Universidad de las Ciencias Informáticas
Facultad 6

Título: Solución para el cálculo distribuido con T-arenal de la metodología MMH a través de la Grid.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Antonio Pascual Rivera Matos

Gustavo Rafael Viamonte Veloz

Tutores: Ing. Yuneimy Tellez Pérez

Ing. Roberto Tellez Ibarra

Junio 2010

“La inteligencia humana es un monstruo, cuando no crea, devora.”

José Martí

DECLARACIÓN DE AUTORÍA

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

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

Antonio Pascual Rivera Matos

_____
Firma del Autor

Yuneimy Tellez Pérez

_____
Firma del Tutor

Gustavo Rafael Viamonte Veloz

_____
Firma del Autor

Roberto Tellez Ibarra

_____
Firma del Tutor

DATOS DE CONTACTO

Autores:
Antonio Pascual Rivera Matos.
Universidad de las Ciencias Informáticas.
e-mail: aprivera@estudiantes.uci.cu

Gustavo Rafael Viamontes Veloz.
Universidad de las Ciencias Informáticas.
e-mail: grviamontes@estudiantes.uci.cu

Tutores:
Yuneimy Tellez Pérez.
Ingeniero en Ciencias Informáticas.
Instructor recién graduado.
e-mail: ytellez@uci.cu

Roberto Tellez Ibarra.
Ingeniero en Ciencias Informáticas.
Instructor recién graduado.
e-mail: rtibarra@uci.cu

AGRADECIMIENTOS

A la Revolución y a Fidel por habernos dado la oportunidad de estudiar en esta Universidad de excelencia.

A nuestros padres y demás familiares por apoyarnos siempre y guiarnos, para poder llegar al lugar que hoy estamos.

A todos nuestros amigos que han estado con nosotros a lo largo de la carrera, en los buenos y malos momentos.

A los profesores por la dedicación y por enseñarnos algo nuevo cada día.

A todos aquellos que colaboraron de una forma u otra para la realización exitosa del presente trabajo.

A nuestros tutores Roberto y Yuneimy por su preocupación y apoyo incondicional.

Al tribunal y al oponente por contribuir con sus críticas y por las acertadas recomendaciones para mejorar la calidad del trabajo.

A todos muchas gracias...

DEDICATORIA

De Pascual

Primeramente quiero agradecer a los dioses de todas las religiones, ya que este sueño hecho realidad constituye sin lugar a dudas un milagro. A la luz de mi vida, la que siempre me ilumina incluso cuando todas las demás luces se apagan: mi madre, por luchar tanto por mí, por regalarme su corazón y por hacer de mí un hombre de bien ante cualquier circunstancia. A mi padre por enseñarme a no perder el tiempo, y que la modestia se debe mostrar por fuera y no en el interior. A toda mi familia, por preocuparse en todo momento. A mis abuelas por toda la ternura derrochada. A mis abuelos por esas primeras lecciones de la vida, y como conducirme por el camino correcto. A mis hermanas por confiar siempre en mí. A mis tías Aleidita, Zulema y Milagros, por enseñarme que lo esencial es invisible a los ojos. A mi tío Alfredo, que por sus consejos y apoyo ha sido como un segundo padre para mí. Discúlpenme aquellos que no mencioné aquí, es que se haría interminable esta sección.

Quisiera agradecer también a todos los muchachos que compartieron conmigo un poco más de cinco maravillosos años. A mis compañeros de lucha: Ángel, Edilberto, Bola, Enrique, Jesús, Carlos Javier, Gustavo, Alex, Dayana, Mayelín, Juan Carlos y otros que no llegaron al final, pero que la experiencia fue única, nunca los olvidaré. A mis camaradas del equipo de baloncesto. Y sobre todo a las musas de la inspiración de todo momento.

Agradecer también a mis tutores por su guía profesional. A Adolfo e Idelsis por comportarse como verdaderos ángeles de la guarda. A todos los maravillosos profesores por el asesoramiento en todos los sentidos, este logro también es de ustedes.

Finalmente agradecer a los oponentes de mi vida profesional y de manera general, porque me hicieron forjarme de la manera más efectiva, gracias por ayudarme a demostrar que los cometas solo alcanzan su mayor altura contra el viento y nunca a favor de este.

De Gustavo

Me resulta muy difícil intentar resumir en nombres de personas lo agradecido que me siento en estos momentos. Quizás esta sección no sea suficiente para mencionar a todos aquellos que han dedicado un poquito de su tiempo y me han prestado su ayuda desinteresada para poder cumplir con este, mi sueño. Pido las más sinceras disculpas si alguien se siente excluido en mis palabras y les puedo asegurar que siempre tendrá mi gratitud.

Quiero agradecer a mi madre por preocuparse constantemente y por apoyarme siempre en cada paso que doy. A mi padre por confiar en mí y ayudarme a ser mejor persona. A los dos por constituir mis ejemplos a seguir en la vida.

A mis abuelos por el impulso que me han dado para seguir adelante.

A mis tíos y primos de Ciudad de la Habana por acogerme en sus casas y portarse tan bien conmigo.

A Rodolfo (Fito) por su colaboración y ayuda desinteresada.

A mis hermanos de siempre: Omar, René, Angel, Wilfredo, Jose, Ochoa y Wilder por la amistad que nos une y que espero no termine nunca.

A mis buenos amigos de la facultad, los que están hoy aquí y los que desgraciadamente no pudieron terminar con nosotros este largo recorrido de cinco años, quiero decirles que nunca los olvidaré: Dresky, Carlos, Juan Carlos, Pascual, Angel, Frank, Rosnel, Renier, Javier, Joan, Angel, Raidel, Luis, Dairon, Alex, Dayana y muchos más, gracias a todos por formar parte de mi vida.

A los tutores y al tribunal por ser tan flexibles y ayudarnos en todo lo que pudieron.

A todo aquel que después de un apretón de manos, preguntó ¿Y la tesis qué? gracias.

RESUMEN

En la presente investigación se reutilizará la implementación de una alternativa distribuida de la metodología de Hipersuperficies de Múltiples Mínimos (MMH), sobre la Plataforma de Tareas Distribuidas (T-arenal) realizada en la Universidad de las Ciencias Informáticas (UCI) con anterioridad; dicho procedimiento es de gran importancia en la Bioinformática. No obstante, y a pesar de realizar los cálculos distribuidos sobre T-arenal la mencionada aplicación solo puede ser ejecutada desde la universidad, siendo imposible para especialistas foráneos el uso de la aplicación.

El actual trabajo presenta una solución que permite a los especialistas acceder a la aplicación a través de la Web y utilizar el poder computacional con el que cuenta la UCI, mediante la creación de un portal de servicios en Grid, ya que para la realización de los cálculos distribuidos de la metodología MMH es necesario el recurso de varias computadoras. La fundamentación teórica, las características del sistema a desarrollar, el diseño de la solución y su implementación, son tratados, en ese orden, en los cuatro capítulos que estructuran la tesis.

PALABRAS CLAVE: MMH, T-arenal, portal de servicios, Grid, cálculos distribuidos.

ÍNDICE GENERAL

AGRADECIMIENTOS I

DEDICATORIA II

RESUMEN IV

INTRODUCCIÓN 1

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA 5

1.1. Surgimiento de los Sistemas Distribuidos y la Computación Grid. 5
1.2. Definiciones de Sistemas Distribuidos y Computación Grid. Características. 5
1.2.1. Sistemas Distribuidos. 5
1.2.2. Características esenciales de los Sistemas Distribuidos: 6
1.2.3. Ventajas de los Sistemas Distribuidos: 9
1.2.4. Desventajas de los Sistemas Distribuidos. 9
1.2.5. Características de la Computación Grid. Aplicaciones. 10
1.3. Aplicación de los Sistemas Distribuidos en la Bioinformática. Metodología MMH. 12
1.4. Herramientas y Tecnologías. 13
1.4.1. Metodología de Desarrollo de Software. 14
1.4.2. Lenguaje de Modelado. 16
1.4.3. Herramienta Case. 16
1.4.4. IDE y Lenguaje de Programación. 16
1.4.5. Herramientas y Tecnologías para el desarrollo. 18
1.5. Conclusiones 22
CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA 24

2.1 Breve descripción del Sistema. 24
2.2. Modelo de Dominio. 24
2.3. Especificación de los requisitos del sistema. 26
2.3.1. Requisitos funcionales. 26
2.3.2. Requisitos no funcionales. 26
2.4. Actores y casos de uso del sistema. 27
2.4.1. Actores del sistema. 27
2.4.2. Casos de Uso del Sistema. 28
2.4.3. Diagrama de Casos de Uso del Sistema. 29
2.4.4 Patrón de Casos de Uso utilizado 29
2.5 Conclusiones 30
CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA 31

3.1. Estilo de arquitectura. 31
3.2. Patrón de Arquitectura. 32
3.3. Patrones de diseño. 33
3.2.1. Patrón GRASP. 34
3.3.2. Inyección de Dependencia (ID). 35

3.4. Diagramas de clases de diseño....35
3.5. Diagramas de interacción....38
3.6. Vista de Despliegue. 40
3.6.1 Diagrama de despliegue....40
3.7. Conclusiones....41
CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS....42

4.1. Diagrama de Componentes....42
4.1.1. Diagrama de despliegue de componentes....47
4.2. Pantallas de las aplicaciones. Resultados....47
4.3. Modelo de Pruebas. 51
4.3.1. Casos de pruebas. 52
4.4. Conclusiones....54
CONCLUSIONES....55

RECOMENDACIONES....56

BIBLIOGRAFÍA....57

ANEXOS....59

GLOSARIO DE TÉRMINOS....69

INTRODUCCIÓN

El desarrollo continuo de la ciencia, en la que se produce una ampliación creciente de las especializaciones que aborda, accionada además por una tecnología progresivamente poderosa y conducida por el incremento de complejos problemas a resolver, le impone a la actividad científica el uso, a la par que el desarrollo, de la computación. Ciencia muy importante y necesaria para el análisis de datos y la colaboración tanto de los esfuerzos individuales, como de comunidades de investigadores.

No obstante a que el poder computacional, el almacenamiento de datos y las telecomunicaciones en general continúan mejorando exponencialmente, los recursos computacionales son insuficientes para las demandas de los científicos en la solución de problemas investigativos complejos.

Una computadora personal en el 2001 era tan rápida como una supercomputadora en 1990, pero en ese tiempo los biólogos eran felices de computar una simple estructura molecular. Hoy en día precisan calcular complejas estructuras de macromoléculas, y analizar millones de pruebas para nuevas sustancias, lo que le plantea una tarea inmediata a la computación en el orden de avanzar en la solución de problemas similares al ejemplo anterior. Ofreciendo recursos en este sentido, surgen, entre otros, los sistemas distribuidos, por la posibilidad de integrar el esfuerzo de investigadores y máquinas disímiles en la realización de complejas tareas.

Los sistemas distribuidos tienen sus orígenes alrededor del principio de los 70', simultáneamente con el desarrollo de las redes locales de alta velocidad, en sus inicios son conocidos el de Information Power Grid (IPG), creado para lograr la integración y gestión de recursos de los centros de la NASA; el proyecto SETI@Home a nivel mundial, dedicado a la investigación de vida extra-terrestre o búsqueda de vida inteligente en el espacio; ellos pueden ser considerados como pioneros de esta tecnología, que al tiempo que sirvieron y sirven para la colaboración en el tratamiento de grandes volúmenes de datos y cálculos, evolucionaron de acuerdo a la necesidad de la finalidad para el cual fueron diseñados. En este escenario en evolución y como resultado del mismo, se producen las condiciones para la aparición, en la actualidad de los sistemas en Grid.

Para muchos, el término "Grid" es una forma potencial para superar dichos obstáculos. Construida en Internet y escala mundial, es una nueva clase de infraestructura que provee mecanismos dimensionales, seguros y muy efectivos para el acceso y descubrimiento de servicios y recursos remotos. Posibilita que las diferentes colaboraciones científicas puedan compartir recursos a una escala sin precedente, pues los

grupos antes lejanos geográficamente pueden ahora trabajar mancomunadamente, compartiendo y complementando tareas en función de un problema de investigación complejo.

Las ventajas incuestionables de los sistemas distribuidos en Grid, constituyen un campo de significativa importancia para el propósito de lograr avances sustanciales en el desarrollo científico-técnico, para la solución de problemas medulares en áreas tan disímiles del quehacer investigativo como son: la Biotecnología, la Neurociencia, los estudios de Meteorología, Geología, los servicios educativos, entre otros.

En las condiciones concretas de ambiente científico-técnico de nuestro país, se produce actualmente una peculiaridad en relación a lo planteado anteriormente. Se considera que se ha alcanzado un nivel de desarrollo de la ciencia e informático adecuado, existe una infraestructura y los insumos mínimos que permiten implementar sistemas en Grid, que faciliten la actividad científica en diversas áreas y por otro lado, aún no se ha implementado una computación de este tipo, de lo que se infiere el valor de un esfuerzo en este sentido.

De acuerdo a lo anterior, se reutilizará una implementación distribuida de la metodología de Hipersuperficies de Múltiples Mínimos (MMH), proceder de significativo valor en la Bioinformática, pues facilita calcular, a partir de un número de agregados moleculares en un espacio multidimensional dado, las energías de asociación estadísticas y la(s) estructura(s) más favorecida(s) energéticamente en dicho espacio.

Esta metodología, por su complejidad, demanda el uso del recurso de varias computadoras conectadas de tal modo, que facilite, a partir del cálculo de cada una, su posterior integración de los resultados, con la finalidad de obtener datos válidos y fiables sobre las energías de asociación estadísticas y las estructura(s) más favorecida(s). Para la obtención de los resultados se necesita realizar una gran cantidad de cálculos. Por otro lado, se tiene que la Universidad de las Ciencias Informáticas (UCI), cuenta con un clúster compuesto por ocho nodos, además de más de 7000 computadoras localizadas en la red universitaria.

Específicamente, en la facultad 6 de la UCI existe una Plataforma de Tareas Distribuidas sobre la cual se implementó una alternativa de aplicación para la metodología MMH, la cual solo permite realizar los cálculos para dicha metodología utilizando la red universitaria, siendo imposible para especialistas de proyectos fuera de la universidad acceder a la aplicación. Dichos especialistas trabajan en centros del

Polo Científico de Cuba entre los que se encuentran: el Centro de Inmunología Molecular, Centro de Ingeniería Genética y Biotecnología, Centro de Química Farmacéutica y la Facultad de Química de la Universidad de La Habana, muy alejados geográficamente, pero que mediante la red, podrían utilizar los recursos computacionales con los que cuenta la universidad. Por lo cual se considera la posibilidad de implementar una aplicación en Grid para acceder a los recursos de la Metodología MMH.

Lo anterior impone entonces el estudio de las posibilidades de implementación de sistemas distribuidos en Grid, en las condiciones reales de la actividad investigativa del entorno, de esta necesidad, condicionada por la formación de especialista en este perfil, es que surge el siguiente problema de investigación.

Problema Científico: ¿Como compartir el recurso T-arenal a través de una Grid, para realizar cálculos basados en la Metodología MMH?
Objeto de estudio: Asignación del recurso T-arenal a través de la Grid.
Campo de acción: Sistemas Distribuidos Grid.
Objetivo general: Desarrollar una solución para el cálculo distribuido con T-arenal de la metodología MMH a través de la Grid.

Objetivos específicos:

- ✓ Realizar el levantamiento de Requisitos.
- ✓ Diseñar el sistema propuesto.
- ✓ Implementar el sistema propuesto.
- ✓ Realizar pruebas exploratorias a la aplicación implementada.

Tareas:

- ✓ Realizar un estudio del estado del arte de la computación Grid y de la Metodología MMH para la elaboración de los fundamentos teóricos-metodológicos.
- ✓ Describir los requisitos funcionales y no funcionales del sistema.
- ✓ Diseñar los casos de uso del sistema.
- ✓ Realizar los casos de uso del diseño para el desarrollo de la arquitectura.
- ✓ Implementar una solución para el cálculo distribuido con T-arenal de la metodología MMH a través de la Grid.
- ✓ Validar la solución desarrollada a partir de pruebas de caja negra.

Importancia

Se ofrece un Sistema Grid para la implementación distribuida de la Metodología MMH, que constituye una vía nueva en las condiciones de desarrollo, la cual permitirá avances sustanciales en el acceso a los recursos englobados en el sistema. Además de acelerar los complejos cálculos que se realizan de esta metodología en las condiciones concretas de Cuba, en las diferentes instituciones investigativas del sistema nacional.

Estructura

La tesis está estructurada en introducción, cuatro capítulos, conclusiones, recomendaciones, bibliografía y anexos.

Capítulo 1. Fundamentación Teórica: en este capítulo se presentan los referentes históricos y las posiciones teóricas en torno a los Sistemas Distribuidos. Se realiza un análisis de las características y estructura de los Sistemas Grid, de la Metodología MMH y su relación para compartir recursos de esta última, y se proponen los Materiales y Métodos utilizados para elaborar el sistema. Se ofrecen además las características, estructura lógica-metodológica, así como los Lenguajes y Herramientas de Modelado y Programación.

Capítulo 2. Características del sistema: en este capítulo se describen las principales características del sistema a desarrollar, sus requisitos funcionales y no funcionales, y los actores que intervienen en el mismo. Se representa también el diagrama de casos de uso del sistema, así como una breve descripción de los casos de usos identificados.

Capítulo 3. Diseño del sistema: en este capítulo se dará una descripción del estilo arquitectónico utilizado para el desarrollo del sistema, se describirán los patrones de diseño empleados y los diagramas de clases de diseño e interacción de los principales casos de uso. Se mostrará además el diagrama de despliegue del sistema.

Capítulo 4. Implementación y pruebas al sistema: en este capítulo se describe la implementación del sistema en términos de componentes, y la manera en que estos componentes serán desplegados. Se ilustrarán los principales resultados obtenidos. Además, los casos de pruebas para los principales casos de uso.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Estudios preliminares.

1.1. Surgimiento de los Sistemas Distribuidos y la Computación Grid.

La idea de compartir recursos tecnológicos para la solución de determinados problemas no es un concepto nuevo; ya desde 1965 Fernando Corbato del Instituto Tecnológico de Massachusetts (MIT), y otros diseñadores del sistema operativo Multics, imaginaron la posibilidad del funcionamiento de la computadora *"como una planta hidráulica o como una compañía de electricidad"*. En 1968 J. C. R. Licklider y Robert W. Taylor en su artículo "La Computadora como un Dispositivo de Comunicaciones" anticiparon el surgimiento de los sistemas en escenarios Grid. A finales de los 60' se comenzaron a desarrollar trabajos relacionados con sistemas distribuidos con resultados favorables. [1]

El punto de partida del desarrollo de los Sistemas Distribuidos tuvo lugar al mismo tiempo que se crearon las redes locales de alta velocidad a inicios de 1970, o a partir de ese mismo hecho y el incremento de su uso y eficiencia, el cual ha sido proporcional a las mejoras de hardware y software del mundo informático.

Hoy en día, la disponibilidad de computadoras personales de alto rendimiento, ordenadores servidores u otras estaciones de trabajo se ha orientado mayoritariamente hacia los sistemas distribuidos en detrimento de los ordenadores centralizados multiusuario. La tendencia se ha acelerado a partir del desarrollo de hardware y software para sistemas distribuidos, diseñados para las aplicaciones distribuidas, de manera que los ordenadores puedan coordinar sus actividades y compartir los recursos del sistema.

1.2. Definiciones de Sistemas Distribuidos y Computación Grid. Características.

1.2.1. Sistemas Distribuidos.

Existe un consenso entre diversos autores para definir como Sistema Distribuido (**SD**) a:

"Una colección de computadores autónomos interconectados por una red, que cuentan con el adecuado software distribuido para que el sistema sea visto como una sola entidad para los usuarios que necesiten alguna facilidad o funcionalidad de procesamiento de los recursos remotos del sistema". [2]

“Sistemas cuyos componentes hardware y software, que están en ordenadores conectados en red, se comunican y coordinan sus acciones mediante el paso de mensajes, para el logro de un objetivo. Se establece la comunicación mediante un protocolo prefijado por un esquema cliente-servidor”. [3]

La anterior definición revela la esencia del concepto y por lo cual es aplicable al análisis de un sistema para determinar si posee esta característica, lo que determina que sea asumirlo como un elemento de partida de la plataforma teórica en la presente tesis.

Una de las ventajas significativas de los **SD** es que estos pueden ser implementados en varias plataformas de hardware: podrían ser unas pocas estaciones de trabajo conectadas a una red local, o hasta una colección de redes de área extensa interconectadas que cuentan con miles de ordenadores. Tal opción abre infinitas posibilidades de utilización y aplicación para las más disímiles actividades humanas, entre ellas, el desarrollo investigativo en aras de la solución de complejos problemas que, en el contexto del uso unitario y solitario de la computación, requeriría enormes recursos humanos y tecnológicos y retardaría considerablemente dichas soluciones.

1.2.2. Características esenciales de los Sistemas Distribuidos:

La compartición de recursos es una de las características más importantes de los **SD**, ya que posibilita que recursos heterogéneos logren integrarse en pos de resolver un problema común.

Compartición de Recursos: A pesar de que el término “recurso” es bastante abstracto, es el que mejor concreta la colección de entidades que pueden ser compartidas en un sistema distribuido. Tal colección de entidades podría ser ficheros, bases de datos, ventanas, aplicaciones y otros objetos de datos.

Para que sea posible compartir dichos recursos, el sistema debe ser manejado por un gestor que ofrezca una interfaz de comunicación al usuario, permitiéndole a este acceder de una manera consistente y fiable.

Un **SD** puede verse de manera abstracta como una colección de gestores de recursos y una serie de programas que utilizan dichos recursos. Dicho de esta manera se llega a dos modelos de Sistemas Distribuidos: el modelo Cliente – Servidor y el modelo basado en objetos.

El modelo basado en objetos en un **SD** se refiere a los objetos distribuidos, los cuales se presentan al usuario como algo muy familiar, no como computadoras, redes o lenguajes de programación; son actores o sustitutos del mundo real, o representación de algunos conceptos. Este modelo permite distribuir objetos a través de una red heterogénea, permitiendo a cada uno de sus componentes interactuar como una sola

unidad. Provee además de una infraestructura para abastecer los componentes de servicios disponibles y así reunir los procesos cooperativos distribuidos de una forma universal y transparente.

El modelo Cliente – Servidor de un sistema distribuido es el modelo más conocido y ampliamente adoptado en la actualidad. En este modelo existen un conjunto de procesos servidores, cada uno actuando como un gestor de recursos de un tipo, y una colección de procesos clientes, cada uno llevando a cabo una tarea que requiere el acceso a algunos recursos de hardware y software compartidos. Los gestores de recursos a su vez podrían necesitar acceder a recursos compartidos manejados por otros procesos, de manera que algunos procesos son ambos clientes y servidores. En el modelo Cliente – Servidor, todos los recursos compartidos son mantenidos y manejados por los procesos servidores. Los procesos clientes realizan peticiones a los servidores cuando necesitan acceder a algún recurso. Si la petición es válida, entonces el servidor lleva a cabo la acción requerida y envía una respuesta al proceso cliente.

Para el diseño e implementación de la presente tesis se decide la utilización del modelo Cliente – Servidor por ser el más aplicable a las condiciones concretas del entorno.

Otra de las características importantes y necesarias de los **SD** es la concurrencia, la cual resulta ser hasta cierto punto la medida para conocer el poder computacional con el que cuenta cualquier sistema distribuido.

Concurrencia: cuando existen varios procesos en una única computadora, se dice que se están ejecutando concurrentemente. Si la computadora cuenta con un único procesador central, la concurrencia ocurre entrelazando la ejecución de los distintos procesos. Si la computadora tiene N procesadores, entonces pueden ejecutarse estrictamente a la vez N procesos. En los **SD** pueden existir muchas computadoras, es decir que si hay M computadoras, hasta M x N procesos podrían ejecutarse en paralelo.

Si el **SD** está basado en el modelo de compartimentación de recursos, la posibilidad de ejecución paralela podría ocurrir por dos razones:

1. Múltiples usuarios interactúan simultáneamente con los programas de aplicación.
2. Múltiples procesos servidores se ejecutan de manera concurrente, cada uno respondiendo a diferentes peticiones de los usuarios.

La primera razón ocasiona menos problemas ya que las aplicaciones de interacción se ejecutan de forma aislada en la estación de trabajo del usuario de manera que no entran en conflicto con las aplicaciones de otros usuarios. La segunda razón surge debido a la existencia de uno o más procesos servidores para cada tipo de recurso. Las peticiones para acceder a los recursos de un servidor dado pueden ser encoladas y ser procesadas secuencialmente, o también ser procesadas de manera concurrente por múltiples instancias del gestor de recursos; cuando esto ocurre la sincronización de las acciones de los procesos servidores debe ser cuidadosamente planeada de manera que no se pierdan los beneficios de la concurrencia.

La tolerancia a fallos en cualquier sistema da la mesura exacta de cuán estable es el sistema. En los **SD** resulta muy importante esta característica porque existen numerosos recursos heterogéneos, y conocer la disponibilidad de tales recursos resulta vital.

Tolerancia a Fallos: cuando ocurren fallos en el hardware o el software, los programas podrían detenerse o producir resultados incorrectos. El diseño de sistemas tolerantes a fallos está basado en el uso de componentes redundantes (redundancia de software), y el diseño de programas capaces de recuperarse ante los fallos. Los **SD** cuentan con un alto grado de disponibilidad ante los fallos de hardware. La disponibilidad de un sistema es la proporción de tiempo en que este está disponible, en caso de que uno de los componentes del sistema falle, solo se ve afectado el área de ese componente, permitiendo al usuario moverse a otra estación de trabajo.

La transparencia es una característica esencial de los **SD**, ya que ejerce una gran influencia en el diseño de software del sistema.

Transparencia: está definida como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes de un **SD**, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes; de manera que ejerce una gran influencia en el diseño de software del sistema.[4].

El manual de referencia RM-ODP [ISO1996a] identifica ocho formas de transparencia:

1.- Transparencia de Acceso.
2.- Transparencia de Concurrencia.
3.- Transparencia de Replicación.
4.- Transparencia de Fallos.

5.- Transparencia de Prestaciones.

6.- Transparencia de Localización.

7.- Transparencia de Migración.

8.- Transparencia de Escalado.

Las más importantes son las de Localización y Acceso, en ocasiones se les hace referencia a ambas como Transparencias de Red, ya que su presencia o ausencia puede afectar seriamente el uso de los recursos distribuidos, por lo cual se considera muy importante tenerlas en cuenta a la hora de la implementación de la presente tesis.

1.2.3. Ventajas de los Sistemas Distribuidos:

Los **SD** cuentan con muchas características ventajosas. A continuación se explican las más importantes:

Velocidad: El uso de los **SD** posibilita lograr una mayor velocidad de cómputo que con una sola computadora sería imposible de lograr.

Independencia de fallo de los recursos: El fracaso o mal funcionamiento de algún recurso lógico o físico no implica que el sistema falle.

Distribución inherente: Un grupo de recursos ubicados en lugares diferentes trabajan en conjunto para realizar una tarea común.

Compartición de datos y dispositivos: Permite a muchos usuarios compartir periféricos sin importar el tipo que sean tales como: dispositivos de almacenamiento masivo de archivos, impresoras entre otros. Además de permitir el acceso de varios usuarios a datos comunes, como servidores de bases de datos, servidores de cómputo, servidores de realidad virtual, entre otros.

Flexibilidad y extensibilidad: Los **SD** posibilitan el crecimiento incremental y proporcionan la modificación o extensión, ya que se pueden adaptar a ambientes cambiantes sin tener que modificar sus operaciones.

1.2.4. Desventajas de los Sistemas Distribuidos.

Uno de los problemas de los **SD** tiene que ver con la red de comunicación, ya que existe la posibilidad de que se pierdan los mensajes. Con la finalidad de recuperarlos el sistema deberá tener programas especiales y la red puede sobrecargarse, cuando esto ocurre debe ser reemplazada o una segunda red

debe ser agregada. Una vez que el sistema dependa de la red, y si esta no posee un ancho de banda óptimo, su saturación puede negar muchas de las ventajas a lograr para lo cual el **SD** fue construido.

Otra de las desventajas involucra al software: aunque hoy en día los **SD** están siendo implementados por todo el mundo, no se tiene mucha experiencia en su diseño. Es difícil aún dar respuestas a preguntas como: ¿qué tipo de sistema operativo es el más apropiado?, ¿qué lenguaje de programación usar?, ¿cuáles aplicaciones son apropiadas para el sistema?, ¿cuánto deberían conocer los usuarios sobre la distribución?, ¿cuánto deberían hacer el sistema y los usuarios? Por ello se debe tener muy en cuenta los recursos físicos que se poseen y qué es lo que se quiere lograr con el **SD** para adaptarlo objetivamente a la realidad.

Por todo lo anterior, se considera vital y por consiguiente necesario, el uso de los sistemas distribuidos en el diseño e implementación de la presente tesis. A continuación se exponen las principales características de la Computación Grid.

1.2.5. Características de la Computación Grid. Aplicaciones.

Computación Grid se define como la aplicación de los recursos de varias computadoras conectadas en red a un solo problema a la vez. Constituye un nuevo paradigma de computación distribuida**,** donde los recursos de todas las computadoras interconectadas pueden ser tratados como una única supercomputadora de manera transparente. [5]

La Computación Grid es una herramienta versátil y poderosa debido a que:

1.- Hace un uso más efectivo de un determinado grupo recursos computacionales.
2.- Es una vía para resolver problemas que necesitan de una gran potencia de cálculo.
3.- Posibilita que los recursos de varias computadoras puedan enfocarse de manera cooperativa y sinérgica hacia un objetivo común.

La tecnología Grid brinda a las empresas el beneficio de la velocidad, lo que supone una ventaja competitiva, con lo cual se provee una mejora de los tiempos para la producción de nuevos productos y servicios. Facilita la posibilidad de compartir, acceder y gestionar información, mediante la colaboración y la flexibilidad operacional, aunando no sólo recursos tecnológicos dispares, sino también personas y aptitudes diversas. Otro de los aspectos es que tiende a incrementar la productividad, otorgando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten, cuando los necesiten.

El paralelismo puede estar visto como un problema, ya que una súper computadora es muy costosa. Pero, si se tiene disponibilidad de un conjunto de máquinas heterogéneas de pequeño o mediano porte, cuya potencia computacional sumada sea considerable, eso permitiría generar sistemas distribuidos de muy bajo costo y gran potencia computacional.

La computación Grid necesita, para mantener su estructura, de diferentes servicios como Internet, conexiones de 24 horas, durante todo el año, con banda ancha, servidores de capacidad, seguridad informática, firewalls, comunicaciones seguras, políticas de seguridad, y estar regida por normas ISO.

Aplicaciones de la Computación Grid:

Existen innumerables aplicaciones de la computación Grid, a continuación se exponen las que brindan potencialidades al sistema que se pretende diseñar:

Súper computación distribuida: Son aquellas aplicaciones cuyas necesidades no pueden ser satisfechas en un único nodo. Las necesidades se producen en instantes de tiempo determinados y consumen muchos recursos.

Sistemas distribuidos en tiempo real: Son aplicaciones que generan un flujo de datos a alta velocidad que debe ser analizado y procesado en tiempo real.

Servicios puntuales: Aquí no se tiene en cuenta la potencia de cálculo y capacidad de almacenamiento, sino los recursos que una organización puede considerar como no necesarios. Grid presenta a la organización esos recursos.

Proceso intensivo de datos: Son aquellas aplicaciones que hacen un gran uso del espacio de almacenamiento. Este tipo de aplicaciones desbordan la capacidad de almacenamiento de un único nodo y los datos son distribuidos por el sistema. Además de los beneficios por el incremento de espacio, la distribución de los datos a lo largo de la aplicación Grid permite el acceso a los mismos de forma distribuida.

Entornos virtuales de colaboración: Área asociada al concepto de Tele inmersión, de manera que se utilizan los enormes recursos computacionales de la Grid y su naturaleza distribuida para generar entornos virtuales 3D distribuidos.

Existen aplicaciones reales que hacen uso de mini-Grids, las cuales están centradas en el campo de la investigación en el terreno de las ciencias físicas, médicas y del tratamiento de la información. Además existen diversas aplicaciones en el campo de la seguridad vial. Por ejemplo, esta aplicación permite

traducir el riesgo de herir a un peatón y la resistencia del parachoques de un vehículo en una serie de datos, que ayudan a diseñar la solución de protección más adecuada.

Entre los primeros proyectos Grid, surge Information Power Grid (IPG), que permite la integración y gestión de recursos de los centros de la NASA. El proyecto SETI@Home a nivel mundial, de investigación de vida extra-terrestre, o búsqueda de vida inteligente en el espacio, puede ser considerado como precursor de esta tecnología, si bien la idea de Computación Grid es mucho más ambiciosa, puesto que no sólo se trata de compartir ciclos de CPU para realizar cálculos complejos, sino que se busca la creación de una infraestructura de computación distribuida, con interconexión de diferentes redes, de definición de estándares, de desarrollo de procedimientos para la construcción de aplicaciones.

Un gran número de corporaciones, grupos profesionales y consorcios universitarios están desarrollando frameworks y software para administrar proyectos Grid. La Comunidad Europea patrocina un proyecto Grid con aplicaciones en física de alta energía, observación terráquea y aplicaciones en la biología, y permite proyectos de ciencia electrónica muy populares como el proyecto del Genoma, que exigen interacción global.

Por los intereses de la presente tesis, se utilizará la computación Grid debido a todas las ventajas antes expuestas.

1.3. Aplicación de los Sistemas Distribuidos en la Bioinformática. Metodología MMH.

La utilización de las computadoras para resolver cuestiones biológicas inició con la aplicación de algoritmos al estudio de las interacciones de los procesos biológicos y las relaciones genéticas entre diversos organismos. En los últimos tiempos, las altas demandas de cómputo y la complejidad de las técnicas que emplean dichas computadoras para el análisis de los datos**,** implica al uso de los Sistemas de Cómputo Distribuido**,** con el objetivo de procesar de la forma más eficiente posible las informaciones biológicas.

En la Universidad Nacional de Irlanda se han desarrollado varias aplicaciones bioinformáticas en entornos distribuidos. Ejemplos de tales aplicaciones son **DPRml, DSEARCH** y **MultiPhyl** [6], esta última puede procesar cientos o miles de aminoácidos o nucleótidos alineados simultáneamente, y realiza computacionalmente, tareas intensivas: modelo de selección, árboles de búsqueda, e inicio de flejado (bootstrapping) de cada una de las alineaciones.

En la Universidad de las Ciencias Informáticas (UCI) se desarrolló el *"Módulo para el cálculo distribuido de mecánica molecular y mecánica cuántica",* con el que se redujo el tiempo de procesamiento del MOPAC [7]. En la actualidad, como parte del Proyecto *BioGrid* de la facultad 6, se han desarrollado algunas aplicaciones para reducir los tiempos de respuestas de algunas aplicaciones como *Gaussian*, *Vega* y la metodología Hipersuperficies de Múltiples Mínimos (**MMH**).

La metodología **MMH** permite la exploración completa del espacio multidimensional de agregados moleculares con la finalidad de determinar sus energías de asociación estadísticas y la obtención de la o las estructuras más favorecidas energéticamente en todo el espacio considerado [8]. Para esto utiliza varias aplicaciones: Granada, MOPAC y Q3. [28]

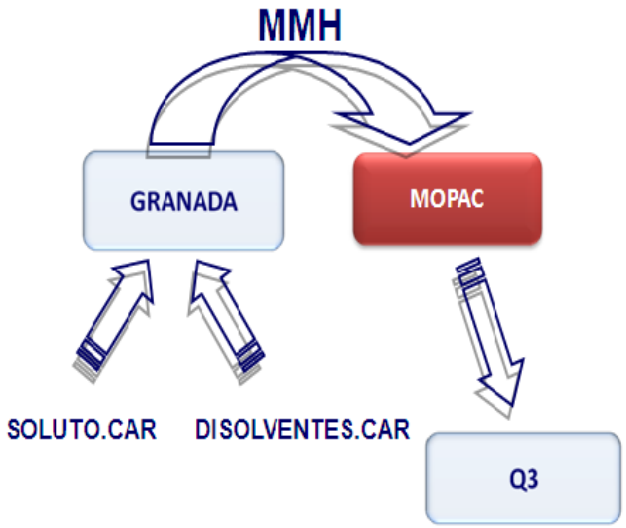

Figura 1.1. Pasos de la Metodología MMH.

En la presente tesis se pretende implementar una aplicación Web que permita a cualquier especialista, previamente autenticado, la utilización de la aplicación; sin importar dónde se encuentre geográficamente, mediante la creación de servicios de computación Grid en el Portal de Servicios BioGrid.

1.4. Herramientas y Tecnologías.

Muchos profesionales creen que un grupo de desarrollo debería organizarse teniendo en cuenta las potencialidades de los individuos altamente calificados, que hacen bien el trabajo y que raramente necesitan dirección. Esto constituye un craso error en la mayoría de los casos y una grave equivocación en el desarrollo de software. Por lo cual, es necesario un proceso que esté ampliamente disponible de

forma que todos los interesados puedan comprender su papel en el desarrollo en el que se encuentran implicados. Todo esto unido a una correcta selección de las herramientas, métodos, técnicas y procedimientos contribuye a obtener un producto de elevada calidad, de todo lo antes expuesto es de lo que se ocupa el presente epígrafe.

1.4.1. Metodología de Desarrollo de Software.

El desarrollo de software no es una tarea fácil, como resultado a este problema ha surgido una alternativa: el uso de las metodologías de desarrollo, las cuales imponen un proceso disciplinado sobre el desarrollo de software con el fin de hacerlo más predecible y eficiente; lo hacen desarrollando un proceso detallado con un fuerte énfasis en planificar**,** inspirado por otras disciplinas de la ingeniería.

Una metodología de desarrollo de software se refiere a un framework que es usado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. A lo largo del tiempo, una gran cantidad de métodos han sido desarrollados diferenciándose por su fortaleza y debilidad.

Escoger la metodología adecuada para realizar este ciclo de desarrollo es muy importante para lograr un sistema de alta calidad en un tiempo razonablemente corto. Existen varias metodologías de desarrollo de software, dentro de las cuales están **RUP** [9], **MSF** [10], **XP** [11] y **OpenUP**.

OpenUP es una de las metodologías ágiles de desarrollo de software. Ofrece las mejores prácticas de una variedad de líderes en ideas sobre desarrollo de software y comunidades de desarrollo que cubren una diversidad de conjuntos de perspectivas y necesidades de desarrollo. Preserva las características esenciales de RUP que incluye el desarrollo interactivo, casos de uso y escenarios de conducción de desarrollo, la gestión de riesgo y el enfoque centrado en la arquitectura.

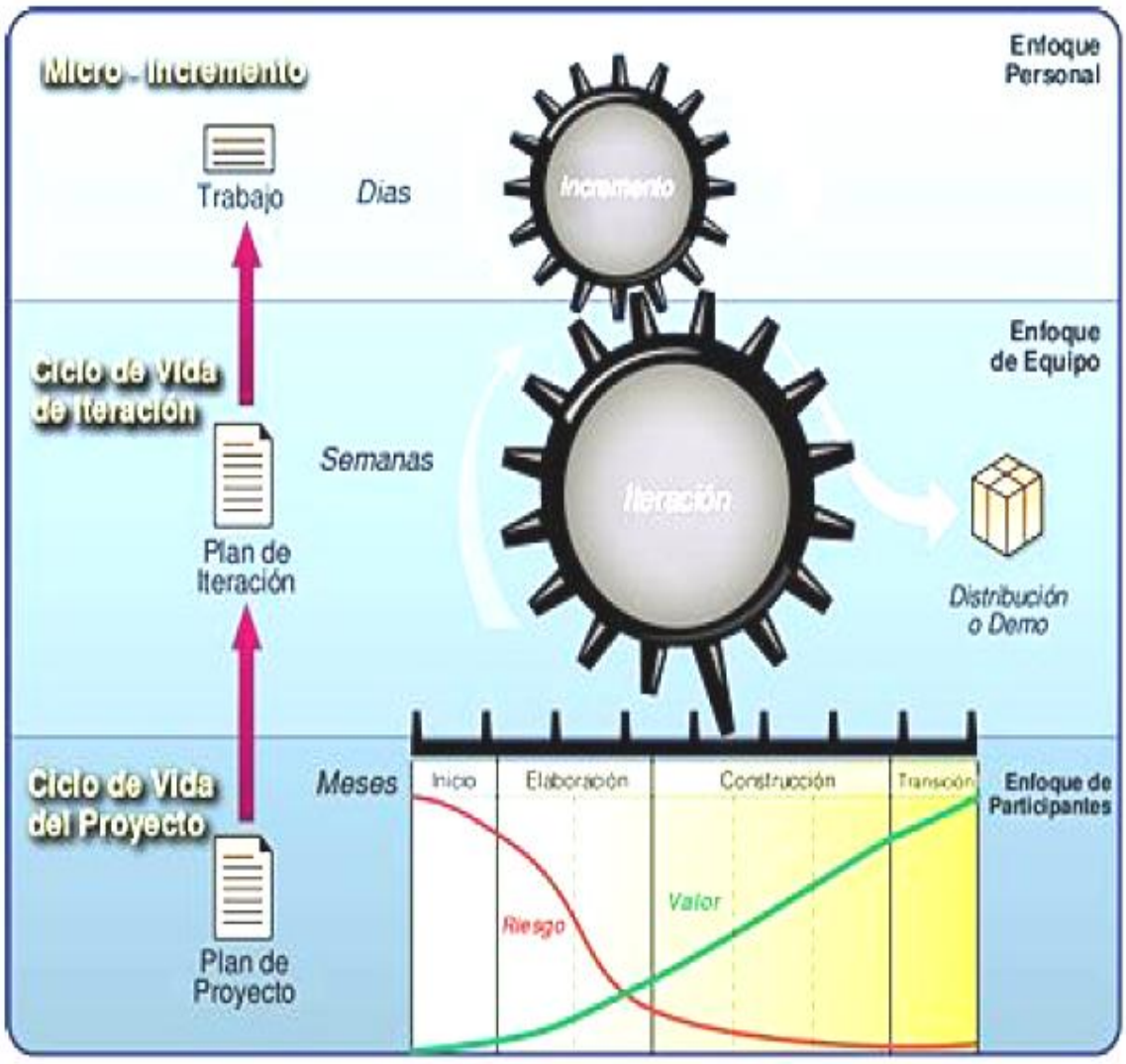

Figura 1.2. Capas de OpenUP.

Además de lo anterior **OpenUP** define un proceso de desarrollo de software completo, porque especifica un conjunto de componentes que guían el proceso de desarrollo hasta la obtención del producto. Su extensibilidad permite añadir actividades, artefactos y componentes a la metodología, según lo requiera el sistema a desarrollar. Por todas las razones expuestas **OpenUP** constituye la metodología seleccionada para el desarrollo de la aplicación propuesta en este trabajo.

1.4.2. Lenguaje de Modelado.

Para solucionar el problema planteado se utilizará como lenguaje de modelado UML (Unified Modeling Language), Lenguaje Unificado de Modelado, que es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos de un sistema de software [12]. Es utilizado para entender, diseñar, configurar y mantener la información sobre el sistema. UML puede ser utilizado en todos los métodos de desarrollo, dominios de aplicación, medios y etapas del ciclo de vida, incluye conceptos semánticos, notación y principios generales. Tiene partes estáticas, dinámicas, de entorno y organizativas. La especificación de UML no define un proceso estándar pero es muy útil en un proceso de desarrollo interactivo.

1.4.3. Herramienta Case.

Las Herramientas CASE (Computer Aided Software Engineering) [13] son un conjunto de métodos, utilidades y técnicas que facilitan la automatización del ciclo de vida del desarrollo del sistema de información, completamente o en algunas fases. Como ejemplo de herramientas CASE están: MagicDraw [14], Rational Rose [15], Umbrello [16], Visual Paradigm for UML, ArgoUML [17] y otras.

Visual Paradigm soporta la última versión del Lenguaje de Modelado Unificado (UML) y la Notación del Proceso de Modelado de Negocio (BPMN), y genera código para un gran número de lenguajes de programación. Además brinda una versión libre para uso no comercial. La herramienta fue desarrollada para una amplia gama de usuarios, incluyendo ingenieros de software, analistas de sistemas, analistas del negocio y arquitectos de sistemas. Permite la integración con varias herramientas de Java, y brinda además una gran interrelación con otras herramientas CASE como Rational Rose.

Por todas las ventajas antes expuestas se decidió utilizar Visual Paradigm como Herramienta CASE para el diseño de la presente tesis.

1.4.4. IDE y Lenguaje de Programación.

Lenguaje de Programación.

El lenguaje de programación seleccionado es Java. La razón principal de esta selección es que es independiente de plataforma y arquitectura de red. Por eso una aplicación escrita en Java, puede ejecutarse en cualquier sistema. Esta portabilidad es extremadamente importante para cualquier sistema distribuido, ya que se espera que los clientes puedan correr en múltiples sitios y en diferentes plataformas.

En los primeros días de Java, gran parte de sus críticas se originaban de su pobre rendimiento respecto a lenguajes nativos como C y Fortran. Mucho ha cambiado desde entonces. Actualmente se han producido enormes mejoras en el rendimiento de la Máquina Virtual de Java (JVM), principalmente atribuida a la introducción del compilador just-in-time [18] y la tecnología hotspot [19]. Estas mejoras han dado lugar a que la ejecución de la JVM, sea comparable a la de otros lenguajes nativos. Otro de los aspectos más importantes de Java es el modelo de seguridad. Este simplifica enormemente la implementación de una estricta política de seguridad para una aplicación Java [20], haciendo posible que las aplicaciones puedan cargar dinámicamente código, sin tener que preocuparse por las posibles implicaciones para la seguridad. Además de las características mencionadas con anterioridad, Java constituye un lenguaje "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico", lo cual es muy importante debido a las características heterogéneas de nuestro entorno.

Java Server Pages (JSP).

Constituyen un conjunto de tecnologías que posibilitan la generación dinámica de páginas web, combinando código Java con un lenguaje de marcas como HTML ó XML, para generar el contenido de la página. Como consecuencia, se pueden crear y mantener páginas JSP con las herramientas convencionales para HTML/XML, permitiendo además separar la presentación de la lógica del negocio.
La tecnología JSP, acarrea el paradigma *"escribe una vez, corre en cualquier lugar"* a las páginas web interactivas. Las páginas JSP, pueden ser trasladadas fácilmente a varias plataformas y entre servidores web, sin la necesidad de realizarle cambios de ningún tipo. Esta característica, permite mezclar el poder de la tecnología Java por el lado del servidor, con las ventajas de las páginas HTML estáticas. [25]

Entorno de Desarrollo Integrado.

Un entorno de desarrollo integrado (IDE), es un programa compuesto por un conjunto de herramientas para un programador. Es decir, es un entorno de programación que ha sido empaquetado como un programa de aplicación y consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica de usuario.

NetBeans.

Es un Entorno Integrado de Desarrollo gratuito, de código abierto para desarrolladores de software. Puede obtener todas las herramientas que necesite para crear aplicaciones profesionales para el escritorio, la

empresa, la web (...) con el lenguaje Java (...) NetBeans IDE es fácil de instalar y de uso instantáneo y se ejecuta en varias plataformas...” [21]

Este IDE fue seleccionado principalmente por su potente editor de código, la interfaz amigable que presenta, la facilidad de su creador de interfaz gráfica de usuario para web y para escritorio y la presencia de un módulo para desarrollar Portlets [22] para Gridsphere.

En resumen, NetBeans brinda un poderoso IDE basado en Java, lo suficientemente potente para desarrollar aplicaciones empresariales y ahorrar tiempo de desarrollo permitiendo que el desarrollador se centre en el núcleo.

Eclipse.

Es un IDE extensible y libre, desarrollado en la actualidad por la Fundación Eclipse, que es una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto. Con respecto a las plataformas, la Fundación Eclipse proporciona instalaciones binarias para varios sistemas operativos, entre los que se pueden mencionar: Windows, Linux, Solaris y Mac OS.

La plataforma Eclipse, ofrece características esenciales entre las que se encuentran el resaltado de sintaxis, la compilación incremental de código, un depurador (debugger), un administrador de proyectos e interfaces para el control estándar de sistemas. Posibilita además la refactorización de código, listado de tareas, y soporte para pruebas unitarias con JUnit.

Tiene una arquitectura basada en plug-ins y las APIs que provee el Entorno de Desarrollo para Plug-in (PDE), haciendo de Eclipse un marco de trabajo modular extensible y permitiendo agregar soporte para nuevos tipos de editores, visores o lenguajes de programación, de forma fácil. [26]

1.4.5. Herramientas y Tecnologías para el desarrollo.

A continuación se abordan las herramientas y tecnologías utilizadas para el desarrollo de la aplicación de la presente tesis:

Plataforma de Tareas Distribuidas.

La Plataforma de Tareas Distribuidas: T-arenal, es un sistema de cómputo distribuido programado en java y basado en el software Java. Ha sido utilizado para dar solución a varios problemas de la Bioinformática, y brinda un modelo de programación de alto nivel basado en el paradigma de la POO (Programación Orientada a Objetos), utilizando a RMI para el paso de mensajes.

El sistema T-arenal consiste en un servidor central (T-arenal server) y uno o más clientes (T-arenal clients).

El servidor T-arenal maneja toda la información concerniente al sistema, gestiona la transferencia de ficheros vía sockets TCP, y planifica el orden en que los trabajos serán atendidos. Estos trabajos son ejecutados en los clientes, los cuales pueden estar dedicados o no. Las personas que envían trabajos para el sistema son denominados “usuarios”.

Portal de Servicios basado en GRIDSPHERE.

Un portal de Internet es un sitio web cuyo objetivo es ofrecer al usuario, de forma fácil e integrada, el acceso a una serie de recursos y de servicios, entre los que suelen encontrarse buscadores, foros, documentos, aplicaciones, compra electrónica, entre otros. Principalmente están dirigidos a resolver necesidades específicas de un grupo de personas o de acceso a la información y servicios de una institución pública o privada.

Existen tres modalidades de portales:

1. Portales horizontales, también llamados portales masivos o de propósito general, se dirigen a una audiencia amplia, tratando de llegar a toda la gente con muchas cosas. Como ejemplo de portales de esta categoría están: Terra, AOL, AltaVista, UOL, Lycos, Yahoo y MSN, entre otros.

2. Portales verticales, se dirigen a usuarios para ofrecer contenido dentro de un tema específico como puede ser un portal de música, empleo, inmobiliario, un portal de finanzas personales, arte o de deportes.

3. Portales diagonales: se trata de una mezcla entre el portal horizontal y el vertical. Se trataría de portales que utilizan redes sociales o aplicaciones generalistas como: Facebook, Linkd, Flickr o YouTube complementados con contenidos y/o utilidades dirigidas a un público muy concreto.

Los portales normalmente tienen programación que requiere muchos recursos computacionales y por su alto tráfico generalmente se hospedan en servidores de Internet dedicados.

En el Polo de Bioinformática de la Universidad de las Ciencias Informáticas fue desarrollado un Portal de Servicios basado en Gridsphere. Con el portal se pretende lograr la integración de todos los proyectos de la Facultad 6. En dicho portal se implementará la solución que se propone, mediante la aplicación de una serie de servicios y recursos.

Servidor Web Apache Tomcat.

Funciona como un contenedor de Servlets, que implementa las especificaciones de Sun Microsystems para Servlets y JSP. Fue creado bajo el proyecto Jakarta de la Fundación Apache, siendo mantenido por la comunidad de desarrollo en un ambiente participativo, logrando destacar como un producto robusto, altamente eficiente y uno de los más potentes contenedores de Servlets existentes. [27]

Tomcat es gratis, fácil de instalar, se ejecuta en máquinas con pocos recursos y es compatible con las API más recientes de Java. Apenas ocupa espacio, teniendo su código binario un tamaño total de algo más de un megabyte, de modo que no es raro que se ejecute tan de prisa. Además, su remarcada estabilidad y la capacidad de ejecución multiplataforma, lo han colocado como uno de los servidores más utilizados para el despliegue de aplicaciones web basadas en la tecnología Java.

Framework Spring.

Es un framework de código abierto de desarrollo de aplicaciones para la plataforma Java, y no obliga a usar un modelo de programación en particular. Por su diseño ofrece mucha libertad a desarrolladores y soluciones muy bien documentadas y fáciles de usar para las prácticas comunes. Diseñado en módulos, con funcionalidades consistentes en otros módulos, facilita el desarrollo de funcionalidades específicas y hace que la curva de aprendizaje sea favorable al desarrollador.

El núcleo de Spring se basa en el principio de inyección de dependencias. Esta técnica, hace externa la creación y el manejo de las dependencias de los componentes, logrando mayor limpieza y claridad en el código, pues provee en tiempo de ejecución, todas las instancias de clases de la aplicación y las dependencias que ellas necesitan.

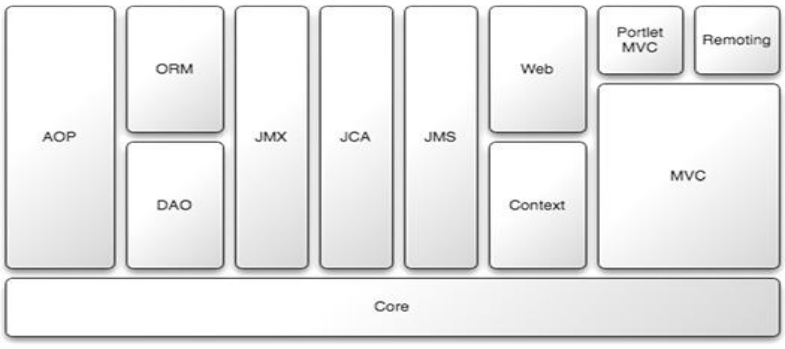

Figura 1.3. Módulos que componen al framework Spring.

Spring brinda la posibilidad de que algunos de sus módulos permiten trabajar con otras alternativas, pudiendo integrarse incluso con otros frameworks y librerías, sin necesidad de que el programador tenga que implementarlas.

Hibernate.

Es una herramienta utilizada para el mapeo objeto – relacional en Java. Hace más fácil el mapeo de atributos entre una Base de Datos (BD) tradicional relacional y el modelo orientado a objetos de cualquier aplicación. Hibernate convierte los tipos de datos utilizados por Java, en los tipos de datos definidos por SQL (Structured Query Language) y viceversa. Posibilita además la generación de sentencias SQL para la persistencia y recuperación de los objetos en la BD, liberando al programador de responsabilidades al igual que el framework Spring, reduciendo así el tiempo de desarrollo del software.

En la Figura 1.4 se muestra un esquema que ilustra la arquitectura base de Hibernate, la cual resume su acción como mediador entre la aplicación y la BD.

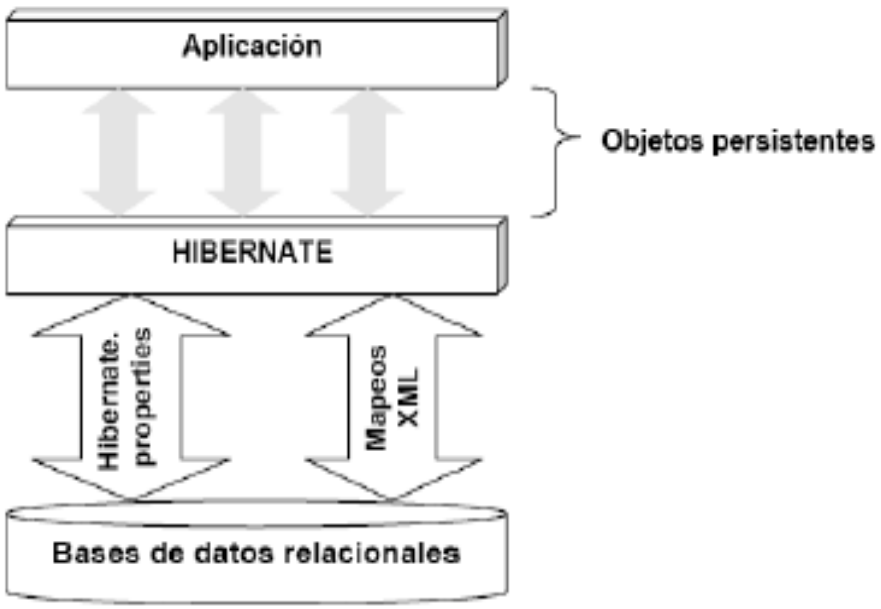

Figura 1.4. Arquitectura Base de Hibernate.

PostgreSQL.

Es un motor de bases de datos relacionales que verifica integridad referencial con gran funcionalidad como base de datos, aunque un poco más lenta que otros motores. Su licencia es tipo BSD (Berkeley Software Distribution). Algunas de sus principales características son:

- Es multiplataforma, puede ejecutarse en: Linux, Unix, Mac OS, Beos y Windows.
- Documentación muy bien organizada, pública y libre, con comentarios de los propios usuarios.
- Comunidades muy activas, varias comunidades en castellano.
- Altamente adaptable a las necesidades del cliente.
- Soporte nativo para los lenguajes más populares del medio: PHP, C, C++, Perl, Python.
- Soporte de todas las características de una base de datos profesional (disparadores de excepciones, funciones de almacenamiento, secuencias, relaciones, reglas, tipos de datos definidos por usuarios y vistas materializadas).
- Soporte de protocolo de comunicación encriptado por SSL.
- Extensiones para alta disponibilidad, nuevos tipos de índices, datos espaciales y minería de datos.

Por todas las ventajas que ofrece PostgreSQL, se decidió utilizarlo como gestor de base de datos para el desarrollo de la aplicación.

1.5. Conclusiones

En este capítulo se concluye que: el uso de los sistemas distribuidos, específicamente de la computación Grid es vital para el desarrollo de la solución, por lo cual se decide utilizar el Portal de servicios GRIDSPHERE para implementar la aplicación Web. Además, teniendo en cuenta principalmente las posibilidades y limitaciones que brindan las herramientas y metodologías analizadas a la solución distribuida que requiere el problema de la investigación, el tamaño de este, así como las condiciones de nuestro país y la tendencia al software libre, se decidió utilizar para el desarrollo:

- ✓ La metodología OpenUP, pues se adapta a condiciones específicas de proyectos ágiles y pequeños, permitiendo entregar un software con calidad.
- ✓ Plataforma de Tareas Distribuidas: T-arenal, la cual se desarrolló en la Universidad y representa una alternativa asequible en costes y recursos.

- ✓ UML como lenguaje de modelado.
- ✓ Visual Paradigm como Herramienta Case.
- ✓ Lenguaje de Programación Java y JSP.
- ✓ La herramienta Hibernate para el mapeo objeto-relacional en Java.
- ✓ La herramienta Spring para el desarrollo de la aplicación.
- ✓ Servidor Web Apache Tomcat 6.0.
- ✓ Como IDE se escogió:
  - Eclipse para desarrollar la aplicación Web.
  - NetBeans para integrar la aplicación al portal de servicios.
- ✓ Como gestor de Base de Datos se seleccionó el PostgreSQL.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

Características del Sistema.

Luego de trazadas las pautas de herramientas y metodologías que serán utilizadas en el proceso de desarrollo de software, se realiza un levantamiento de requisitos y el Modelo de Negocio o de Dominio, según sea más conveniente, para lograr comprender la manera de darle la solución más óptima y viable a la problemática investigada. En el presente capítulo, se describen las principales características del sistema, desglosadas en los requisitos funcionales y no funcionales, los actores que intervienen en los mismos, así como el diagrama de casos de uso del sistema.

2.1 Breve descripción del Sistema.

El sistema a desarrollar permitirá interactuar con la Plataforma de Tareas Distribuidas: T-arenal, solucionando el problema que tiene la actual implementación de la metodología MMH, la cual no permite acceder desde lugares externos de la universidad. Esta versión estará compuesta por dos componentes: módulo del servidor y módulo del cliente. El módulo del servidor permitirá la interacción con los servicios del Gridsphere, que incluye todas las funcionalidades como los web services y los portlets. El módulo cliente introduce nuevas características a T-arenal: el cliente puede configurar a través de la interfaz correspondiente y recibir los resultados de la metodología MMH.

2.2. Modelo de Dominio.

Debido a la relativa simplicidad del sistema y sobre todo por el previo conocimiento que se posee acerca de su funcionamiento, se consideró no realizar un Modelo de Negocio para la comprensión de la problemática a resolver, siendo suficiente un Modelo de Dominio (Modelo Conceptual) pues los procesos no estaban bien definidos. Dicho modelo, es una representación visual de objetos y conceptos del mundo real significativos para el problema de interés. Representa clases conceptuales del dominio del problema, no de componentes de software.

A continuación se muestra el modelo conceptual de la problemática a resolver, identificando los objetos fundamentales y sus relaciones.

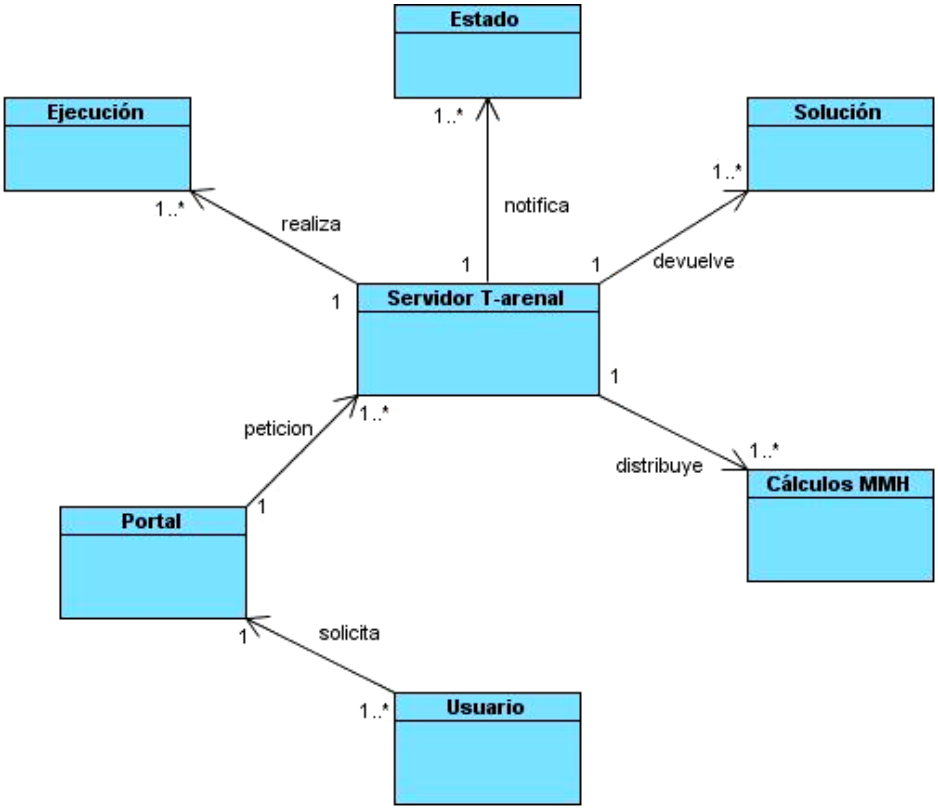

Fig. 2.1 Modelo de Dominio.

Descripción de los Objetos:

Servidor T-arenal: Representa la entidad que recibe las peticiones de los problemas, y se encarga de ejecutar, notificar y devolver la solución.

Ejecución: Representa la entidad que realiza la ejecución del problema.

Estado: Representa la entidad que notifica el estado de la ejecución.

Solución: Representa la entidad que devuelve la solución.

Portal: Representa la entidad que recibe las peticiones de un determinado *usuario* y las envía al *Servidor T-arenal*.

Cálculos MMH: Representa la entidad encargada de realizar los cálculos de la metodología MMH.

Usuario: Representa la entidad que tiene un problema y desea darle solución.

2.3. Especificación de los requisitos del sistema.

2.3.1. Requisitos funcionales.

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir, y se mantienen invariables sin importar con que propiedades o cualidades se relacionen. En particular el sistema debe cumplir:

R.1. Iniciar ejecución de cálculos.

R.1.1. Interrumpir la ejecución.

R.1.2. Mostrar estado de la ejecución.

R.2. Obtener solución.

R.2.1. Descargar solución.

R.2.2. Eliminar solución.

R.3. Autenticar Usuario.

R.4. Gestionar Usuario.

R.4.1. Adicionar Usuario.

R.4.2. Modificar Usuario.

R.4.3. Eliminar Usuario.

2.3.2. Requisitos no funcionales.

Los requisitos no funcionales son características o propiedades que el sistema debe tener, de manera que sea confiable y seguro.

- ✓ ***Usabilidad:***

El sistema podrá ser utilizado por cualquier individuo con conocimientos de computación y química. No se requiere vasta experiencia en su uso para explotar todas sus funcionalidades.

- ✓ ***Software:***

Para el correcto funcionamiento del sistema se requiere la instalación de una máquina virtual de Java y se necesita que en los clientes existan los Sistemas Operativos Linux o Windows.

Cliente:

- ✓ Cualquier sistema operativo con interfaz gráfica y red.
- ✓ Navegador Web Internet Explorer (5.5 o superior) o Mozilla Firefox 2.0 o superior.

Servidor:

- ✓ Se recomienda sistema Operativo Linux.
- ✓ Servidor de Base de Datos PostgreSQL 8.2.
- ✓ Servidor Web Apache Tomcat 6.0.
- ✓ Java Runtime Enviroment (JRE) versión 1.5.

✓ ***Hardware:***

Se requiere que las computadoras tengan conexión a la red local o externa.

Cliente:

- ✓ Mínimo 256 MB de RAM.

Servidor:

- ✓ Microprocesador Pentium IV o superior.
- ✓ Mínimo 1.0 Gigabytes (GB) de RAM.
- ✓ Capacidad de disco duro 40 Gigabytes mínimo.

✓ ***Seguridad:***

El usuario debe identificarse antes de acceder a cualquier acción del sistema. Solo podrán utilizar el sistema aquellos usuarios a los que se les asignaron determinados permisos en el servidor. El administrador de servidor que es el encargado de asignar estos permisos.

2.4. Actores y casos de uso del sistema.

2.4.1. Actores del sistema.

Un actor del sistema es una entidad externa representada por un ser humano, un software o una máquina que interactúa con este. El actor, generalmente le transmite al sistema algunos eventos y en otro caso simplemente obtiene algo de él.

Actores	Descripción
Especialista	Representa al usuario que interactúa con el sistema para realizar los cálculos.
T-arenal	Representa al actor del sistema que realiza los cálculos distribuidos.
Administrador	Representa el rol que gestiona los usuarios de la aplicación web.

2.4.2. Casos de Uso del Sistema.

Los casos de uso son descripciones de las funcionalidades con las que contará el sistema independiente de la implementación.

Orden	Nombre	Prioridad	Breve Descripción
1	Realizar Cálculos	Crítico	Constituye el objetivo fundamental del sistema e incluye todos los cálculos de la metodología MMH.
2	Obtener Solución	Crítico	Resulta imprescindible pues muestra las soluciones de los cálculos que espera el usuario.
3	Autenticar	Crítico	El usuario debe iniciar su sesión para poder acceder a las funcionalidades.
4	Gestionar Usuario	Crítico	Resulta imprescindible porque permite al administrador gestionar a los usuarios que podrán autenticarse al sistema.
5	Mostrar Ejecuciones	Secundario	El usuario tendrá la posibilidad de ver todas sus ejecuciones.
6	Mostrar Estado	Secundario	El caso de uso comienza cuando el usuario desea visualizar el estado actual de la ejecución.
7	Eliminar Solución	Secundario	El caso de uso comienza cuando el usuario desea eliminar la solución. El usuario solo puede eliminar la solución que haya obtenido anteriormente.

8	Descargar Solución	Secundario	El caso de uso comienza cuando el usuario desea descargar la solución.

2.4.3. Diagrama de Casos de Uso del Sistema.

En la Figura 2.2 se muestra el Diagrama de Casos de Uso del Sistema. El *Especialista* inicia los casos de uso: *Autenticar*, *Realizar Cálculos* y *Obtener Solución*. El *Administrador* puede al igual que el *Especialista* realizar los mencionados casos de uso y además realiza *Gestionar Usuario*. El caso de uso *Realizar Cálculos* inicia al actor *T-arenal* el cual representa a la Plataforma de Cálculo Distribuido.

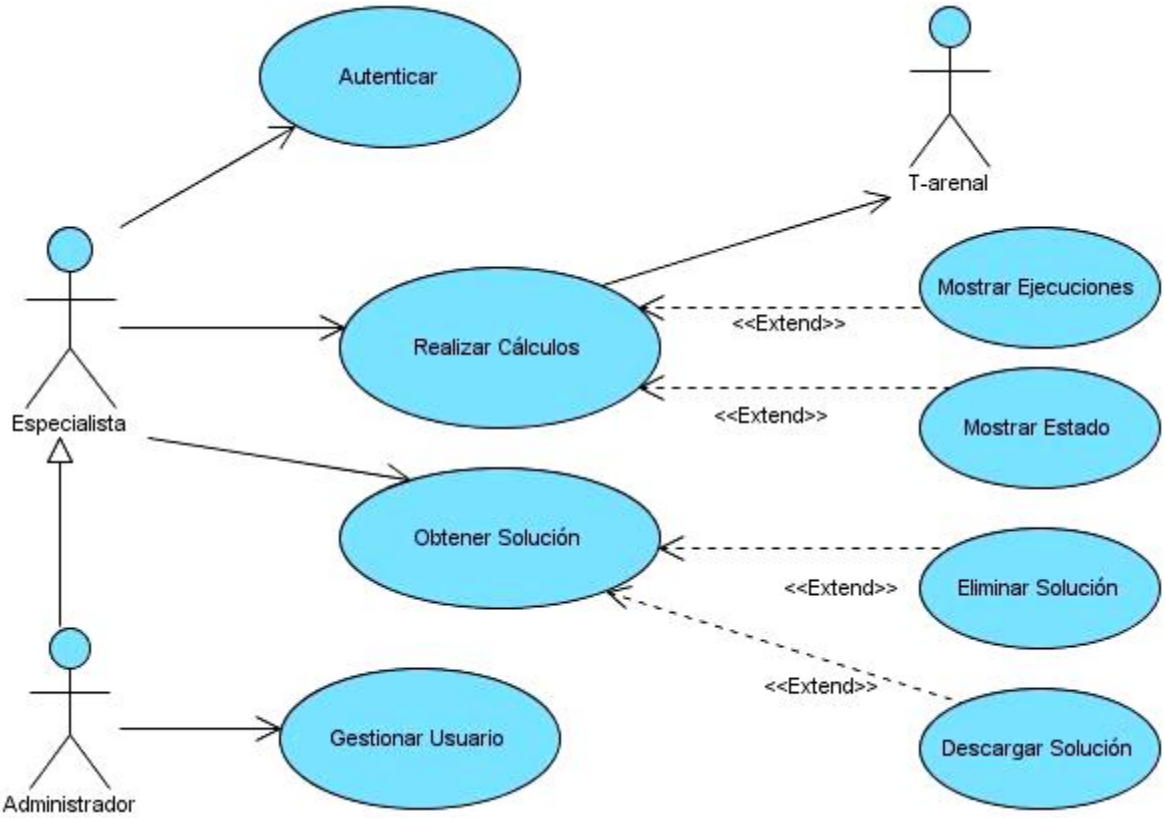

Fig. 2.2 Diagrama de Casos de Uso del Sistema.

2.4.4 Patrón de Casos de Uso utilizado.

En el diseño de los casos de uso del sistema se utilizó el patrón CRUD (Creating, Reading, Updating and Deleting), el cual propone identificar el caso de uso *Realizar Cálculos* y *Gestionar Usuario*. CRUD Modela todas las operaciones que se pueden realizar sobre una parte de información de cierto tipo (o sea en una misma entidad), tal como crearla, leerla, actualizarla y eliminarla.

✓ **Aplicación:** Este patrón debe ser usado cuando todos los flujos contribuyen al mismo valor de negocio, son cortos y sencillos.

2.5 Conclusiones.

En este capítulo se identificaron los requerimientos funcionales y no funcionales, los actores y casos de usos identificados para la realización del diseño del sistema, y las relaciones entre ellos.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

Diseño del sistema

En el presente capítulo se explica la representación arquitectónica del sistema, haciendo énfasis en el estilo y el patrón de arquitectura utilizados, así como en los patrones de diseño más importantes. Se muestran además los diagramas de clases de los paquetes relevantes de cada subsistema, así como los diagramas de secuencias necesarios para comprender el funcionamiento del presente trabajo.

3.1. Estilo de arquitectura.

El estilo de arquitectura utilizado fue el de cliente – servidor debido a la lógica distribuida del sistema como puede verse en la Figura 3.1. Es un modelo que provee usabilidad, flexibilidad, interoperabilidad y escalabilidad en las comunicaciones y que permite además a los usuarios obtener acceso a la información en forma transparente, aún en entornos multiplataforma y de recursos heterogéneos.

En el modelo cliente - servidor, el cliente envía un mensaje solicitando un determinado servicio a un servidor específico (hace una petición) y este envía uno o varios mensajes con la respuesta (provee el servicio). En un sistema distribuido cada máquina puede cumplir el rol de servidor para algunas tareas y el rol de cliente para otras.

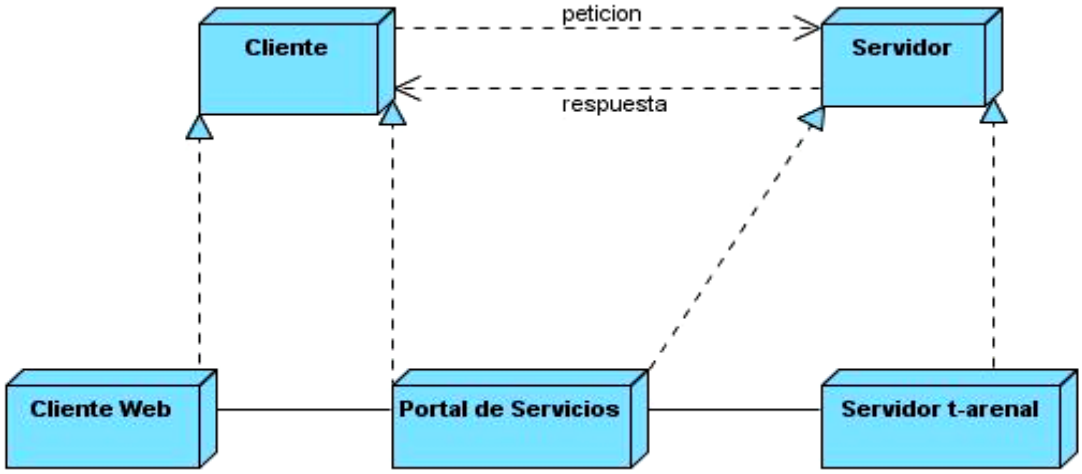

Figura 3.1 Estilo de arquitectura cliente – servidor aplicado al sistema.

Como se puede apreciar en la Figura 3.1, el subsistema *Cliente Web* se comporta como cliente porque solo hace peticiones al subsistema *Portal de Servicios*, este a su vez actúa como cliente y como servidor porque da respuesta al subsistema *Cliente Web* y envía peticiones al *Servidor T-arenal*. El subsistema *Servidor T-arenal* se comporta como servidor porque responde a las solicitudes del *Portal de Servicios*.

3.2. Patrón de Arquitectura.

Para la creación del sistema se utilizó el framework Spring el cual propone una arquitectura Modelo-Vista-Controlador (MVC), para una mejor organización y manejo de las vistas (.jsp). Este patrón separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos. [23]

- Modelo: Agrupa los datos y la lógica de la aplicación.
- Vista: Se encarga de mostrar los datos recogidos en el modelo, generalmente a través de una interfaz.
- Controlador: Se encarga de procesar las interacciones con el usuario y realizar los cambios apropiados en el modelo o en la vista.

Ventajas: Brinda soporte de vistas múltiples, es decir, que la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente, debido a que la vista se halla separada del modelo y no existe dependencia directa entre ellos.

Framework MVC de Spring: Es uno de los módulos del framework Spring que implementa una arquitectura Modelo-Vista-Controlador, el cual es utilizado como base para desarrollar la aplicación Web del presente trabajo. Este módulo, define un conjunto de interfaces, que deben ser implementadas en la aplicación Web, para dar respuesta a un evento determinado. En la Figura 3.2 se detalla cómo está implementado el patrón Modelo-Vista-Controlador sobre el framework Spring.

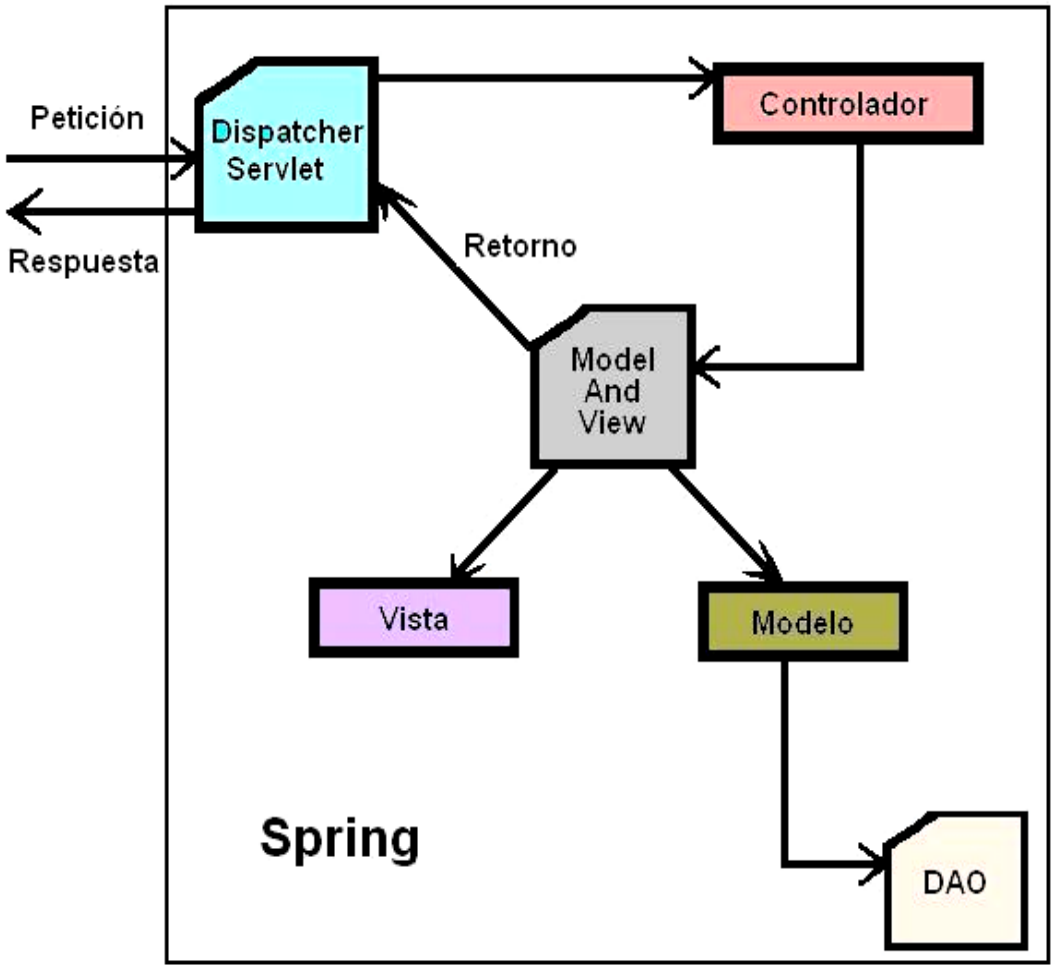

Figura 3.2 Implementación del patrón MVC sobre el framework Spring.

3.3. Patrones de diseño.

Un patrón de diseño es:

- ✓ Una solución estándar para un problema común de programación.
- ✓ Una técnica para flexibilizar el código haciéndolo satisfacer ciertos criterios.
- ✓ Un proyecto o estructura de implementación que logra una finalidad determinada.
- ✓ Un lenguaje de programación de alto nivel.
- ✓ Una manera más práctica de describir ciertos aspectos de la organización de un programa.
- ✓ Conexiones entre componentes de programas.
- ✓ La forma de un diagrama de objeto o de un modelo de objeto. [24]

A continuación se ilustran los patrones utilizados para el diseño del sistema:

3.2.1. Patrón GRASP.

Los patrones GRASP (Patrones de Software para la Asignación General de Responsabilidades) se encargan de definir normas o principios fundamentales en el diseño orientado a objetos y las responsabilidades de estos de acuerdo a su comportamiento. Estos patrones ayudan a la compresión de las soluciones implementadas.

Dentro de los patrones GRASP, por su gran utilidad se destacan (Experto, Creador, Alta cohesión, Bajo acoplamiento y Controlador).

✓ **Experto (Expert)**

Consiste en asignar una responsabilidad a una clase que tiene la información necesaria para la realización de la asignación. Se puede ver la aplicación de este patrón en la Figura 3.4, viendo que la clase *UsuariosService* engloba las funcionalidades de gestionar los usuarios.

Beneficios: Se mantiene el encapsulamiento de la información y el bajo acoplamiento.

✓ **Creador (Creator)**

Consiste en asignarle a la clase B la responsabilidad de crear una instancia de clase A en uno de los siguientes casos:

- B agrega los objetos A.
- B contiene los objetos A.
- B registra las instancias de los objetos A.
- B utiliza específicamente los objetos A.
- B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado.
- B es un creador de los objetos A.
- Si existe más de una opción, prefiera la clase B que agregue o contenga la clase A.

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

Beneficios: Favorece el bajo acoplamiento.

La ejemplificación de este patrón se ilustra en la Figura 3.3, cuando la clase *ProviderManager* crea instancias en la clase *AuthenticationProvider*.

✓ **Bajo Acoplamiento (Low Coupling)**

Problema: ¿Cómo soportar bajas dependencias, bajo impacto del cambio e incremento de la reutilización?

Solución: Asignar una responsabilidad de manera que el acoplamiento permanezca bajo.

Beneficios: Disminuye el impacto de los cambios. Se facilita el entendimiento y la reutilización.

Este patrón se pone de manifiesto en la Figura 3.4, cuando la clase de acceso a datos *AbstractHibernateDAO* reutiliza funcionalidades de la clase *UsuariosService*, logrando así una baja dependencia.

- ✓ **Alta cohesión (High Cohesion)**

Problema: ¿Cómo mantener la complejidad manejable?

Solución: Asignar una responsabilidad de manera que la cohesión permanezca alta.

Beneficios: Se facilita la comprensión y mantenimiento. Favorece el bajo acoplamiento y la reutilización.

El uso de este patrón se evidencia en la Figura 3.4, cuando el subsistema *Spring* logra manejar todo el flujo del portal y gestiona en el controlador las funcionalidades, logrando así una complejidad manejable.

- ✓ **Controlador (Controller)**

Problema: ¿Quién debería ser el responsable de gestionar un evento de entrada al sistema?

Solución: Asignar una responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase.

Beneficios: Posibilita la organización y claridad en el diseño. Favorece el bajo acoplamiento.

Este patrón se ve aplicado en la Figura 3.3, cuando en el subsistema *Spring* el *DispatcherServlet* envía un mensaje de evento del sistema a la clase *PresentationController*.

3.3.2. Inyección de Dependencia (ID).

Este patrón, hace externa la creación y el manejo de las dependencias de los componentes, logrando mayor limpieza y claridad en el código, pues provee en tiempo de ejecución, todas las instancias de clases y las dependencias que ellas necesitan. Es una forma para mitigar la proliferación de dependencias, fomentando el bajo acoplamiento entre los componentes y reduciendo la cantidad de código de infraestructura que se debe escribir. Su uso, se ve claramente reflejado en el contenedor de dependencias ofrecido por Spring.

3.4. Diagramas de clases de diseño.

Un diagrama de clases de diseño es un diagrama de estructura estática que describe gráficamente las especificaciones para las clases e interfaces de una aplicación, y además contiene información como

clases, asociaciones y atributos, interfaces con sus operaciones y constantes, métodos o funciones, navegabilidad, dependencias y multiplicidad en algunas de sus relaciones.

Los diagramas de clases del diseño permiten que se describa detalladamente y de forma gráfica las especificaciones de las clases de software.

Las clases del diseño de los principales casos de uso son mostradas a continuación:

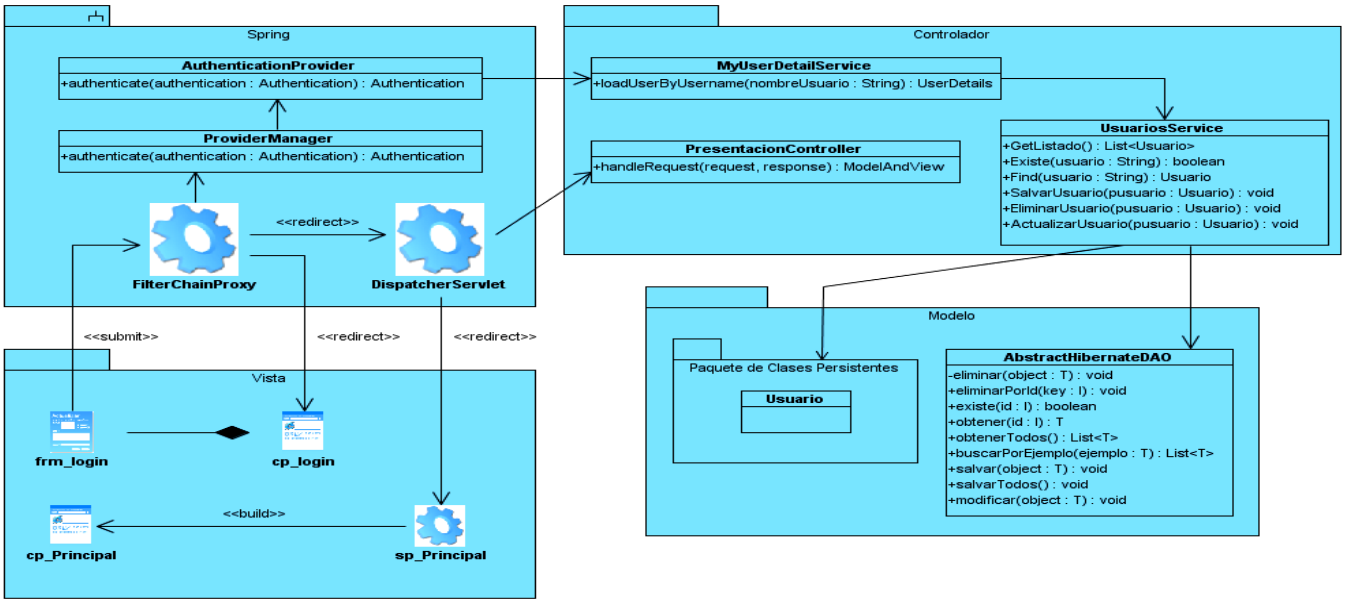

Figura 3.3 Diagrama de clases del diseño: CU Autenticar.

En la Figura 3.3 se ilustra como el subsistema *Spring* que cuenta con las clases *ProviderManager* y *AuthenticatioProvider,* utiliza en el Controlador la funcionalidad de los usuarios para autenticarse mediante la clase *UsuarioService*, la cual accede en el Modelo a la clase de acceso a datos *AbstractHibernateDAO*.

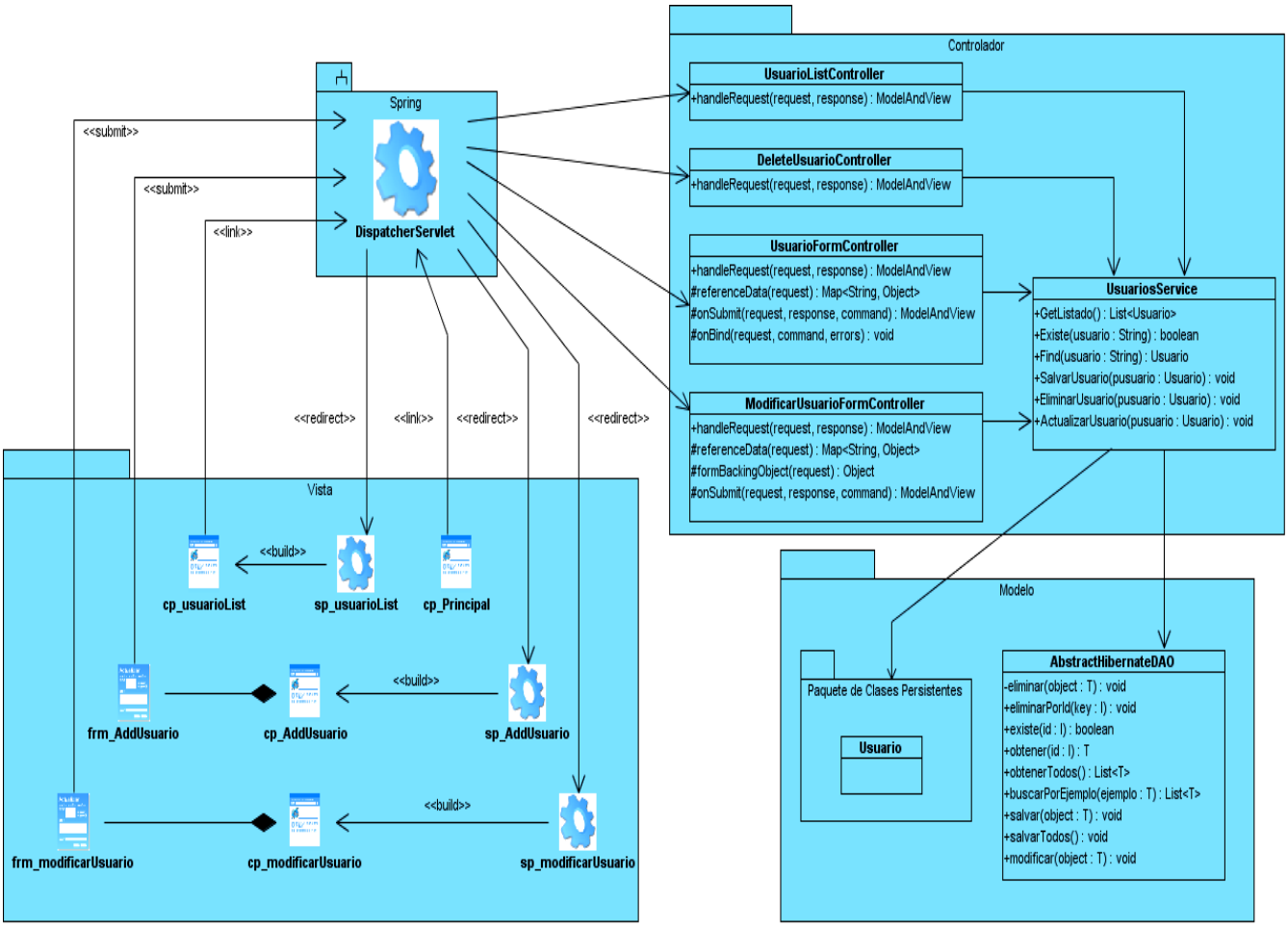

Figura 3.4 Diagrama de clases del diseño: CU Gestionar Usuario.

La Figura 3.4 muestra como el subsistema *Spring* utiliza las funcionalidades en el Controlador mediante las clases *UsuarioListController*, *DeleteUsuarioController*, *UsuarioFormController* y *ModificarUsuarioController*, las cuales convergen en la clase *UsuariosService*. Esta clase accede en el Modelo a la clase de acceso a datos *AbstractHibernateDAO*.

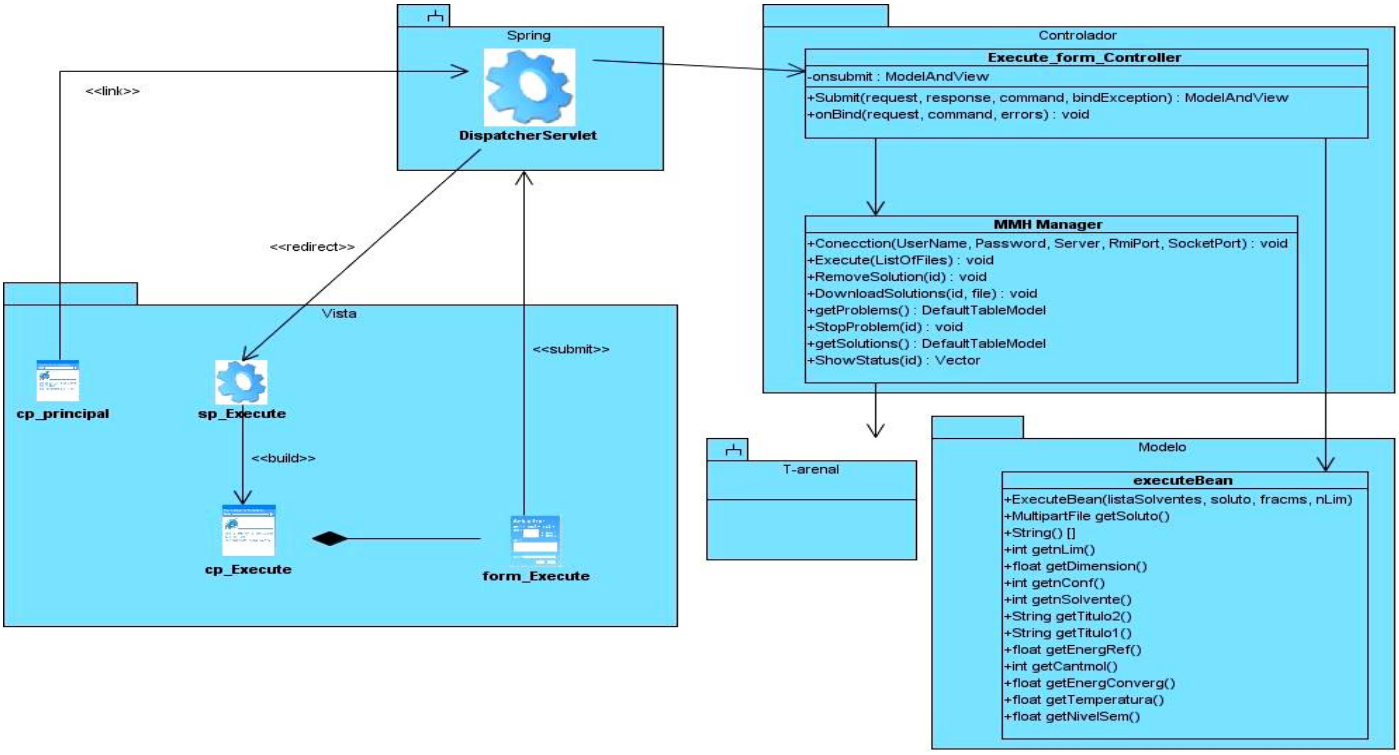

Figura 3.5 Diagrama de clases del diseño: CU Realizar Cálculos.

En la Figura 3.5 se puede observar que el subsistema *Spring* utiliza en el Controlador la clase *Execute_form_Controller*, la cual accede en el Modelo a la clase de acceso a datos *executeBean*. La clase *Execute_form_Controller* utiliza también la clase *MMHManager*, la cual se comunica con el subsistema *T-arenal*, que es la entidad encargada de realizar los cálculos distribuidos.

3.5. Diagramas de interacción.

Los diagramas de interacción describen secuencias de intercambios de mensajes entre los roles que implementan el comportamiento de un sistema. Son utilizados para el modelado de los aspectos dinámicos de un sistema, proporcionan una vista integral de su comportamiento. Esto conlleva modelar instancias concretas, componentes y nodos junto con los mensajes enviados entre ellos que representan su interacción.

Los diagramas de interacción tienen dos formas de manifestarse:

- Diagramas de secuencia.
- Diagramas de colaboración.

Un diagrama de secuencia destaca la ordenación temporal de los mensajes, ilustra los objetos que se encuentran en un escenario, y la secuencia de mensajes intercambiados entre ellos para llevar a cabo la funcionalidad descrita; el diagrama de colaboración a su vez destaca la organización estructural de los objetos que envían y reciben mensajes.

A continuación se muestran los principales diagramas de interacción:

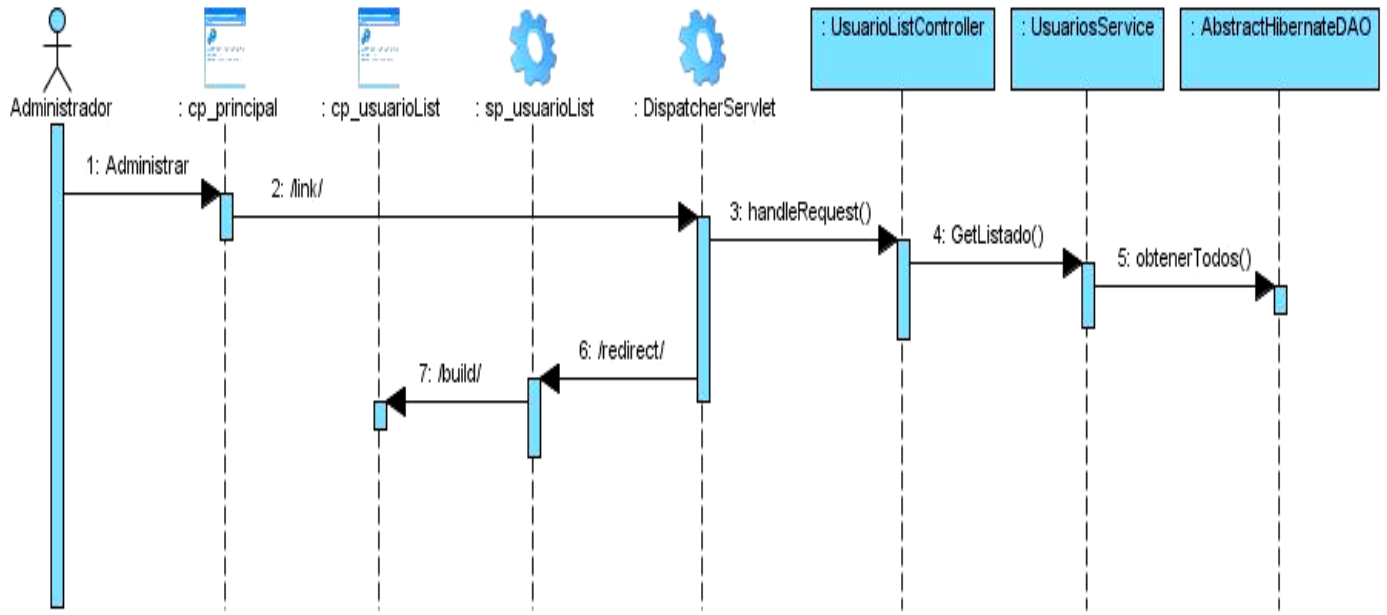

Figura 3.6. Diagrama de Secuencia: CU Gestionar Usuario.

En la Figura 3.6 se describe como el Administrador cuando desea administrar los usuarios en la página cliente *cp_principal* esta le envía la petición a la página servidor *DispatcherServlet*. Esta página accede a la clase *UsuarioListController*, la cual obtiene el listado de usuarios de la clase *UsuariosService*; esta clase accede a la clase de acceso a datos *AbstractHibernateDao*. Luego *DispatcherServlet* re-direcciona hacia la página servidor *sp_usuarioList*, que construye la página cliente *cp_usuarioList*.

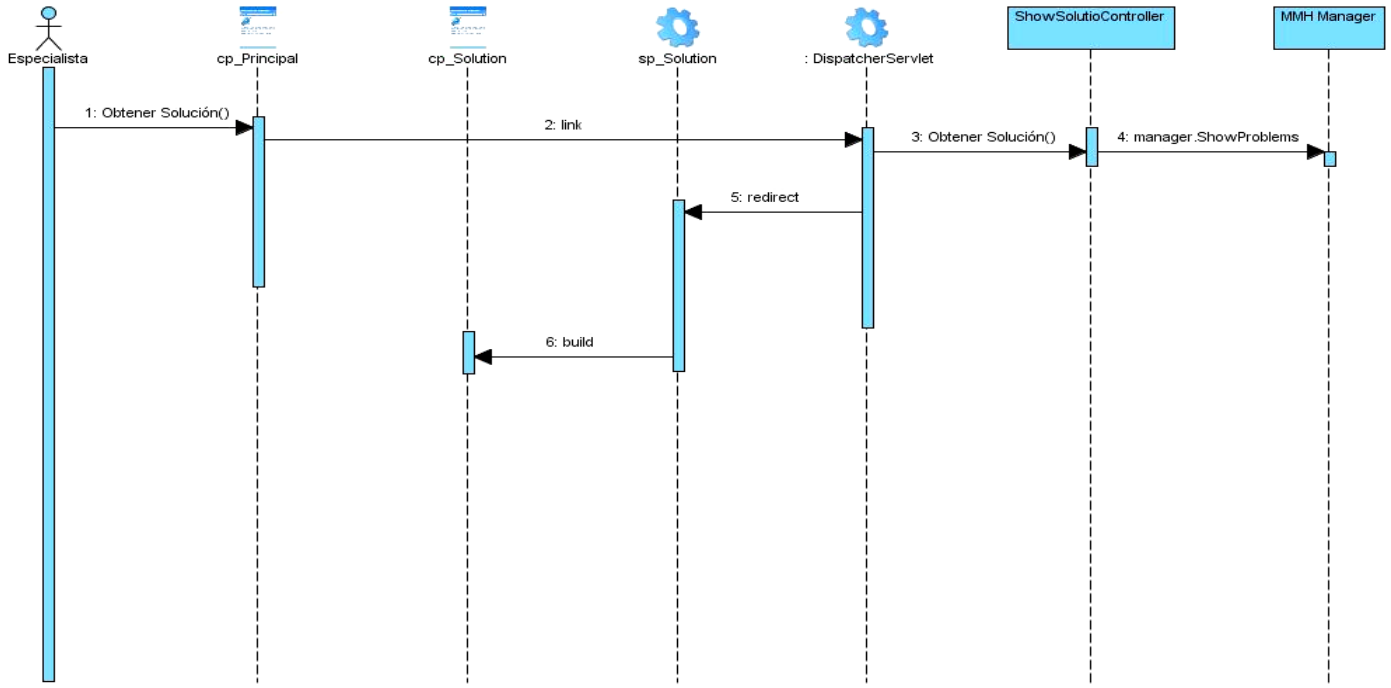

Figura 3.7. Diagrama de Secuencia: CU Obtener Solución.

En la Figura 3.7 se muestra como el Especialista, cuando desea obtener la solución en la página cliente *cp_Principal*, esta le envía la petición a la página servidor *DispatcherServlet*. Esta a su vez accede a la clase *ShowSolutionController* para obtener la solución, *ShowSolutionController* accede a la clase *MMHManager*. Después de obtener la solución *DispatcherServlet* re-direcciona hacia la página servidor *sp_Solution* la cual construye la página cliente *cp_Solution*.

3.6. Vista de Despliegue.

Esta vista constituye la visión física del sistema, representa un grafo de nodos unidos por conexiones de comunicación.

3.6.1 Diagrama de despliegue.

En el diagrama de despliegue se describe la arquitectura física del sistema durante la ejecución, en términos de procesadores, dispositivos y componentes de software. Describen la topología del sistema: la estructura de los elementos de hardware y el software que ejecuta cada uno de ellos.

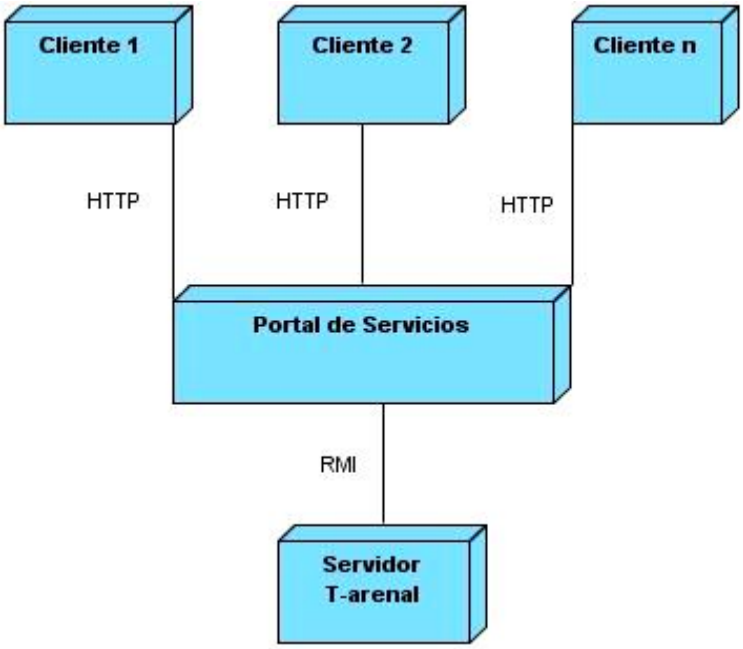

Figura 3.16. Diagrama de despliegue del sistema.

Descripción de los nodos.

- **Nodo Servidor T-arenal:** es el nodo donde reside el *Servidor T-arenal* de la Plataforma de Calculo Distribuido, el cual engloba el conjunto de recursos necesarios para realizar los cálculos distribuidos de la metodología MMH.
- **Nodo Portal de Servicios:** es el nodo donde reside el Portal de Servicios del sistema, el cual permite que los usuarios ejecutar los cálculos de la metodología MMH, así como bajar las soluciones.
- **Nodo cliente:** es el nodo donde reside el cliente del sistema. Pueden ser computadoras con características heterogéneas, con sistema operativo Linux o Windows.

Descripción de la comunicación.

HTTP o RMI: esta relación define una comunicación entre dos nodos cualesquiera que estos sea, que puede ser a través del protocolo RMI o HTTP. En el caso de la comunicación entre el Portal de Servicios y el servidor T-arenal, esta se hace solamente por RMI.

3.7. Conclusiones.

En este capítulo se puede concluir que: la arquitectura del sistema está sustentada en el estilo cliente – servidor, y en el MVC propuesto por el uso del framework Spring. Queda elaborado el sistema desde la estructura de paquetes de diseño hasta los diagramas de clases del diseño. Se muestran además los diagramas de interacción de los principales escenarios del sistema, así como la distribución física del mismo.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBAS

Implementación y pruebas del Sistema.

En este capítulo se describe la implementación del sistema en términos de componentes y la manera en que estos componentes serán desplegados. Se ilustran además algunas pruebas de caja negra realizadas al sistema para comprobar su correcto funcionamiento.

4.1. Diagrama de Componentes.

El diagrama de implementación describe como los elementos del modelo de diseño se implementan en términos de componentes. Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software, sean estos de código fuente, archivos, binarios, bibliotecas cargadas dinámicamente o ejecutables.

Para una mejor comprensión de la estructura de implementación de la aplicación Web, se muestra en la Figura 4.1 una vista de implementación con todos los paquetes y sus relaciones, su interacción con Spring, Hibernate y los ficheros de configuración utilizados. Se definen cada una de las capas del patrón MVC y las relaciones de dependencias entre los componentes que lo integran. El uso de interfaces minimiza el nivel de dependencia entre los componentes. Los ficheros de configuración definidos son de gran importancia ya que constituyen el centro de la inyección de dependencia vinculando cada uno de los componentes.

Seguidamente, se dividen en tres diagramas cada uno de los componentes generados que se corresponden a las capas definidas del patrón MVC.

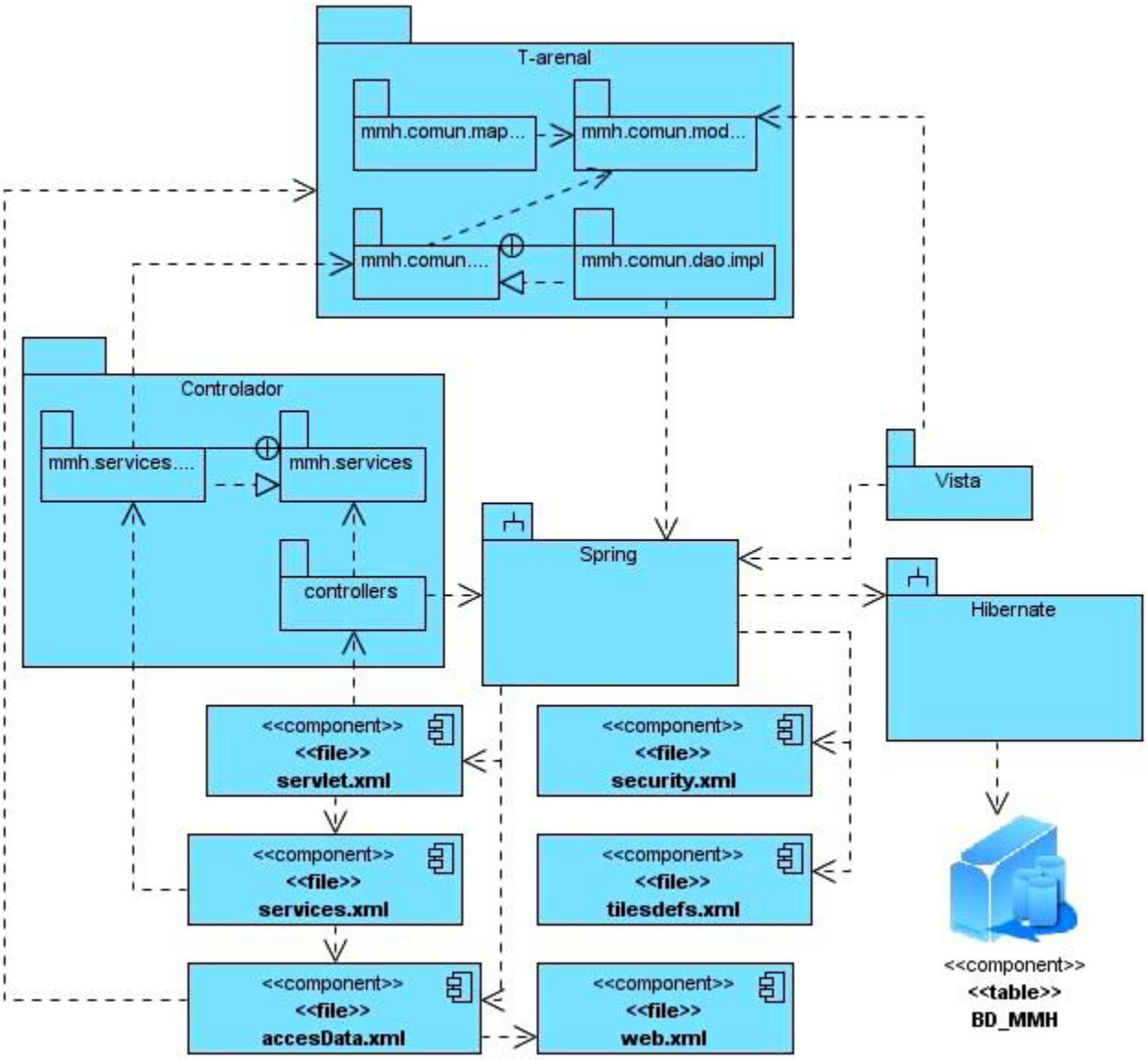

Figura 4.1. Diagrama de componentes del sistema.

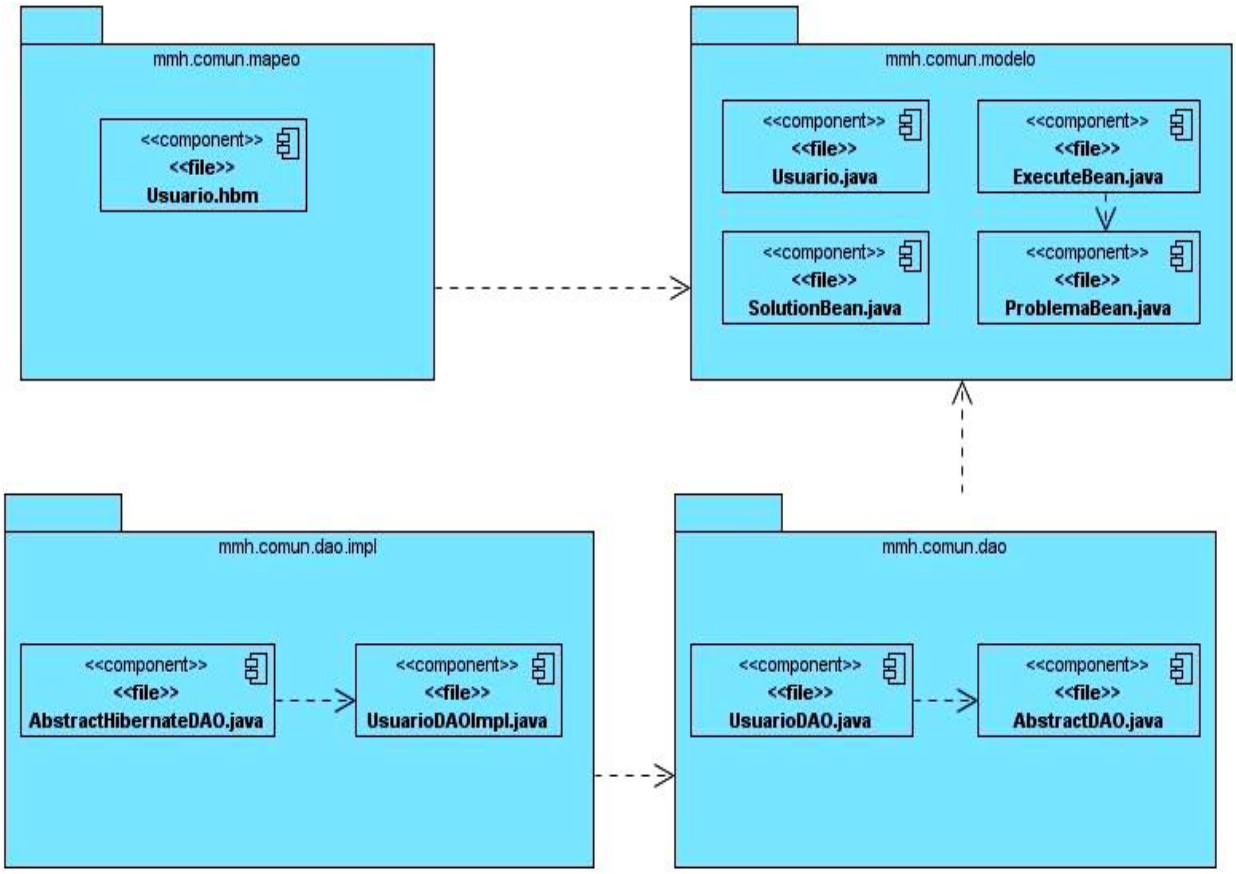

Figura 4.2. Modelo.

En la Figura 4.2 se muestran los componentes relevantes del Modelo. En el sistema *mmh.comun.mapeo* el componente ejecutable *Usuario.hbm* tiene su par *Usuario.java* en el sistema *mmh.comun.modelo* y en *el mmh.comun.dao*, el cual es utilizado para acceder al componente ejecutable *AbstractHibernateDAO.java*, que utiliza el componente ejecutable *UsuarioDAOImpl.java*.

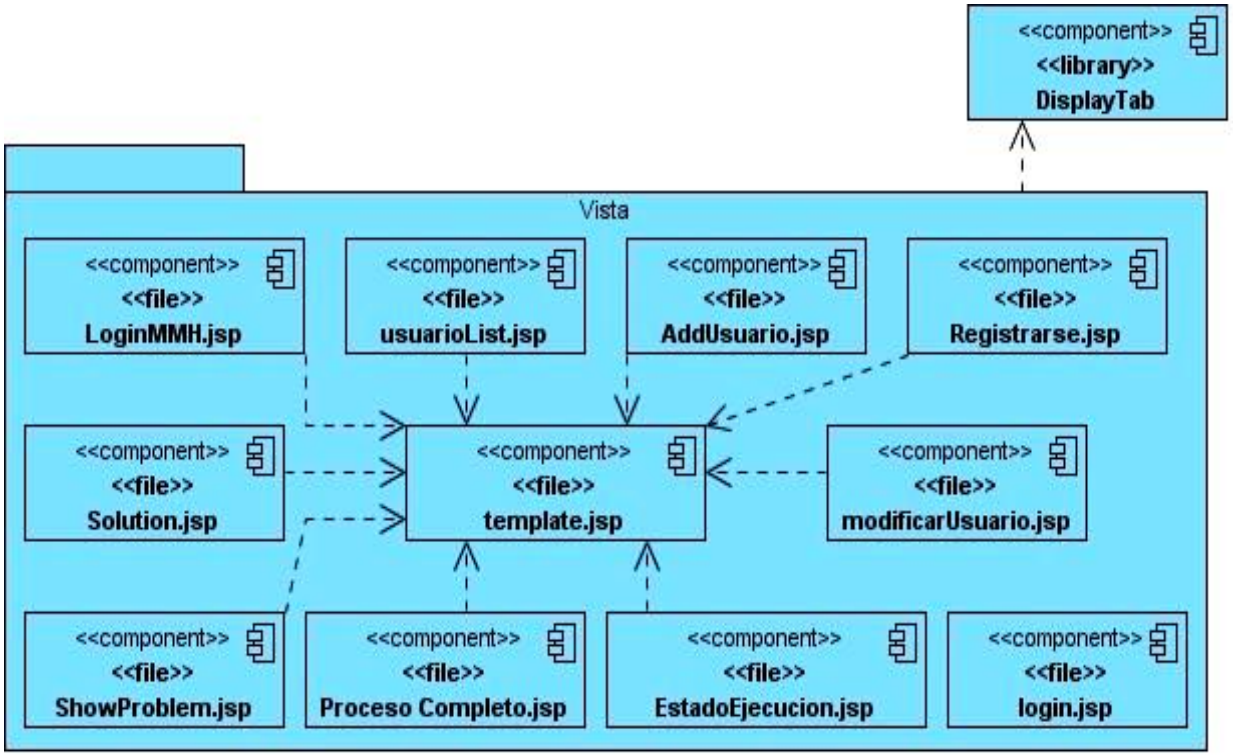

Figura 4.3. Vista.

En la Figura 4.3 se muestran las diferentes JSP (Java Server Pages) que conforman las interfaces de la aplicación Web. Todas ellas extienden la plantilla genérica *template.jsp*.

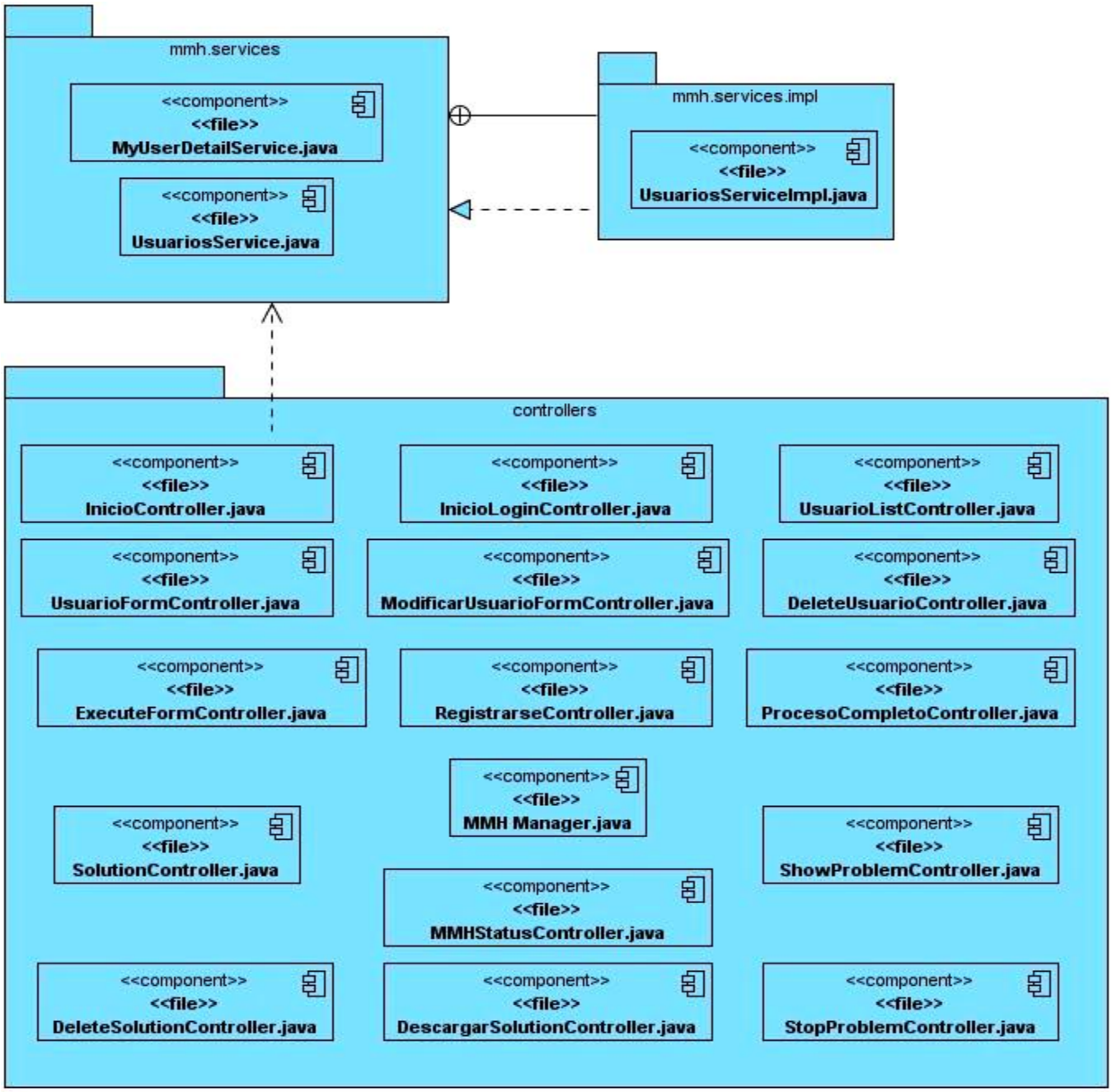

Figura 4.4. Controlador.

En la Figura 4.4 se muestra como el sistema *mmh.services* utiliza los controladores de la aplicación Web y es implementado por el componente de archivo *UsuariosServiceImpl.java* del sistema *mmh.services.impl*.

4.1.1. Diagrama de despliegue de componentes.

En este diagrama se muestra la distribución de los componentes por los distintos nodos del despliegue.

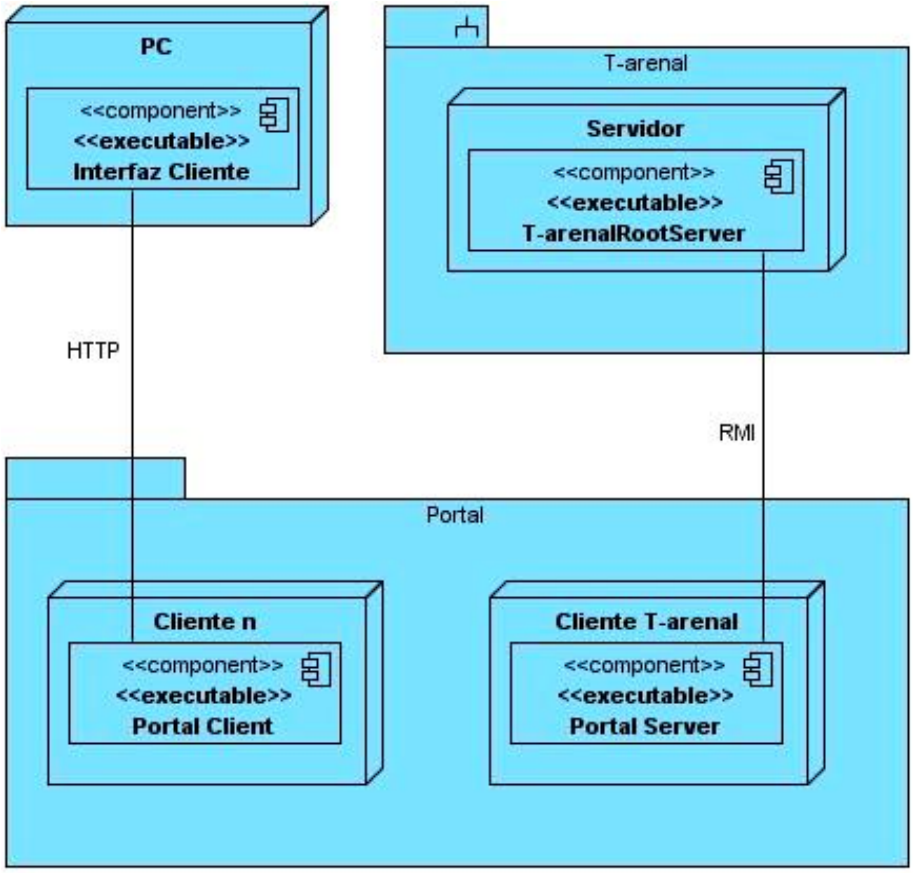

Figura 4.5. Ubicación de los componentes en los nodos del despliegue.

4.2. Pantallas de las aplicaciones. Resultados.

Las pantallas de la aplicación son imágenes del sistema en pleno funcionamiento. Las aplicaciones desarrolladas constituyen los principales resultados del trabajo. A continuación se muestran estas aplicaciones:

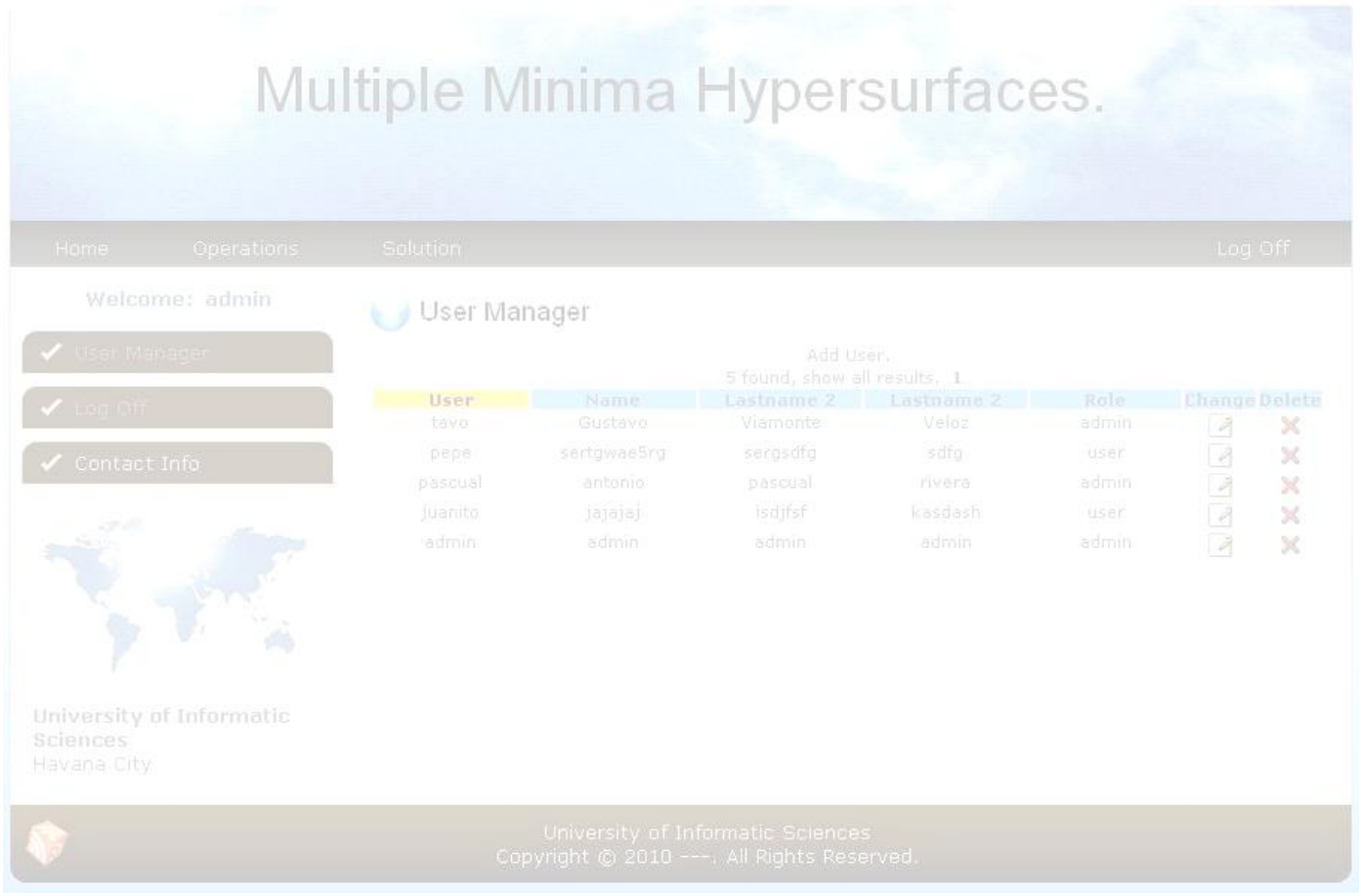

Figura 4.6. Interfaz para Gestionar Usuario.

En la Figura 4.6 se muestra la interfaz para gestionar usuario. La imagen muestra la lista de usuarios registrados, donde el Administrador del sistema controla a los usuarios modificando o eliminando sus datos.

Figura 4.7. Interfaz para Realizar Cálculos.

La imagen de la Figura 4.7 muestra la interfaz para realizar cálculos. En esta interfaz el Especialista puede insertar los datos necesarios para ejecutar la metodología MMH tales como: la cantidad de moléculas, las conformaciones, dimensión, temperatura, entre otras. Se puede insertar además el archivo del soluto, el cual tiene que ser un fichero de extensión car. Si todos los datos de entradas son correctos, entonces el sistema procede a la realización de los cálculos distribuidos.

Figura 4.8. Interfaz para Adicionar Usuario.

En la Figura 4.8 muestra la interfaz para adicionar un usuario. En esta interfaz el usuario entra sus datos como por ejemplo: nombre, apellidos y contraseña. Esto garantiza los niveles de seguridad necesarios para este tipo de aplicación.

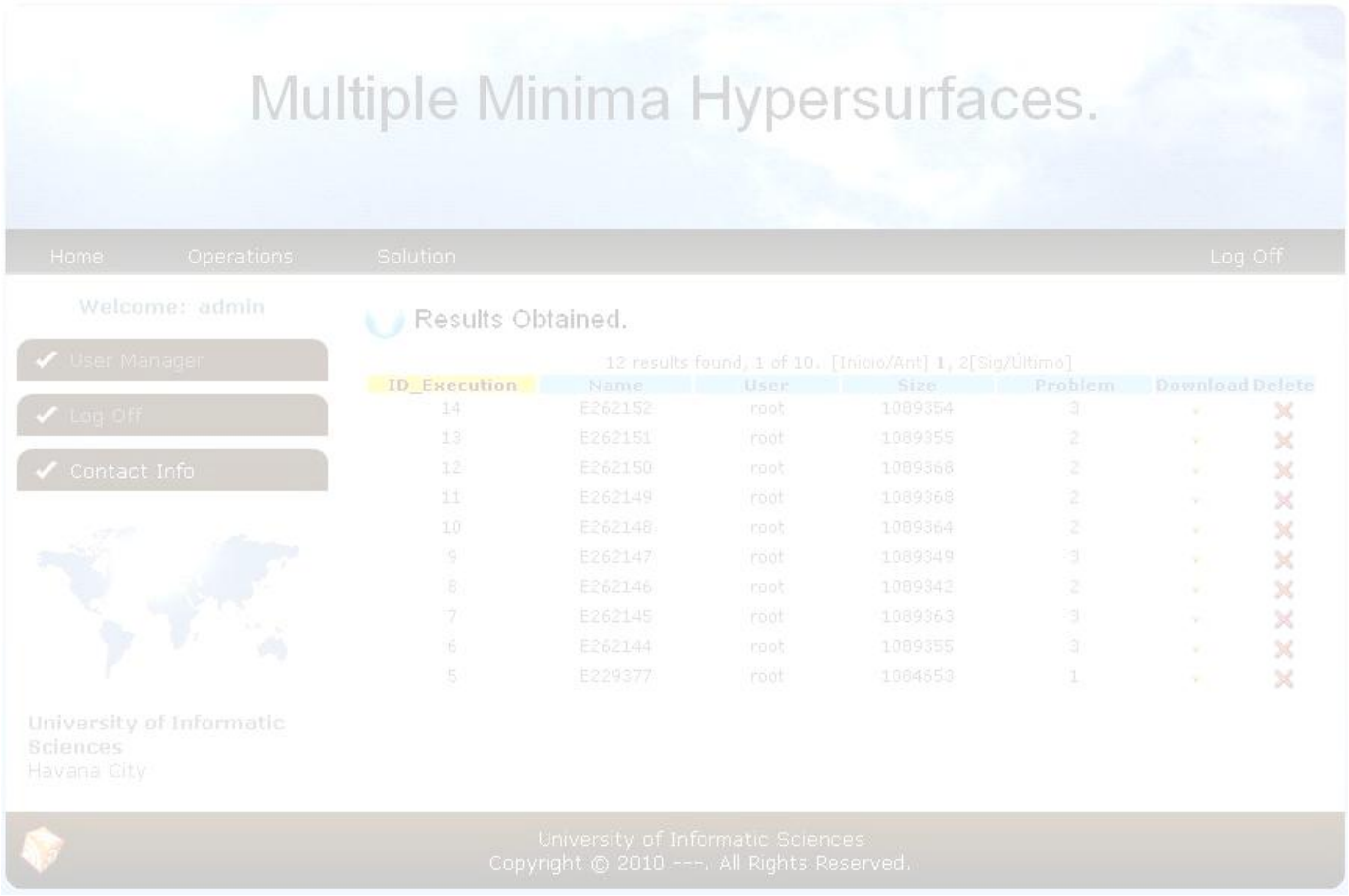

Figura 4.9 Interfaz para Obtener Solución.

La Figura 4.9 muestra la interfaz para obtener la o las soluciones. En esta interfaz el Especialista puede descargar la solución que desee, pero solo puede eliminar la o las soluciones que él haya calculado.

4.3. Modelo de Pruebas.

El principal objetivo del flujo de trabajo de pruebas es evaluar o valorar la calidad de la aplicación a través de la búsqueda y documentación de errores, validando el desempeño y cumplimiento de los requerimientos y dando una indicación de calidad.

La prueba es el proceso de ejecución de un programa con el propósito de descubrir errores. La prueba tiene éxito si descubre un error no detectado hasta entonces.

Las pruebas de caja negra son las que se llevan a cabo sobre la interfaz del software. Tienen como objetivo demostrar que las funciones software son operativas, que las entradas se aceptan de forma adecuada, y se produce un resultado correcto, y que la integridad de la información externa se mantiene.

Para validar las principales funcionalidades del sistema se utilizaron las Pruebas de Caja Negra.

4.3.1. Casos de pruebas.

Los principales escenarios de algunos de los casos de usos críticos fueron comprobados para detectar errores. En las siguientes secciones se representan estos casos de prueba por separado:

Caso de Prueba Autenticar Usuario.

Id del escenario	Escenario	Variable	Respuesta del Sistema	Resultado de la Prueba
CP 1.1	Cancelar autenticación.	-	La ventana se cierra totalmente.	Cierra la ventana de autenticación y no permite realizar otra acción.
CP 1.2	Usuario no válido	aprivera sd	El usuario no debe acceder a la aplicación y debe de mostrarse el mensaje: “Credenciales Incorrectas”.	El usuario no accede a la aplicación y se muestra el mensaje: “Credenciales Incorrectas”.

Caso de Prueba Autenticar Usuario – Autenticación Satisfactoria


Caso de Prueba Autenticar Usuario – Autenticación Fallida

Caso de Prueba Realizar Cálculos.

Id del escenario	Escenario	Variable	Respuesta del Sistema	Resultado de la Prueba
CP 2.1	Realizar cálculos sin llenar todos los campos necesarios de datos.	Cantidad de moléculas, temperatura, dimensión, conformaciones, ficheros soluto y disolvente entre otros.	El sistema debe mostrar un mensaje: “Llene todos los datos”.	No se realizan los cálculos y se muestra un mensaje: “Llene todos los datos”.

Caso de Prueba Descargar Solución.

Id del escenario	Escenario	Variable	Respuesta del Sistema	Resultado de la Prueba
CP 3.1	Descargar la solución.	-	El sistema debe mostrar un mensaje: “¿Desea obtener la solución?”. El usuario luego podrá obtener la solución en forma de un fichero compactado.	El usuario obtiene la solución en forma de un fichero compactado.

4.4. Conclusiones.

Se puede concluir del presente capítulo que: se obtuvieron los diagramas de componentes, el diagrama de despliegue de componentes del sistema, y se muestran algunas imágenes del sistema en pleno funcionamiento. Se muestran además, las pruebas de caja negra realizadas con algunos de los resultados obtenidos, que verifican el correcto funcionamiento de la interfaz de la aplicación.

CONCLUSIONES

- El estudio sobre los sistemas distribuidos y la computación Grid y de la Metodología MMH, permite afirmar que son propuestas ventajosas surgidas a partir del desarrollo de la informática, que actualmente y dadas las condiciones históricos – concretas de la ciencia y la técnica en el país, permiten implementarla para la solución de importantes problemáticas de la ciencia, una de las cuales se defiende en la presente tesis.
- A partir del estudio y profundización de la computación Grid y de la Metodología MMH, fue posible establecer las funcionalidades específicas y por consiguiente el diseño mismo del sistema, a partir de definir el estilo y el patrón de la arquitectura.
- El sistema se implementó en términos de componentes, validando sus funcionalidades a partir de pruebas de caja negra.
- El resultado obtenido constituye una aplicación Web que permite interactuar con la Plataforma de Cálculo Distribuido: T-arenal, de manera que usuarios foráneos puedan realizar cálculos distribuidos para la metodología MMH, utilizando los recursos computacionales de la universidad; cumpliendo así de forma satisfactoria los objetivos planteados para la investigación.

  - La implementación de la solución para el cálculo distribuido con T-arenal de la metodología MMH a través de la Grid, y la correspondiente validación de la solución desarrollada a partir de pruebas exploratorias para determinar su factibilidad, permiten concluir la posibilidad de su utilización.

RECOMENDACIONES

Para dar continuación a este trabajo se recomienda:

1. Desarrollar manuales de usuario para la aplicación desarrollada, incrementando la usabilidad del sistema y reduciendo la curva de aprendizaje de los usuarios.
2. Agregar la funcionalidad al sistema que permita la ejecución de la metodología MMH por partes.
3. Adaptar al sistema para ejecutar otros problemas distribuidos además de la metodología MMH.

BIBLIOGRAFÍA

[1] Fine-Grain Auth Keahey, K., Welch, V., Lang, S., Liu, B., Meder, S. Fine-Grain Authorization Policies in the GRID: Design and Implementation. *1st International Workshop on Middleware for Grid Computing,* 2003.

[2] S. Tanenbaum, Andrew. *Distributed Operating Systems.*

[3] Coulouris, George. *Sistemas Distribuidos.* Madrid: s.n., 2001.

[4] Santoro, Nicola. *DESIGN AND ANALYSIS OF DISTRIBUTED ALGORITHMS.* Ottawa, Canada: s.n., 2007.

[5] Foster, I., Kesselman, C., Nick, J. and Tuecke, S. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. Globus Project, 2002.

[6] A Java based heterogeneous distributed computing system. [En línea] [Citado el: 15 de diciembre de 2007.] http://distributed.cs.nuim.ie/.

[8] Hui Lin, Tseng, Haupt, Tomasz y C. Fox, Geoffrey. *Parallelizing MOPAC on distributed computing.* Syracuse, NY: s.n.

[9] James Rumbaugh IJ, Booch G. El Proceso Unificado de Desarrollo de Software. Addison Wesley; 1999.

[10] Microsoft Solution Framework; Disponible en: http://www.microsoft.com/technet/

itsolutions/msf/.

[11] eXtreme Programming; Disponible en: http://www.extremeprogramming.org/.

[12] Unified Modeling Language; Consultado Febrero 2009. Disponible en: http://www.uml.org/.

[13] James Rumbaugh IJ, Booch G. El Lenguaje Unificado de Modelado. Manual de Referencia. Segunda Edición. Addison Wesley; 2007.

[14] Magic Draw; Disponible en: http://www.magicdraw.com/.

[15] Rational Rose; Disponible en: http://www.ibm.com/software/rational.

[16] Umbrello: Disponible en: http://uml.sourceforge.net/.

[17] ArgoUML; Disponible en: http://argouml.tigris.org/.

[18] Suganuma T. Overview of the IBM Java Just-in-Time Compiler. IBM System. 2000; Disponible en: http://www.research.ibm.com/journal/sj/391/suganuma.html.

[19] Armstrong E. HotSpot: A new breed of virtual machine, Javaworld; Consultado Febrero 2009. Disponible en: http://www.javaworld.com/jw-03-1998/jw-03-hotspot.html.

[20] Zukowski J. Programación Java 2 J2SE 1.4. Vol. 1. SYBEX, Inc.; 2003.

[21] Netbeans; Consultado 2009. Disponible en: http://www.netbeans.org/.

[22] Ron Lynn. Programming Portlets: From JSR 168 to IBM WebSphere Portal Extensions. 2008.

[23] Buschmann F, et al. Pattern-Oriented Software Architecture. John Wiley & Sons; 1996.

[24] http: //mit.ocw.universia.net /6.170/6.170 /f01 /pdf/lecture-12.pdf.

[25] Hall, Marty y Brown, Larry. *Core Servlets and JavaServer Pages.* s.l. : Prentice Hall, 2004. ISBN: 3827266459.

[26] Storkel, Scott. ONJava.com. [En línea] O'Reilly Media, 12 de Noviembre de 2002. [Citado el: 12 de Enero de 2010.] http://onjava.com/pub/a/onjava/2002/12/11/eclipse.html.

[27] Chopra, Vivek, Li, Sing y Genender, Jeff. *Professional Apache Tomcat 6.* Indianapolis : Wiley Publishing, 2007. ISBN: 9780471753612.

[28] Tellez, Yuneimy. MMH en un entorno distribuido: Habana 2008.

ANEXOS

Descripción de los Casos de Uso

Caso de Uso	Gestionar usuario
Actores	Administrador
Resumen	El caso de uso inicia, cuando el administrador desea realizar alguna de las siguientes operaciones: -Adicionar usuario. -Modificar usuario. -Eliminar usuario. El sistema muestra la interfaz correspondiente según su solicitud y ejecuta las acciones necesarias. El caso de uso finaliza cuando se ejecuta alguna de las acciones solicitadas.
Precondiciones	El Administrador tiene que estar autenticado.
Referencias	RF10, RF11, RF12
Prioridad	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El administrador selecciona la opción “Administrar” del menú principal.	2 El sistema muestra una interfaz con el listado de los usuarios existentes y las opciones de: • “Adicionar Usuario”. • “Modificar”, para cada usuario. • “Eliminar”, para cada usuario.
3 El administrador selecciona una de las siguientes opciones: • Adicionar Usuario. • Modificar. • Eliminar.	4 El sistema, según la operación seleccionada por el administrador realiza una de las siguientes acciones: • Si el especialista selecciona la opción “Adicionar Usuario”, ir a la Sección “Adicionar

	usuario”. • Si el especialista selecciona la opción “Modificar” de uno de los usuarios listados, ir a la Sección “Modificar usuario”. • Si el especialista selecciona la opción “Eliminar” de uno de los usuarios listados, ir a la Sección “Eliminar usuario”.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3.1 Si el administrador no desea seleccionar ninguna opción, finaliza el caso de uso.	
Sección “Adicionar usuario”	
Acción del Actor	**Respuesta del Sistema**
	1 El sistema muestra un formulario, solicitando la información necesaria para adicionar un nuevo usuario. Los datos solicitados son los siguientes: a) Usuario. b) Nombre. c) Primer Apellido. d) Segundo Apellido. e) Contraseña. f) Rol.
2 El administrador entra los datos solicitados y presiona el botón “Adicionar”.	3 El sistema verifica que los campos no estén vacíos.
	4 El sistema verifica que el usuario no exista.
	5 El sistema adiciona el nuevo usuario y retorna a la actividad 2 del Flujo Normal de Eventos.

Flujos Alternos Sección “Adicionar usuario”	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el administrador presiona el botón “Cancelar”, ir a la actividad 2 del Flujo Normal de Eventos.	3.1 Si existe algún campo vacío, el sistema muestra un mensaje de error.
	4.1 Si existe el usuario, el sistema muestra un mensaje de error.
Sección “Modificar usuario”	
Acción del Actor	**Respuesta del Sistema**
	1 El sistema muestra un formulario, solicitando la información necesaria para modificar el usuario seleccionado. Los datos solicitados son los siguientes: a) Nombre. b) Primer Apellido. c) Segundo Apellido. d) Contraseña. e) Rol.
2 El administrador entra los datos solicitados y presiona el botón “Modificar”.	3 El sistema verifica que los campos no estén vacíos.
	4 El sistema modifica el usuario seleccionado y retorna a la actividad 2 del Flujo Normal de Eventos.
Flujos Alternos Sección “Modificar usuario”	
Acción del Actor	**Respuesta del Sistema**
2.1 Si el administrador presiona el botón “Cancelar”, ir a la actividad 2 del Flujo Normal de Eventos.	3.1 Si existe algún campo vacío, el sistema muestra un mensaje de error.

Sección “Eliminar usuario”	
Acción del Actor	**Respuesta del Sistema**
1 El especialista selecciona el usuario que desea eliminar.	2 El sistema muestra un mensaje de alerta solicitando confirmación.
3 El especialista confirma que desea eliminar el usuario.	4 El sistema elimina el usuario seleccionado por el especialista y retorna a la actividad 2 del Flujo Normal de Eventos.
Flujos Alternos Sección “Eliminar usuario”	
Acción del Actor	**Respuesta del Sistema**
3.1 El especialista confirma que no desea eliminar el usuario.	3.2 El sistema retorna a la actividad 2 del Flujo Normal de Eventos.
Poscondiciones	El usuario es insertado, eliminado o modificado.

Caso de Uso	Autenticar
Actores	Especialista
Resumen	El caso de uso inicia cuando el especialista introduce usuario y contraseña para acceder a la aplicación, estos datos son verificados por el sistema y en caso de ser correctos, se le otorga acceso al usuario en dependencia de los permisos que tenga habilitados su rol. El caso de uso termina cuando el usuario accede al sistema.
Precondiciones	-
Referencias	RF1
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El especialista introduce usuario y	2 El sistema verifica que el usuario exista y que

contraseña en el formulario de entrada.	la contraseña sea correcta.
	3 Si los datos son correctos, el sistema muestra la interfaz principal de la aplicación, donde aparecen habilitadas las funcionalidades según el privilegio que tenga. Finaliza el caso de uso.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	3.1 Si los datos no son correctos, el sistema muestra un mensaje de error, permitiéndole al especialista entrar nuevamente usuario y contraseña.
Poscondiciones	El especialista queda autenticado.

Caso de Uso	Realizar Cálculos
Actores	Especialista
Resumen	El caso de uso inicia cuando el Especialista selecciona la opción “Realizar Cálculos” del menú principal y el sistema muestra un formulario para que este inserte los datos necesarios para la ejecución del problema. El caso de uso finaliza cuando se han enviado todos los datos correctamente y comienza la ejecución.
Precondiciones	El Especialista tiene que estar autenticado.
Referencias	-
Prioridad	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El especialista accede a la opción	2 El sistema muestra un formulario con los

"Realizar Cálculos" del menú principal.	siguientes campos de entrada de datos: • Título 1 • Energía-Referencia • Cantidad-Moléculas • Temperatura • Energía-Convergencia • Nivel-Semejanza • Número-Limitaciones • Dimensión • Conformaciones • Número-Solventes • Título2 • Fracms
3 El especialista procede a llenar los campos y selecciona el botón "Enviar".	4 El sistema establece una conexión con el servidor central T-arenal y le envía los datos para comenzar la ejecución del problema.
	5 El sistema muestra el mensaje "Los datos han sido enviados satisfactoriamente, su problema se está ejecutando".
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
3.1 El especialista llena los campos nuevamente y selecciona el botón "Enviar".	4.1 Si existe algún campo vacío el sistema muestra el mensaje "Debe llenar todos los campos."; si existe algún campo con entrada de datos incorrectos, el sistema muestra el mensaje "Error en la entrada de datos."
Poscondiciones	-

Caso de Uso	Obtener Solución.
Actores	Especialista.
Resumen	El caso de uso inicia cuando el Especialista selecciona la opción “Obtener Solución” en el menú principal. El sistema muestra una vista que contiene las soluciones obtenidas. El usuario tiene la posibilidad de descargar o eliminar la solución. El caso de uso termina luego de realizadas estas acciones.
Precondiciones	El especialista debe estar autenticado.
Referencias	-
Prioridad	Crítico

Sección “Obtener Solución”	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona la opción “Obtener Solución” en el menú principal.	2 El sistema establece una conexión con el servidor central T-arenal y muestra una vista con las soluciones obtenidas.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1 Si no hay soluciones el sistema muestra un mensaje “no hay soluciones disponibles”.
Sección “Descargar solución”	
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona una solución y accede a la opción “Descargar”.	2 El sistema establece una conexión con el servidor central T-arenal y realiza la transferencia de archivos correspondientes a la solución

	seleccionada.
Pos condiciones	-
Sección “Eliminar Solución”	
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona la solución y accede a la opción “Eliminar Solución”.	2 El sistema establece una conexión con el servidor central T-arenal y elimina la solución seleccionada.
Pos condiciones	-

Caso de Uso	Mostrar Ejecuciones.
Actores	Especialista.
Resumen	El caso de uso inicia cuando el Especialista selecciona la opción “Mostrar Ejecuciones” en el menú principal. El sistema muestra una vista que contiene todas las ejecuciones que se producen en ese momento. El usuario tiene la posibilidad de detener una ejecución o mostrar el estado de una ejecución. El caso de uso termina luego de realizadas estas acciones.
Precondiciones	El especialista debe estar autenticado.
Referencias	CU Mostrar Estado.
Prioridad	Crítico
Sección “Mostrar Ejecuciones”	
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona la opción “Mostrar Ejecuciones”.	2 El sistema establece una conexión con el servidor central T-arenal y muestra una lista de

	todas las ejecuciones.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	2.1 En caso de no existir ejecuciones el sistema muestra un mensaje “No hay ejecuciones”.
Sección “Detener Ejecución”	
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona la opción “Detener Ejecución”.	2 El sistema establece una conexión con el servidor central T-arenal y detiene la ejecución.
	3 El sistema muestra la lista actualizada.
Poscondiciones	-

Caso de Uso	Mostrar Estado.
Actores	Especialista.
Resumen	El caso de uso inicia cuando el Especialista selecciona la opción “Mostrar Estado” en la vista del caso de uso “Mostar Ejecuciones”. El sistema muestra una vista que contiene el estado de la ejecución seleccionada. El usuario tiene la posibilidad de actualizar dicho estado. El caso de uso termina luego de realizadas estas acciones.
Precondiciones	El especialista debe estar autenticado.
Referencias	-
Prioridad	Crítico
Sección “Mostrar Estado”	
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1 El actor selecciona una ejecución y accede a la opción “Mostrar Estado”.	2 El sistema establece una conexión con el servidor central T-arenal y muestra una vista con el estado de la ejecución seleccionada.
Sección “Actualizar Estado”	
Flujo Normal de los Eventos	
Acción del Actor	**Respuesta del Sistema**
1 El actor selecciona la opción “Actualizar”.	2 El sistema establece una conexión con el servidor central T-arenal y muestra el nuevo estado de la ejecución.
Pos condiciones	-

GLOSARIO DE TÉRMINOS

***Mainframe*:** computadora grande, potente y costosa usada principalmente por una compañía para el procesamiento de una gran cantidad de datos.

***GRID*:** infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Su propósito es facilitar la integración de recursos computacionales.

***Framework*:** estructura de soporte definida, en la cual un proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir un soporte de programas, bibliotecas y un lenguaje de scripting, para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

***T-arenal*:** Plataforma de Tareas Distribuidas.

***Portlets*:** componente web gestionado por un contenedor que tras la petición de un usuario genera y presenta contenidos dinámicos personalizables que se insertan en el interfaz de usuario del portal como componentes de contenido.

SSL: Protocolo de Capa de Conexión Segura y Transport Layer Security -Seguridad de la Capa de Transporte- (TLS), su sucesor, son protocolos criptográficos que proporcionan comunicaciones seguras por una red, comúnmente Internet.

Debugger: depurador, es un programa que permite depurar o limpiar los errores de otro programa informático.

JUnit: es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck que son utilizadas en programación para hacer pruebas unitarias de aplicaciones Java.

Plug-in: aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API.

***GRIDSPHERE*:** herramienta open-source para crear portales. Permite a los desarrolladores generar y empaquetar rápidamente aplicaciones basadas en web portlets.

***Web Services*:** conjunto de protocolos y estándares que sirven para intercambiar datos entre aplicaciones.

***Portal*:** sitio web cuya característica fundamental es la de servir de *Puerta de entrada* (única) para ofrecer al usuario, de forma fácil e integrada, el acceso a una serie de recursos y de servicios relacionados a un mismo tema. Incluye: enlaces, buscadores, foros, documentos, aplicaciones, compra electrónica, etc.

***XP*:** eXtreme Programming. Metodología de desarrollo de software basada en valores como simplicidad, comunicación, retroalimentación y coraje.

***RUP*:** Rational Unified Process. Proceso de desarrollo de software. Constituye la metodología estándar más utilizada para el análisis, implementación y documentación de sistemas orientados a objetos.

***IDE*:** Entorno de desarrollo integrado.

***TCP/IP*:** Protocolo de Control de Transmisión. Es uno de los protocolos fundamentales en internet. Fue creado entre los años 1973 y 1974 por Vint Cerf y Robert Kahn.

***NetBeans*:** plataforma que permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos.

***HTTP*:** Hypertext Transfer Protocol (Protocolo de Transferencia de Hipertexto). Es el protocolo usado en cada transacción de la World Wide Web.

***Java*:** lenguaje de programación orientado a objetos desarrollado por Sun Microsystems.

***API*:** interfaz de programación de aplicaciones. Es un conjunto de rutinas que hacen funciones como crear ventanas y desplegar varios gráficos.

***MVC*:** Modelo, vista, controlador. Patrón de Arquitectura.

***FTP*:** File Transfer Protocol, por sus siglas en inglés. Protocolo de transferencia de ficheros entre sistemas conectados a una red TCP, basado en la arquitectura cliente-servidor.