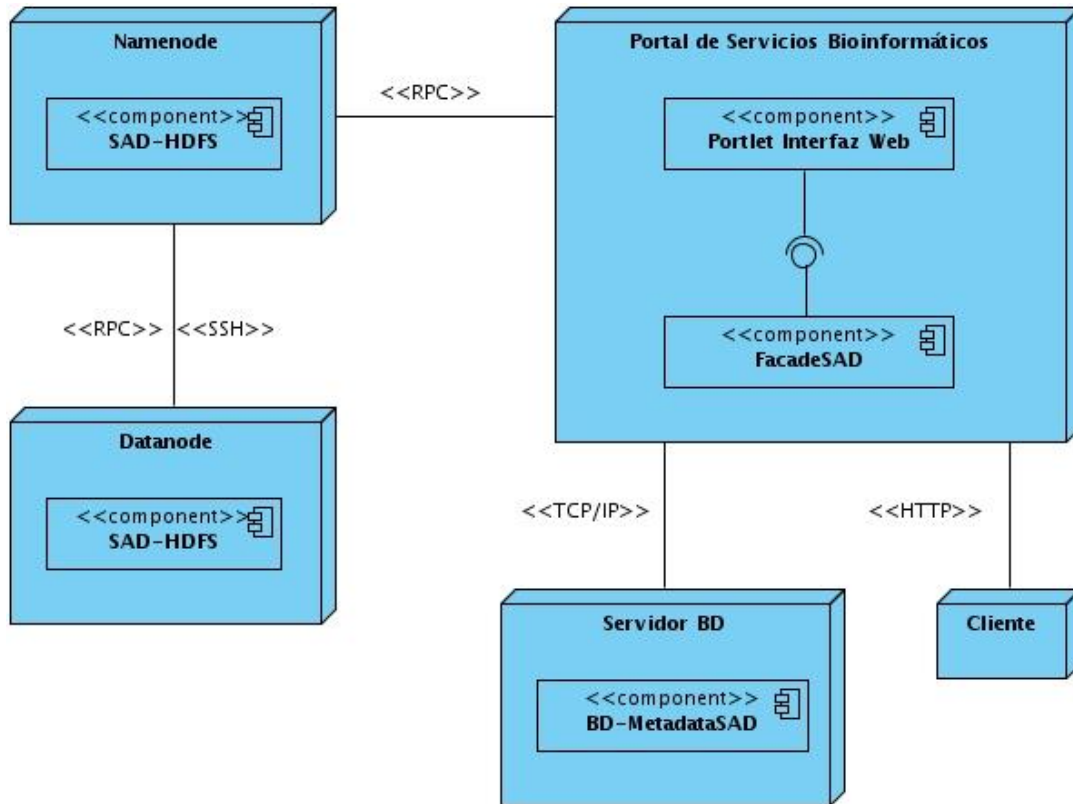


Figura 22: Diagrama de Despliegue del Sistema.

2. Integración del Sistema de Archivos Distribuidos con la interfaz Web.

Una vez implementado el subsistema FacadeSAD, la integración entre el Sistema de Archivos Distribuidos Hadoop y la interfaz Web se realizó de una manera muy intuitiva. Cumpliéndose con buenas prácticas de la ingeniería de software, se garantizó que si en el futuro fuese necesario sustituir Hadoop por otro Sistema de Archivos Distribuidos, la interfaz Web no sufriera ningún cambio. A continuación se muestran imágenes del código del subsistema Interfaz Web donde éste solicita servicios del Sistema de Archivos Distribuidos a través del subsistema FacadeSAD.

```

public ModelAndView handleRenderRequest(RenderRequest rr, RenderResponse rrl) throws Exception {
    LinkedList<LinkedList<FacadeMetadata>> mylist = FacadeSAD.getMetadatasFacade();
    LinkedList<String> listaEnviar=new LinkedList<String>();
    while(!mylist.isEmpty()){
        LinkedList<FacadeMetadata> aux=mylist.removeFirst();
        while(!aux.isEmpty()){
            FacadeMetadata facaAux= aux.removeFirst();
            listaEnviar.add(facaAux.GetKey());
            listaEnviar.add(facaAux.GetValue());
        }
    }
}

```

Figura 23: Código de integración entre Interfaz Web y FacadeSAD.

```

if (archivo != null && archivo.getArchivo() != null) {
    archivo.saveFile();
    String nombreArchivo="" +archivo.getArchivo().getOriginalFilename();
    LinkedList<FacadeMetadata>listMetadatas = new LinkedList<FacadeMetadata>();
    listMetadatas.add(new FacadeMetadata("Nombre", request.getParameter("nombre")));
    listMetadatas.add(new FacadeMetadata("Fecha", request.getParameter("Fecha")));
    listMetadatas.add(new FacadeMetadata("Descripción", request.getParameter("descripcion")));
    FacadeSAD.SaveOnSAD(nombreArchivo,listMetadatas,true);
}

```

Figura 24: Código de integración entre Interfaz Web y FacadeSAD.

3. Validación de la integración.

La calidad del software es el grado con el que un sistema, componente o proceso cumple los requerimientos especificados y las necesidades o expectativas del cliente o usuario.

Las pruebas de software permiten verificar y revelar la calidad de un producto de software. Éstas son una fase en el proceso de desarrollo de software y se realizan con el objetivo de comprobar que el software funcione como fue diseñado y verificar el grado de cumplimiento respecto a las especificaciones iniciales del software (13).

Nivel de Prueba

Las pruebas tienen diferentes niveles donde cada nivel especifica la técnica a utilizar y los atributos de calidad que se desean verificar. Para comprobar el correcto funcionamiento del sistema propuesto en esta investigación se aplicaron Pruebas de Desarrollador.

Técnica de Prueba

Existen varias técnicas para realizar las pruebas de software. Después de realizar el estudio de las técnicas se decidió aplicar como técnica de prueba las Pruebas de Funcionalidad, estas pruebas se realizan para validar las funciones, métodos, servicios y casos de uso. Analizando cada funcionalidad implementada para verificar que se cumplan todos los requisitos.

Tipo de Prueba

Se realizaron las pruebas de tipo Funcionales, cuyo objetivo es verificar el correcto cumplimiento de los requisitos funcionales, incluyendo la navegación, la entrada de datos, el procesamiento y la obtención de resultados.

Método de Prueba

El método de prueba utilizado fue el de Caja Negra, donde se realizaron pruebas sobre la interfaces del sistema, entendiendo por interfaz las entradas y salidas del mismo. Para emplear este método no es necesario conocer la lógica del programa, únicamente la funcionalidad que debe realizar. Se ejecuta cada caso de uso, flujo de caso de uso, o función, usando datos válidos e inválidos, para verificar lo siguiente:

- Que se aplique apropiadamente cada regla de negocio.
- Que los resultados esperados ocurran cuando se usen datos válidos.
- Que sean desplegados los mensajes apropiados de error y precaución cuando se usan datos inválidos.
- Funcionalidades incorrectas o ausentes.

- Errores de interfaz.

3.1. Casos de Prueba

Las pruebas de Caja Negra se aplicaron mediante los Casos de Prueba, que constituyen un conjunto de entradas, condiciones de ejecución y resultados esperados, desarrollados para cumplir un objetivo en particular o una función esperada.

3.1.1. Escenario: Guardar Estructura Molecular.

Clases válidas	Resultado esperado	Resultado de la prueba
Se selecciona la opción "Subir estructura molecular" y se introducen los siguientes datos: Nombre: "Genoma Humano", Fecha: "14/6/2011", Descripción: "Genoma humano modificado", Archivo: Genoma.rar.	El sistema verifica que se hayan introducido todos los datos, guarda el archivo seleccionado en el Sistema de Archivos Distribuidos, agrega a la base de datos el nombre, la fecha y la descripción.	Satisfactorio.
Se selecciona la opción "Subir estructura molecular" y se introducen los siguientes datos: Nombre: "Genoma Humano", Fecha: "", Descripción: "Genoma humano modificado", Archivo: Genoma.rar.	El sistema verifica si existen campos vacíos y muestra el siguiente mensaje de error: "Error: Existen campos vacíos."	Satisfactorio.

3.1.2. Escenario: Eliminar Estructura Molecular.

Clases válidas	Resultado esperado	Resultado de la prueba
Se selecciona la opción "Eliminar estructura molecular", se selecciona la estructura a	El sistema elimina la estructura molecular seleccionada del Sistema de Archivos Distribuidos, elimina de la base de datos los datos pertenecientes	Satisfactorio.

eliminar y se ejecuta la acción.	a esta estructura molecular, se actualiza la vista.	
Se selecciona la opción “Eliminar estructura molecular”, se selecciona la estructura a eliminar y se ejecuta la acción.	El sistema no puede eliminar la estructura molecular y muestra un mensaje de error: “Error inesperado, no se pudo eliminar la estructura molecular”.	Satisfactorio.

3.1.3. Escenario: Modificar Metadatos de Estructura Molecular.

Clases válidas	Resultado esperado	Resultado de la prueba
Se selecciona la opción “Modificar metadatos”, se seleccionan los metadatos a modificar, se modifican los datos deseados, se ejecuta la acción Actualizar.	El sistema muestra una vista donde se introducirán las modificaciones a realizar, actualiza los datos, actualiza la vista.	Satisfactorio.
Se selecciona la opción “Modificar metadatos”, se seleccionan los metadatos a modificar, se modifican los datos deseados, se ejecuta la acción Actualizar.	El sistema no puede modificar los metadatos de la estructura molecular y muestra un mensaje de error: “Error inesperado, no se pudieron modificar los metadatos”.	Satisfactorio.

4. Conclusiones del capítulo.

Durante el desarrollo de este capítulo se desplegó el Sistema de Archivos Distribuidos Hadoop. Se implementaron los subsistemas FacadeSAD e Interfaz Web, que permiten la gestión desde la interfaz Web de las estructuras moleculares almacenadas en el Sistema de Archivos Distribuidos. Y se validó mediante pruebas de tipo Funcionales la integración de Interfaz Web y Hadoop al Portal de Servicios Bioinformáticos.

CONCLUSIONES

- Se realizó un estudio de diferentes Sistemas de Archivos Distribuidos existentes y se determinó que el indicado para integrarlo al Portal de Servicios Bioinformáticos es el Sistema de Archivos Distribuidos Hadoop.
- Durante el análisis y diseño de la solución propuesta se identificaron seis requisitos funcionales agrupados en tres casos de usos del sistema y doce requisitos no funcionales.
- Se desplegó el Sistema de Archivos Distribuidos Hadoop logrando aumentar la capacidad de almacenamiento del Portal de Servicios Bioinformáticos.
- Se desarrolló una interfaz Web que permite gestionar los datos almacenados en el Sistema de Archivos Distribuidos desde el Portal de Servicios Bioinformáticos.
- Se implementó un subsistema que sirve de fachada de comunicación entre la interfaz Web y el Sistema de Archivos Distribuidos.
- Las pruebas realizadas arrojaron resultados positivos de la integración de la interfaz Web con el Sistema de Archivos Distribuidos al Portal de Servicios Bioinformáticos.

RECOMENDACIONES

En aras de darle continuidad al presente trabajo por su importancia en cuanto al almacenamiento de datos biológicos en particular y cualquier tipo de datos en general, se proponen las siguientes recomendaciones.

- Desplegar el Sistema de Archivos Distribuidos Hadoop en los laboratorios docentes de la Universidad de Ciencias Informáticas.
- Desarrollar nuevas funcionalidades en la interfaz Web y la fachada del sistema que permitan la gestión por usuarios de los datos almacenados.
- Integrar el Sistema de Archivos Distribuidos Hadoop con la plataforma de tareas distribuidas T-arenal para que los datos almacenados puedan ser procesados de forma distribuida.

REFERENCIAS BIBLIOGRÁFICAS

1. **Tanenbaum, A. S.** *Sistemas Operativos Distribuidos*. 2000.
2. **Richard Stevens, W.** *TCP/IP Illustrated : The Protocols*. 1994.
3. *A storage for mass small files*. **Yu, Lihua, y otros, y otros**. 2007.
4. *The Google Filesystem*. **Sanjay, Ghemawat, Howard, Gobio y Shun-Tak, Leung**. 2003.
5. *A parallel i/o middleware to integrate heterogeneous storage resources on grids. In Grid Computing*. **Pérez, J.M., y otros, y otros**. Berlín, Alemania : s.n., 2004.
6. **White, Tom.** *Hadoop: The Definitive Guide*. United States of America : s.n., 2009.
7. **Lussón, y otros, y otros.** *Propuesta de modelo de desarrollo para líneas de productos de software en el Centro de Tecnologías de Almacenamiento y Análisis de Datos*. 2010.
8. **Eclipse.** *OpenUp/Basic*. 2008.
9. UML. [En línea] [Citado el: 04 de diciembre de 2010.] <http://www.din.uem.br/>.
10. **Flanagan, David.** *Java en pocas palabras*. México : s.n., 1999. ISBN 1-56592-262-X.
11. Eva. [En línea] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_10/Conferencia_10/Materiales_complementarios/Introduccion_a_la_Disciplina_Analisis_y_Disenio.pdf.
12. Eva. [En línea] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.
13. *Taller de Calidad de Software: Introducción a la Calidad de Software*. **Hugo Vázquez, Roberto**.

BIBLIOGRAFÍA

1. **Tanenbaum, A. S.** *Sistemas Operativos Distribuidos*. 2000.
2. **Richard Stevens, W.** *TCP/IP Illustrated : The Protocols*. 1994.
3. *A storage for mass small files*. **Yu, Lihua, y otros, y otros**. 2007.
4. *The Google Filesystem*. **Sanjay, Ghemawat, Howard, Gobio y Shun-Tak, Leung**. 2003.
5. *A parallel i/o middleware to integrate heterogeneous storage resources on grids. In Grid Computing*. **Pérez, J.M., y otros, y otros**. Berlín, Alemania : s.n., 2004.
6. **White, Tom.** *Hadoop: The Definitive Guide*. United States of America : s.n., 2009.
7. **Lussón, y otros, y otros.** *Propuesta de modelo de desarrollo para líneas de productos de software en el Centro de Tecnologías de Almacenamiento y Análisis de Datos*. 2010.
8. **Eclipse.** *OpenUp/Basic*. 2008.
9. UML. [En línea] [Citado el: 04 de diciembre de 2010.] <http://www.din.uem.br/>.
10. **Flanagan, David.** *Java en pocas palabras*. México : s.n., 1999. ISBN 1-56592-262-X.
11. Eva. [En línea] http://eva.uci.cu/file.php/102/Curso_2010-2011/Clases/Semana_10/Conferencia_10/Materiales_complementarios/Introduccion_a_la_Disciplina_Analisis_y_Disenio.pdf.
12. Eva. [En línea] http://eva.uci.cu/file.php/259/Curso_2010-2011/Semana_8/Conferencia_6/Materiales_Basicos/Implementacion.pdf.
13. *Taller de Calidad de Software: Introducción a la Calidad de Software*. **Hugo Vázquez, Roberto**.
14. **Visconti, Marcello y Astudillo, Hernán.** *Fundamentos de Ingeniería de Software*. Santa María : s.n.
15. **Garlan, David y Shaw, Mary.** *An introduction to software architecture*. New Jersey : V.Ambriola and G.Tortora, World Scientific.
16. **Gamma, Erich, y otros, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software*. 1994.

17. IngenierosSoftware. [En línea] [Citado el: 06 de diciembre de 2010.]
<http://www.ingenierossoftware.com/analisisydiseno/patrones-diseno.php>.
18. Epidata Consulting. [En línea] [Citado el: 27 de noviembre de 2010.]
http://www.epidataconsulting.com/tikiwiki/tiki-read_article.php?articleId=15#Conociendo_UML_2.0.
19. **Pressman, Roger.** *Ingeniería del Software. Un enfoque práctico.* 2002.
20. **SpringSource.** SpringSource Community. [En línea] 2011. <http://www.springsource.org/>.
21. **SpringHispano.** SpringHispano.org. [En línea] 2011. <http://www.springhispano.org/>.
22. ProgramacionJ2ee. [En línea] 2011. <http://www.programacionj2ee.com/>.

GLOSARIO DE TÉRMINOS

API: Interfaz de Programación de Aplicaciones (API, por sus siglas en inglés) es el conjunto de funciones que ofrece una biblioteca para ser utilizada por otro software.

Cluster: Término que se aplica a los conjuntos de componentes de hardware comunes que se comportan como si fuesen una única computadora.

Namenode: servidor que maneja el espacio de nombres del sistema de archivos y regula el acceso de los usuarios a estos en HDFS.

Datanode: estaciones de trabajo, en los que se almacenan los datos en HDFS.

write-once, read-many-times: en los sistemas de archivos es cuando solo es permitido escribir en un archivo por un proceso y leerse por varios paralelamente.

Proxy: componente que sirve de intermediario entre otros dos o más componentes.

Framework: estructura conceptual y tecnológica de soporte definida, normalmente con artefactos o módulos de software concretos, con base en la cual otro proyecto de software puede ser organizado y desarrollado.

Ruby: lenguaje de programación interpretado y orientado a objetos, creado por el programador japonés Yukihiro Matsumoto.

Rails: framework de aplicaciones Web de código abierto escrito en el lenguaje de programación Ruby.


LDAP: Protocolo Ligero de Acceso a Directorios (LDAP, por sus siglas en inglés) a nivel de aplicación que permite el acceso a un servicio de directorio ordenado y distribuido para buscar diversa información en un entorno de red.

EJB: API que forma parte del estándar de construcción de aplicaciones empresariales J2EE (ahora JEE

5.0) de la Corporación Oracle.

Singletons: patrón de diseño con el fin de restringir la creación de objetos pertenecientes a una clase.

ANEXOS



The image shows a web interface window titled "Guardar Estructura Molecular". It contains the following elements:

- A label "Nombre:" followed by a text input field.
- A label "Fecha:" followed by a date selection field with a calendar icon.
- A text input field containing "Seleccione un archivo" and a "Buscar" button.
- A large empty rectangular area below the file selection field.
- "Aceptar" and "Cancelar" buttons at the bottom of the window.

Figura 25: Interfaz Web - Escenario Subir Estructura Molecular.

Estructuras Moleculares			
Buscar <input type="text"/> < >			
#	Nombre	Fecha	Descripción
1	Genoma Humno	12/05/2011	Estructura molecular del Genoma Humano
2	Banco de preteinas	15/05/2011	Proteinas modificadas
3	Banco de preteinas	16/05/2011	Proteinas
4	Lípidos	17/05/2011	Muestra utilizada en ensayos clinicos
5	Estructura Molecular	18/05/2011	Estructura molecular
6	Genoma Humno	19/05/2011	Estructura molecular sometida a pruebas
8	Genoma del mosquito	12/06/2011	Estructura molecular Genoma-Mosquito
14	Genoma Humno	25/07/2011	Estructura molecular del Genoma Humano
No se encontraron datos			

Figura 26: Interfaz Web - Escenario Mostrar Metadatos de Estructuras Moleculares