UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

TÍTULO: IMPLEMENTACIÓN DISTRIBUIDA DEL GAUSSIAN SOBRE LA PLATAFORMA JDCS

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas.

AUTORAS:

YAIMA COBAS MORA.

AYLIN ALMAGUER GUERRERO.

TUTORES:

LIC. LIESNER ACEVEDO MARTÍNEZ.

LIC. RAFAEL ARTURO TRUJILLO RASÚA.

CIUDAD DE LA HABANA, CUBA

JUNIO 2008

"Si tuviera el privilegio de vivir otra vez mi propia vida, muchas cosas las haría diferente de como las hice hasta hoy, pero puedo a la vez asegurar, que toda mi vida lucharía con idéntica pasión por los mismos objetivos por los que he luchado hasta hoy".

Fidel Castro Ruz.

DECLARACIÓN DE AUTORÍA

Declaramos ser autoras 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 *27* días del mes de *Junio* del año *2008*.

_____ _____

Yaima Cobas Mora (Autora) Aylin Almaguer Guerrero (Autora)

_____ _____

Lic. Liesner Acevedo Martínez (Tutor) Lic. Rafael Arturo Trujillo Rasúa (Tutor)

DATOS DE CONTACTO

Autoras

Yaima Cobas Mora
E-mail: ycobas@estudiantes.uci.cu

Aylin Almaguer Guerrero
E-mail: aalmaguer@estudiantes.uci.cu

Tutores

Liesner Acevedo Martínez
E-mail: frodo@uci.cu

Rafael Arturo Trujillo Rasúa
E-mail: trujillo@uci.cu

AGRADECIMIENTOS

A todas aquellas personas que de una forma u otra han contribuido a hacer realidad este sueño.

A nuestro Comandante en Jefe Fidel Castro Ruz por ser el motor impulsor de la creación de nuestra Universidad de las Ciencias Informáticas.

A nuestros padres por toda su dedicación.

A nuestros tutores por siempre estar ahí en los momentos que los necesitamos, por su comprensión, apoyo y experiencia brindada.

A Noel, Aurelio y Longendri por soportar nuestra incansable insistencia y estar siempre dispuestos a colaborar.

A los muchachos del proyecto Randy, Abel, Héctor, Edel, César y en especial a Luisito por sacarnos de tantos aprietos.

Yayi y Aylin

DEDICATORIA

A mi mamita Mireya y a mi papi Jorge por ser las personas mas importantes en mi vida, por ser la fuerza mayor que me ha impulsado a luchar hasta por lo imposible para tratar de ser cada día un poquito mejor, porque todo lo que he sido y lo que soy se lo debo a ustedes. Por su apoyo incondicional, por su amor, su paciencia, su dedicación, por hacerme sentir la persona más afortunada de este mundo solo por el hecho de tenerlos junto a mí. Los quiero más que a mi propia vida, sepan que por ustedes y para ustedes vivo.

A Geyser por darme tanto amor y llenarme de detalles que me hacen sentir cada día que serás My Endless Love. Gracias por saber sobrellevar a una chiquilla malcriada y caprichosa durante tanto tiempo, gracias por todos los momentos lindos que hemos vivido juntos, gracias por estar ahí siempre que te necesito más que como el amor de mi vida como un compañero, dispuesto a escuchar y a dar todo de ti solo por verme feliz, gracias por amarme tanto, gracias por existir. Te amo mi bebé.

A mi hermanito y a Melvis por su cariño y apoyo.

A mis cuatro abuelitos por quererme tanto y ser mis tesoros más preciados.

A toda mi familia, sé que me adoran, yo también a ustedes.

A mi tía Yayi y a mi tío Juan Carlos que aunque no están presentes físicamente los llevo siempre conmigo y estoy segura de que se sentirían muy orgullosos de mí. Donde quiera que estén sepan que los quiero y los extraño mucho.

A Yayi, mi compañera de tesis, gracias por tu amistad, dedicación y esfuerzo realizado para hacer realidad este sueño.

A todas las niñas del apartamento Jen, Mar, Evy, Rosy, Ide, Zoily, Kenia, Yane y Yune por ser chicas tan especiales que han sabido ganarse un ladito muy importante en mi corazón. Nunca las olvidaré.

Aylin

Dedicatoria

Lo genial de vivir es que hay personas que te llenan la vida, que te acompañan a enfrentarte a todos los tiempos, que están ahí siempre que los necesitas, que te dan oportunidades, que te enseñan… A todas esas personas que han tenido un segundo de luz para mí quiero darle las gracias.

En especial a mi mamita querida Berena Mora Zaldívar por su amor infinito, dedicación, confianza, por ser mi mejor amiga, por comprenderme y apoyarme siempre y darme los consejos precisos en el momento preciso, por guiarme con mucho empeño por el camino correcto, por ayudarme a crecer y llegar hasta donde estoy, por ser la persona más linda, dulce y tierna que jamás conocí, TE AMO MAMI ¡todo lo que soy te lo debo a ti!

A Pedrito por ser mi padre durante todos estos años, por estar siempre ahí cuando lo he necesitado, por preocuparte por mí, por darme tanto cariño y acogerme como tu hija mayor.

A mi hermanita Yaily por quererme tanto, por ser la niña de mi corazón y por convertirme en la hermana mayor de la que habla con tanto orgullo, te adoro tata, espero no defraudarte nunca.

A mi bebé: Argenis Dibur Faxas, mi gran amor, gracias por haberme dedicado 2 años maravillosos en esta universidad, por colmarme de afectos y amarme tanto. Discúlpame haberlo echado a perder. A toda tu familia.

A mi abuelita Rosa, que aunque ya no está entre nosotros siempre confió en mi. A Anita. A mis tías María y Vergelia por ser mis segundas madres y a mi primita Mardelis por sus sabios consejos.

A mis hermanitos Rey y Ale que siempre han estado orgullosos de mí, por todo su apoyo y cariño.

A mis compañeras de apto y amigas de estos 5 años: Yane, Jen, Rosy, Mar, Ide, Evy, Yune, Zoila, Keku, y Lorna. A Elsia, Violeta, Coty y Dayaxi.

A mi amiga y compañera de tesis Aylin, gracias por soportar mis ataques durante todo este tiempo, te quiero mimi.

A Yuly, mi amiga de todos los tiempos.

A Infodanz, en especial a Silvia, Isabelita, Belkis y Rassy.

A todos los familiares y amigos que me quedan sin mencionar.

Yayi

RESUMEN

En la actualidad dentro del ámbito de las ciencias informáticas, se trabaja día a día para garantizar la máxima productividad y gestionar de forma eficiente los recursos. La Facultad 6, de la Universidad de las Ciencias Informáticas, a la cual pertenece el polo de Bioinformática, está desarrollando el proyecto BioGrid, que prevé la tarea de realizar cálculos químico-teóricos a un número considerable de compuestos orgánicos. Estos cálculos son de elevado costo computacional por lo que surge a su vez la idea de diseñar e implementar un sistema capaz de realizarlos de forma distribuida. El objetivo principal de este trabajo es analizar, diseñar e implementar un sistema para realizar cálculos químico-teóricos distribuidos utilizando Gaussian98 para Linux, con el fin de lograr reducir los tiempos de respuesta teniendo en cuenta la cantidad de recursos disponibles en una red en el momento en que se envíe el pedido y la magnitud de este. La aplicación se desarrollará en un ambiente multiplataforma, utilizando el lenguaje Java para su implementación, siguiendo la línea de utilización de herramientas libres para garantizar el beneficio gratuito de la misma.

PALABRAS CLAVES: Sistemas Distribuidos, RMI, Cálculos químicos-teóricos, Gaussian.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Con el avance de las ciencias y las tecnologías en el mundo actual, han surgido nuevos problemas en el ámbito de las investigaciones científicas, lo cual requiere el uso de métodos computacionales que proporcionen posibles vías de solución basadas en la calidad, rapidez y eficiencia. Este tema es tratado en Cuba como uno de los principales retos en el marco de la Bioinformática. Existen software o paquetes especializados en cálculos químico-teóricos tales como el Vega, Mopac, Gaussian, NWChem entre otros que cuando se necesita procesar un gran número de moléculas resultan muy costosos computacionalmente, dadas las características de éstos software se hace difícil para los centros investigativos que lo utilizan tener los resultados exactos en el tiempo que se necesitan. Como vía de solución a los problemas de obtención de resultados en un tiempo prudente así como la exactitud de los mismos se han venido aplicando los sistemas computacionales distribuidos y los sistemas de procesamiento en paralelo, los cuales están constituidos por varias computadoras que pueden compartir recursos tanto de software como hardware y ejecutar procesos de forma concurrente. La utilización de estos sistemas aumenta las potencialidades de cómputo de cualquier institución, al no estar sujetos a las restricciones de una máquina, lo que hace posible utilizar y explotar los recursos de toda una red.

La Universidad de las Ciencias Informáticas dedica gran parte de su esfuerzo al desarrollo de proyectos productivos vinculados a diferentes esferas de la economía del país. Específicamente la Facultad 6 cuyo perfil de producción es la Bioinformática se dedica entre otras cosas a desarrollar el montaje de aplicaciones sobre la plataforma Java de Cómputo Distribuido (JDCS) para tratar de reducir los costos y los tiempos de respuestas de los procesos de investigación. Aprovechando esta posibilidad y estimando que si se distribuyen los cálculos químico – teóricos sobre una red de computadoras se obtendrían los resultados en un menor tiempo, se decide realizar la implementación distribuida del Gaussian98 sobre dicha plataforma con el objetivo dar solución al **problema científico** de cómo reducir los tiempos de respuesta de la aplicación Gaussian durante la realización de cálculos químico-teóricos ante grandes volúmenes de datos. Por esta razón se define como **objeto de estudio** del presente trabajo: los sistemas distribuidos para el procesamiento y cálculo masivo de información enmarcado en el **campo de acción**: Sistema Java de Cómputo Distribuido (JDCS). Por tanto se define como **objetivo general** de la investigación: Desarrollar un sistema distribuido que permita reducir los tiempos de respuesta de la aplicación Gaussian98.

Para dar cumplimiento a dicho objetivo es necesario alcanzar los siguientes **objetivos específicos**:

- Capturar requisitos del sistema a implementar
- Diseñar el sistema.
- Implementar el sistema.
- Evaluar la eficiencia del sistema implementado.

Para alcanzar dichos objetivos se planteó desarrollar las siguientes **tareas de investigación:**

- Análisis del estado del arte de los sistemas distribuidos.
- Estudio de la tecnología Java RMI (Invocación a Métodos Remotos).
- Estudio de la plataforma JDCS para el trabajo distribuido.
- Captura de requisitos.
- Diseño del modelo de clases.
- Implementación del sistema.
- Realización de pruebas al sistema y evaluación de la calidad y eficiencia del mismo.

El presente documento se estructura en un Resumen, Introducción, varios capítulos que constituyen el cuerpo de la tesis, Conclusiones, Recomendaciones, Bibliografía y Referencias bibliográficas. Los capítulos son:

Capítulo 1: Estudios Preliminares.
En este capitulo se hará un estudio de los Sistemas Distribuidos, sus características, ventajas y desventajas, aplicaciones que pueden tener en la Bioinformática, justificando así su utilización para dar respuesta al problema planteado con anterioridad. Además se hará una breve descripción del Software en cuestión: Gaussian y se analizarán algunas de las herramientas y metodologías que pueden ser de utilizadas a la hora de desarrollar del sistema. .

Capítulo 2: Materiales y Métodos.
En el presente capitulo se aborda la metodología a seguir para el desarrollo del sistema propuesto así como las diferentes herramientas que se utilizarán para llevar a cabo el desarrollo del sistema, el lenguaje de programación y el entorno integrado de desarrollo para la implementación del mismo.

Capítulo 3: Desarrollo y Resultados.

En este capítulo se expondrán los requisitos funcionales y no funcionales del sistema a implementar, los casos de uso, el diagrama de casos de uso y su descripción, los de clases del diseño realizados así como los diagramas de secuencia, de componentes y despliegue. Además se expondrá la solución y se mostrarán los resultados alcanzados al realizar las pruebas al sistema.

CAPÍTULO 1
ESTUDIOS PRELIMINARES

En este capítulo se describen las características, ventajas, desventajas y aplicaciones de los Sistemas Distribuidos en la Bioinformática, justificando de esta forma por qué fueron escogidos como una posible vía para dar solución a los problemas de cálculos químico - teóricos ante grandes volúmenes de datos. También se describen las características del Gaussian98 y brevemente algunas Herramientas y Metodologías que pueden ser utilizadas durante el desarrollo del sistema.

1.1 Sistemas Distribuidos

Un sistema distribuido se define como un conjunto de computadoras que están unidas por una gran red de alta velocidad, estas computadoras son capaces de trabajar todas en la realización de una tarea común.

Los sistemas distribuidos pueden ser implementados, desde unas pocas estaciones de trabajo conectadas por una red de área local, hasta Internet que constituye una colección de redes de área local y de área extensa interconectadas, que enlazan millones de computadoras.[1]

Las aplicaciones distribuidas son utilizadas tanto por usuarios comunes como por empresas, universidades o centros investigativos de alto nivel por lo que deben estar caracterizadas por la fiabilidad, la seguridad y privacidad en el sistema. Deben permitir el acceso de varios usuarios al mismo tiempo a una información determinada, garantizar que los tiempos de respuesta sean asequibles, además deben estar abiertas al crecimiento del sistema y a la integración al mismo de otros que sean usados por compañías o usuarios.

1.1.1 Características de los Sistemas Distribuidos

- *Concurrencia:* Permite la utilización simultánea de los recursos disponibles por los usuarios conectados a la red.

Los sistemas distribuidos están constituidos por muchas computadoras, cada una con uno o más procesadores centrales. Es decir, si hay M computadoras en un sistema distribuido con un procesador central cada una, entonces pueden estar ejecutándose en paralelo hasta M procesos.

En un sistema distribuido basado en el modelo de compartición de recursos, la posibilidad de ejecución paralela ocurre por dos razones:

- ✓ Muchos usuarios interactúan simultáneamente con programas de aplicación.
- ✓ Muchos procesos servidores se ejecutan concurrentemente, cada uno respondiendo a diferentes peticiones de los procesos clientes.

El primer caso es menos conflictivo, ya que normalmente las aplicaciones de interacción se ejecutan aisladamente en la estación de trabajo del usuario y no entran en conflicto con las aplicaciones ejecutadas en las estaciones de trabajo de otros usuarios.

El segundo caso surge debido a la existencia de uno o más procesos servidores para cada tipo de recurso. Estos procesos se ejecutan en distintas máquinas, de manera que se están ejecutando en paralelo diversos servidores, junto con diversos programas de aplicación. Las peticiones realizadas por los clientes a los recursos de un servidor pueden ser puestas en espera para ser procesadas paulatinamente o pueden ser procesadas varias al mismo tiempo. Cuando esto ocurre los procesos servidores deben sincronizar sus acciones para asegurarse de que no existen conflictos. La sincronización debe ser cuidadosamente planeada para asegurar que no se pierden los beneficios de la concurrencia.

- *Carencia de reloj global:* No se utiliza un temporizador general en las coordinaciones para el envío de mensajes.
- *Compartición de recursos:* Componentes hardware como discos e impresoras hasta elementos software como ficheros, ventanas, bases de datos y otros objetos. [1]

Los usuarios de computadoras personales dentro de un sistema distribuido no obtienen automáticamente los beneficios de la compartición de recursos. Los recursos en un sistema distribuido están físicamente encapsulados en una de las computadoras y sólo pueden ser accedidos por otras computadoras mediante la red. Para que la compartición de recursos sea efectiva, ésta debe ser manejada por un programa que ofrezca una interfaz de comunicación permitiendo el acceso, manipulación y actualización de una manera fiable y consistente, esto es lo que recibe el nombre de gestor de recursos que no son mas que un módulo software que maneja un conjunto de elementos de un tipo en particular [1]. Cada uno de estos tipos requiere algunas políticas y métodos específicos junto con requisitos comunes para todos ellos. Estos requisitos incluyen la provisión de un esquema de nombres para cada clase de recurso, deben permitir que estos sean accedidos desde cualquier localización; así como realizar la traslación de sus nombres

a direcciones de comunicación y la coordinación de los accesos concurrentes que cambian el estado de los recursos compartidos para mantener la consistencia.

Un sistema distribuido puede verse como un conjunto de gestores de recursos y un conjunto de programas que los utilizan. Los usuarios se comunican con los gestores de recursos para acceder a los recursos compartidos del sistema. Esta perspectiva nos lleva a dos modelos de sistemas distribuidos: el modelo cliente-servidor y el modelo basado en objetos.

- *Apertura (Openness):* La apertura de los sistemas distribuidos se determina por el grado hacia el que nuevos servicios de compartición de recursos se pueden añadir sin perjudicar ni duplicar a los ya existentes.

Básicamente los sistemas distribuidos cumplen una serie de características:

- ✓ Las interfaces software claves del sistema están claramente especificadas y se ponen a disposición de los desarrolladores. En resumen, las interfaces se hacen públicas.
- ✓ Los sistemas distribuidos abiertos se basan en la provisión de un mecanismo uniforme de comunicación entre procesos e interfaces publicados para acceder a recursos compartidos.
- ✓ Los sistemas distribuidos abiertos pueden construirse a partir de hardware y software heterogéneo. Pero la conformidad de cada componente con el estándar publicado debe ser cuidadosamente comprobada y certificada si se quiere evitar tener problemas de integración.

- *Escalabilidad:* Cuando el tamaño y complejidad de las redes de ordenadores crece, el sistema distribuido seguirá siendo eficiente y útil con nuevas configuraciones de la red.

Los sistemas distribuidos funcionan de manera efectiva y eficiente a diferentes escalas. La escala más pequeña consiste en dos estaciones de trabajo y un servidor de ficheros, mientras que un sistema distribuido construido alrededor de una red de área local simple podría contener varios cientos de estaciones de trabajo, varios servidores de ficheros, servidores de impresión y otros servidores de propósito específico. A menudo se conectan varias redes de área local para formar *internetworks* y éstas podrían contener muchos miles de ordenadores que forman un único sistema distribuido, permitiendo que los recursos sean compartidos entre ellos.

Tanto el software de sistema como el de aplicación no deberían cambiar cuando la escala del sistema se incrementa. La necesidad de escalabilidad no es sólo un problema de prestaciones de red o de hardware, sino que esta íntimamente ligada con todos los aspectos del diseño de los sistemas distribuidos. El diseño del sistema debe reconocer explícitamente la necesidad de

escalabilidad o de lo contrario aparecerán serias limitaciones.

La demanda de escalabilidad en los sistemas distribuidos ha conducido a una filosofía de diseño en que cualquier recurso simple, hardware o software, puede extenderse para proporcionar servicio a tantos usuarios como se quiera. Esto es, si la demanda de un recurso crece, debería ser posible extender el sistema para dar servicio. Por ejemplo, la frecuencia con la que se accede a los ficheros crece cuando se incrementa el número de usuarios y estaciones de trabajo en un sistema distribuido. Entonces, debe ser posible añadir ordenadores servidores para evitar el cuello de botella que se produciría si un solo servidor de ficheros tuviera que manejar todas las peticiones de acceso a los ficheros. En este caso el sistema deberá estar diseñado de manera que permita trabajar con ficheros replicados en distintos servidores, con las consideraciones de consistencias que ello conlleva.

Cuando el tamaño y complejidad de las redes de ordenadores crece, es un objetivo primordial diseñar software de sistema distribuido que seguirá siendo eficiente y útil con esas nuevas configuraciones de la red. Resumiendo, el trabajo necesario para procesar una petición simple para acceder a un recurso compartido debería ser prácticamente independiente del tamaño de la red. Las técnicas necesarias para conseguir estos objetivos incluyen el uso de datos replicados, la técnica asociada de caching, y el uso de múltiples servidores para manejar ciertas tareas, aprovechando la concurrencia para permitir una mayor productividad. [1]

- *Tolerancia a fallos:* El fallo de cualquier componente no afectará a los demás componentes conectados a la red.

Los sistemas informáticos a veces fallan. Cuando se producen fallos en el software o en el hardware, los programas podrían producir resultados incorrectos o podrían pararse antes de terminar la computación que estaban realizando. El diseño de sistemas tolerantes a fallos se basa en dos cuestiones, complementarias entre sí: *Redundancia hardware* (uso de componentes redundantes) y *recuperación del software* (diseño de programas que sean capaces de recuperarse de los fallos).

La recuperación del software tiene relación con el diseño de software que sea capaz de recuperar el estado de los datos permanentes antes de que se produjera el fallo.

Los sistemas distribuidos también proveen un alto grado de disponibilidad en la vertiente de fallos hardware. La disponibilidad de un sistema es una medida de la proporción de tiempo que está disponible para su uso. Un fallo simple en una máquina multiusuario resulta en la no disponibilidad del sistema para todos los usuarios. Cuando uno de los componentes de un sistema distribuidos

falla, sólo se ve afectado el trabajo que estaba realizando el componente averiado. Un usuario podría desplazarse a otra estación de trabajo; un proceso servidor podría ejecutarse en otra máquina.

- *Transparencia:* La transparencia se define como la ocultación al usuario y al programador de aplicaciones de la separación de los componentes de un sistema distribuido, de manera que el sistema se percibe como un todo, en vez de una colección de componentes independientes. La transparencia ejerce una gran influencia en el diseño del software de sistema. [1]

Existen ocho formas de transparencia. Estas proveen un resumen útil de la motivación y metas de los sistemas distribuidos.

- ✓ Transparencia de Acceso: Permite el acceso a los objetos de información remotos de la misma forma que a los objetos de información locales.
- ✓ Transparencia de Localización: Permite el acceso a los objetos de información sin conocimiento de su localización
- ✓ Transparencia de Concurrencia: Permite que varios procesos operen concurrentemente utilizando objetos de información compartidos y de forma que no exista interferencia entre ellos.
- ✓ Transparencia de Replicación: Permite utilizar múltiples instancias de los objetos de información para incrementar la fiabilidad y las prestaciones sin que los usuarios o los programas de aplicación tengan por qué conocer la existencia de las replicas.
- ✓ Transparencia de Fallos: Permite a los usuarios y programas de aplicación completar sus tareas a pesar de la ocurrencia de fallos en el hardware o en el software.
- ✓ Transparencia de Migración: Permite el movimiento de objetos de información dentro de un sistema sin afectar a los usuarios o a los programas de aplicación.
- ✓ Transparencia de Prestaciones. Permite que el sistema sea reconfigurado para mejorar las prestaciones mientras la carga varia.
- ✓ Transparencia de Escalado: Permite la expansión del sistema y de las aplicaciones sin cambiar la estructura del sistema o los algoritmos de la aplicación.

Las dos formas de transparencia más importantes son las transparencias de acceso y de localización; su presencia o ausencia afecta fuertemente a la utilización de los recursos distribuidos. A menudo se les denomina a ambas *transparencias de red*. La transparencia de red provee un grado similar de

anonimato en los recursos al que se encuentra en los sistemas centralizados.

OTRAS CARÁCTERISTICAS:

- Cada elemento de cómputo tiene su propia memoria y su propio Sistema Operativo.
- Control de recursos locales y remotos.
- Sistemas Abiertos (Facilidades de cambio y crecimiento).
- Plataforma no estándar (Unix, NT, Intel, RISC, Etc.).
- Medios de comunicación (Redes, Protocolos, Dispositivos, Etc.).
- Capacidad de Procesamiento en paralelo.
- Dispersión y parcialidad. [2]

1.1.2 VENTAJAS Y DESVENTAJAS DE LOS SISTEMAS DISTRIBUIDOS.

VENTAJAS [2]

- Capacidad de procesamiento elevada y a menos costos.
- Uso de nuevas interfaces.
- Avances en la Tecnología de Comunicaciones.
  - ✓ Disponibilidad de elementos de Comunicación.
  - ✓ Desarrollo de nuevas técnicas.
- Compartición de Recursos.
  - ✓ Dispositivos (Hardware).
  - ✓ Programas (Software).
- Eficiencia y Flexibilidad.
  - ✓ Respuesta Rápida.
  - ✓ Ejecución Concurrente de procesos (En varias computadoras).
  - ✓ Empleo de técnicas de procesamiento distribuido.
- Disponibilidad y Confiabilidad.
  - ✓ Sistema poco propenso a fallos (Si un componente falla no afecta la disponibilidad del sistema).

- ✓ Mayores servicios que elevan la funcionalidad (Monitoreo, Telecontrol, Correo Eléctrico, Etc.).

- Crecimiento Modular.
  - ✓ Es inherente al crecimiento.
  - ✓ Inclusión rápida de nuevos recursos.
  - ✓ Los recursos actuales no afectan.

DESVENTAJAS

- Gran diversidad de criterios sobre el software, diseño, implementación y uso del software distribuido.
- Velocidad de propagación de información (Muy lenta a veces).
- Puede surgir problemas en la seguridad de la maquina al compartir los recursos.
- Uso ineficiente de los recursos distribuidos.
- Capacidad reducida para administrar apropiadamente grupos de procesadores y memoria localizada en distintos sitios.
- Enorme dependencia del desempeño de la red y de la confiabilidad de la misma (problemas de pérdida de mensajes, saturación en el tráfico de datos).
- Mayor complejidad en la administración y mantenimiento.
- Mayor complejidad en su construcción.[3]

1.2 LOS SISTEMAS DISTRIBUIDOS Y SU APLICACIÓN EN LA BIOINFORMÁTICA.

Como ya se ha mencionado anteriormente los sistemas distribuidos son aplicados en diferentes campos de la ciencia y la técnica siempre en aras de alcanzar resultados que ganen en eficiencia y exactitud. Las necesidades actuales para la realización de proyectos Bioinformáticos de aprovechar los recursos disponibles en los sistemas informáticos conectados a la red y simplificar su utilización, ha dado lugar a una nueva forma de tecnología de la información conocida como Grid Computing. De este modo, los sistemas distribuidos se pueden emplear como un único sistema virtual en aplicaciones intensivas en datos o con alta demanda computacional. Las necesidades básicas de la biología se pueden expresar como una mayor demanda de capacidad de almacenamiento y tratamiento de

información, y un crecimiento desmesurado de la capacidad de cálculo. La forma más efectiva en coste y recursos de resolver los problemas actuales de las Ciencias Biológicas y responder a las crecientes demandas sociales sobre ellas es recurrir a sistemas masivamente distribuidos de tecnología Grid. [4] En el mundo se han desarrollado una serie de programas sobre sistemas distribuidos algunos de ellos son:

GATE: Permite modelar y planificar el tratamiento sobre tumores empleando la herramienta Geant4 de simulación.

Siesta-2.0: Está diseñado para cálculos de estructuras mecano-cuántica lineal. Realiza cálculos de dinámica molecular ab-initio.[5]

En la Universidad de las Ciencias Informaticas se desarrolló un módulo para el cálculo distribuido de mecánica cuántica y molecular sobre la plataforma JDCS, con el cual se logró la reducción de los tiempo de procesamiento del MOPAC en un 98%. Actualmente en la Facultad 6 se estan desarrollando aplicaciones distribuidas para reducir los tiempos de respuesta de los cálculos realizados por el Vega, la metodología MMH y el Gaussian.

1.3 GAUSSIAN

Gaussian es capaz de realizar cálculos a diferentes niveles de teoría: ab-initio, semiempíricos, teoría del funcional de la densidad e interacción de configuración; permitiendo modelar sistemas químicos que van desde átomos hasta moléculas de muchos electrones, en estados base y excitados, en fase gaseosa o en solución. Gaussian es utilizado por químicos, bioquímicos, físicos y otros profesionales para la investigación. Existen proteínas y moléculas que por su tamaño han estado fuera del alcance de los métodos de estructura electrónica, Gaussian contiene un método nombrado ONIOM que surge primero con Gaussian 98 y luego se mejora en Gaussian 03 que eliminará estas limitaciones.[5] Los métodos semiempíricos son de bajo costo computacional pero la exactitud en la predicción desde el punto de vista cualitativo y cuantitativo depende en gran medida del tamaño del sistema, del número de átomos que lo conformen y de que tan buenos sean los conjuntos de parámetros, mientras que los métodos ab-initio son muy costosos computacionalmente pero a diferencia de los semiempíricos la exactitud de las predicciones cualitativas y cuantitativas es mayor. Los diversos módulos del programa permiten:

- Optimizar estructuras moleculares, determinar energías, espectros vibracionales y electrónicos así como las cargas atómicas en los sistemas moleculares.
- Calcular estados fundamentales y estados excitados, singletes y tripletes (ZINDO, CIS y TD).

- Otros cálculos posibles son IRC (para recorrido de reacción), SCAN (explora una región de la superficie de energía potencial), STABILITY (reoptimiza la función de onda) y finalmente determina espectros de resonancia magnética nuclear.

Gaussian es excelente como instrumento para la enseñanza y la investigación, entre sus capacidades se encuentra la predicción de:

- Energías y estructuras moleculares.
- Energías y estructuras de estado de transición.
- Frecuencias vibracionales.
- Espectros infrarrojos y Raman.
- Propiedades termodinámicas.
- Energías de reacción y de enlace.
- Trayectorias de reacción.
- Orbitales moleculares
- Cargas atómicas.
- Momentos multipolares.
- Apantallamiento en resonancia magnética nuclear y susceptibilidad magnética.
- Afinidades electrónicas y potenciales de ionización.
- Polarización e hiperpolarización.
- Potenciales electrostáticos y densidades electrónicas

En nuestro país es utilizado en diferentes centros como la universidad de Camagüey, la universidad de Villa Clara, la universidad de La Habana entre otros. Al terminar el desarrollo del sistema este será de gran utilidad para todos estos centros pues en ninguno se brinda la posibilidad de realizar estos cálculos de alto nivel de forma distribuida.

1.4 Herramientas y Metodologías

- *El lenguaje Unificado de Modelado UML:*

El Lenguaje Unificado de Modelado permite modelar, construir y documentar los elementos que forman un producto de software que responde a un enfoque orientado a objetos. Es un lenguaje de propósito

general y también puede considerarse como un lenguaje de modelado visual. Intenta solucionar el problema de propiedad de código que se da con los desarrolladores, al implementar un lenguaje de modelado común para todos los desarrollos se crea una documentación también común, que cualquier desarrollador con conocimientos de UML será capaz de entender, independientemente del lenguaje utilizado. En estos momentos se convierte en un Standard, no existe otra especificación de diseño orientado a objetos, ya que es el resultado de las tres opciones existentes en el mercado. Su utilización es independiente del lenguaje de programación y de las características de los proyectos, ya que UML ha sido diseñado para modelar cualquier tipo de proyectos, tanto informáticos como de arquitectura, o de cualquier otro rama [6]. Permite la modificación de todos sus miembros mediante estereotipos y restricciones. Un estereotipo nos permite indicar especificaciones del lenguaje al que se refiere el diagrama de UML. Una restricción identifica un comportamiento forzado de una clase o relación, es decir mediante la restricción estamos forzando el comportamiento que debe tener el objeto al que se le aplica.

- *Metodología XP*

La metodología XP o Extreme Programming es una de las variantes de las metodologías ágiles con más aceptación en la comunidad internacional de desarrollo. Su creador, Kent Beck la comenzó a gestar junto con Ward Cunningham en 1990 y tomó su forma final en 1996.

Los fundamentos de la XP según Beck son: mejorar la comunicación, buscar la simplicidad, buscar retroalimentación en que tan bien va nuestro trabajo y siempre hay que proceder con valentía.

En consistencia con sus valores, una de las formas de crear documentación en la XP es mediante el mismo proceso de desarrollo, y esto se logra mediante las pruebas unitarias. Una de las herramientas más importantes de la XP es el desarrollo orientado a pruebas, que utiliza las pruebas unitarias como eje de todo desarrollo. Siendo una directriz de esta herramienta no escribir código para el que no tengamos previamente una prueba unitaria, no sólo mejoramos la seguridad del desarrollo, sino que también documentamos el código de producción con el código de pruebas. Las pruebas unitarias nos dicen qué código existe, qué se espera que haga y cómo se usa, al mismo tiempo que permiten un apoyo imprescindible en “refactoring” (otro pilar de la XP y de la orientación a pruebas) y mantenimiento del código.[13]

La documentación “ágil” puede no ser lo más convencional del mundo, pero está enfocada a ser funcional, exhaustiva, formalmente descriptiva, actualizada y de desarrollo concurrente con el código que documenta. El inconveniente que pudiera tener este enfoque es al momento de comunicarnos con los usuarios o con los gerentes de producto, más preocupados por la “usabilidad” del sistema que por

las entrañas del mismo. Pero incluso aquí la XP no tiene problemas. Aunque no es indispensable, se puede utilizar documentación de usuario como cartas de requerimientos, historias de uso, cartas de responsabilidad y otros artefactos para apoyar a los miembros del equipo que no tienen ni necesitan un conocimiento profundo sobre el código del sistema. En este sentido, uno de los grupos de artefactos de documentación que mejor conjuntan las dos necesidades: agilidad en la elaboración y documentación protocolaria, son los definidos por el MSF (Microsoft Solution Framework). La XP combina de manera muy eficiente una buena práctica de documentación con un enfoque iterativo mucho más agresivo. Las iteraciones suelen ser muy cortas y se promueve que los programadores busquen soluciones y experimenten con ellas, programar sin miedo a descomponer el sistema permite un trabajo mucho más ágil y creativo. Por supuesto, esto no sería posible sin buenas prácticas de "versionamiento" que para la XP son indispensables. En XP se procura trabajar en parejas, dos desarrolladores por máquina piensan mejor que uno sólo. Al igual que esta técnica, las reuniones diarias del equipo para discutir avances y un continuo trabajo de talleres favorecen una integración mayor en el equipo de desarrollo, una mejor y más eficiente comunicación y un desarrollo más rápido. Otras prácticas como la integración continua y el monitoreo de las métricas del código apoyan a la XP para conseguir resultados rápidos con software muy seguro, resistente a "refactoring" y fácil de mantener. [13]

DERECHOS DEL CLIENTE

- Decidir que se implementa.
- Saber el estado real y el progreso del proyecto.
- Añadir, cambiar o quitar requerimientos en cualquier momento.
- Obtener lo máximo de cada semana de trabajo.
- Obtener un sistema funcionando cada 3 o 4 meses.

DERECHOS DEL DESARROLLADOR

- Decidir como se implementan los procesos.
- Crear el sistema con la mejor calidad posible.
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos.
- Estimar el esfuerzo para implementar el sistema.
- Cambiar los requerimientos en base a nuevos descubrimientos.

- Lo fundamental en este tipo de metodología es:
  - ✓ La comunicación, entre los usuarios y los desarrolladores
  - ✓ La simplicidad, al desarrollar y codificar los módulos del sistema
  - ✓ La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales.[14]

- *Rational Unified Process (RUP)*

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

Inicio: El Objetivo en esta etapa es determinar la visión del proyecto.

Elaboración: En esta etapa el objetivo es determinar la arquitectura óptima.

Construcción: En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.

Transición: El objetivo es llegar a obtener el release del proyecto.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes.
Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

DISCIPLINA DE DESARROLLO

Ingeniería de Negocios: Entendiendo las necesidades del negocio.

Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.

Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.

Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.

Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

DISCIPLINA DE SOPORTE

Configuración y administración del cambio: Guardando todas las versiones del proyecto.

Administrando el proyecto: Administrando horarios y recursos.

Ambiente: Administrando el ambiente de desarrollo.

Distribución: Hacer todo lo necesario para la salida del proyecto

Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

LOS ELEMENTOS DEL RUP SON:

Actividades: Son los procesos que se llegan a determinar en cada iteración.

Trabajadores: Vienen hacer las personas o entes involucrados en cada proceso.

Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software. [14]

- *Open UP*

Enfocados en brindar un servicio de calidad y de reconocimiento internacional, nuestros proyectos de desarrollo de software están fundamentados en el framework de Proceso Unificado de Desarrollo de Software OpenUP, desarrollado por una de las comunidades de software libre más grandes del mundo: Eclipse.Org.

Este proceso de desarrollo unificado está basado en Rational Unified Process (RUP), desarrollado por IBM y reconocido mundialmente como uno de los procesos de desarrollo de software de mayor calidad, basándose en los principios de Adaptación, Importancia a los involucrados e interesados en los resultados del proyecto; Colaboración, Valor a la iteración; y Calidad Continua.[7]

CONCLUSIONES DEL CAPÍTULO

Luego de hacer un estudio de los Sistemas Distribuidos, sus ventajas y desventajas así como su aplicación a la Bioinformática y analizar las características del Gaussian y su forma de funcionamiento se llega a la conclusión de que la utilización de un sistema distribuido puede ser una posible vía de solución para el desarrollo del sistema. Se definen las herramientas y metodología a utilizar.

CAPÍTULO 2
MATERIALES Y MÉTODOS

En este Capítulo se hace una breve descripción de las herramientas y las metodologías para desarrollar el sistema, justificando al mismo tiempo la utilización de las mismas. Además se da una breve introducción sobre lo referente a tecnologías GRID, plataformas distribuidas y funcionalidades de la aplicación Gaussian.

2.1 LENGUAJE DE MODELADO Y METODOLOGÍAS.

El lenguaje de modelado de objetos es un conjunto estandarizado de símbolos y de modos de disponerlos para modelar parte de un diseño de software orientado a objetos. Algunas organizaciones los usan extensivamente en combinación con una metodología de desarrollo de software para avanzar de una especificación inicial a un plan de implementación y para comunicar dicho plan a todo un equipo de desarrolladores. El uso de un lenguaje de modelado es más sencillo que la auténtica programación, pues existen menos medios para verificar efectivamente el funcionamiento adecuado del modelo. Esto puede suponer también que las interacciones entre partes del programa den lugar a sorpresas cuando el modelo ha sido convertido en un software funcionante. Por otra parte una metodología es un conjunto de actividades necesarias para transformar los requisitos de los usuarios en un sistema de software. Cuando se va a desarrollar dicho software este suele tornarse algo difícil y riesgoso pues pueden existir clientes insatisfechos con el resultado o desarrolladores aún mas insatisfechos, para evitar este tipo de situaciones lo primero que debe decidirse a la hora de comenzar el desarrollo del mismo es la metodología a utilizar.

Por todo lo antes expuesto se realizó un estudio de los lenguajes de modelado y las metodologías que puedan ser aplicadas a un software como el que propuesto a desarrollar y que además cumplieran con las reglas de calidad establecidas por la universidad, para luego decidir el lenguaje y la metodología mas adecuados para el desarrollo del mismo.

2.1.1 LENGUAJE UNIFICADO DE MODELADO (UML)

Ventajas del lenguaje de Unificado de Modelado (UML)

- Cubre las cuestiones relacionadas con el tamaño propio de los sistemas complejos y críticos.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.
- Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis, diseño e implementación, construyendo así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo.

2.1.2 OPEN UNIFIED PROCESS (OPENUP)

OpenUP es una metodología de procesos de desarrollo de software de código abierto, que con el tiempo espera cubrir un amplio conjunto de necesidades en el campo del desarrollo de software. Permite un abordaje ágil al proceso de desarrollo de software, con sólo proveer un conjunto simplificado de contenidos, fundamentalmente relacionados con orientación, productos de trabajo, roles, y tareas. Es un proceso interactivo de desarrollo de software simplificado, completo y extensible para pequeños equipos de desarrollo que valoran los beneficios de la colaboración y de los involucrados con el resultado del proyecto, por encima de formalidades innecesarias. OpenUP está caracterizado por cuatro principios básicos interrelacionados, a saber:

- Colaboración para unificar intereses y compartir conocimientos.
- Equilibrio de prioridades competentes a maximizar el valor de los involucrados con el resultado del proyecto.
- Enfoque en la articulación de la arquitectura.
- Desarrollo continuo para obtener realimentación y realizar las mejoras respectivas.

Se centra en articular la arquitectura para facilitar la colaboración técnica, reducir el riesgo y minimizar el sobreesfuerzo de desarrollo, procura un equilibrio entre las necesidades de los involucrados con los resultados del proyecto y los costos técnicos, con el fin de maximizar el valor de los involucrados y las

guías del proceso de desarrollo. OpenUP desarrolla un ciclo de vida interactivo que mitiga el riesgo a tiempo y ofrece demostrar resultados en curso al cliente del proyecto.

Es extensible debido a que este puede ser usado como punto de partida para desarrollar otros procesos de desarrollo de software o inclusive simplificarlo como sea necesario. Es un proceso dirigido a gestión y desarrollo de proyectos de software basados en desarrollo iterativo, ágil e incremental; y es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. [7]

2.2 HERRAMIENTAS DE MODELADO Y PROGRAMACIÓN.

El modelado de software es una técnica para tratar con la complejidad inherente a estos sistemas. El uso de modelos ayuda al ingeniero que desarrollara el software a visualizar el sistema a construir. Por la importancia que tienen las herramientas de modelado y de programación a la hora de desarrollar un sistema, se debe hacer un estudio preliminar de ellas para trabajar con las más adecuadas y llegas así a desarrollar un software con eficiencia y calidad. A continuación se describen las herramientas a utilizar en el desarrollo del sistema.

2.2.1 HERRAMIENTA CASE VISUAL PARADIGM.

Visual Paradigm es una herramienta de modelado gráfico fácil de usar y de soporte multiplataforma, proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Visual Paradigm fue creado para el ciclo vital completo del desarrollo del software que lo automatiza y acelera, permitiendo la captura de requisitos, análisis, diseño e implementación, también proporciona características tales como generación del código, ingeniería inversa y generación de informes.

Está diseñado para usuarios interesados en sistemas de software de gran escala con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. A pesar de todas estas posibilidades la licencia es muy restringida y las imágenes y reportes generados, no son de muy buena calidad. Visual Paradigm ofrece:

- ✓ Entorno de creación de diagramas para UML 2.0.
- ✓ Diseño centrado en casos de uso y enfocado al negocio.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Capacidades de ingeniería directa (versión profesional) e inversa.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- ✓ Disponibilidad de múltiples versiones, para cada necesidad.
- ✓ Disponibilidad de integrarse en los principales IDEs.
- ✓ Multiplataforma. [8]

2.2.2 CARACTERÍSTICAS PRINCIPALES DE JAVA.

Java supone un significativo avance en el mundo de los entornos software, y esto viene avalado por tres elementos claves que diferencian a este lenguaje desde un punto de vista tecnológico:

- Pone al alcance de cualquiera la utilización de applets.
- Es un lenguaje de programación que ofrece la potencia del diseño orientado a objetos con una sintaxis fácilmente accesible y un entorno robusto y agradable.
- Proporciona un conjunto de clases potente y flexible.

Dentro de sus características principales tenemos:

SIMPLE

El único requerimiento para aprender Java es tener una comprensión de los conceptos básicos de la programación orientada a objetos; se ha pretendido un lenguaje simple sin caer en la ineficacia y la falta de expresividad, pudiendo mostrarse cualquier planteamiento por parte del programador sin que las interioridades del sistema subyacente sean desveladas.

POTENTE

Aunque cada tarea se puede realizar de un número reducido de formas, se ha conseguido un gran potencial de expresión e innovación desde el punto de vista del programador.

SEGURO

Existe una preocupación lógica en Internet por el tema de la seguridad pues los virus navegan de forma usual por la red, constituyendo una amenaza palpable. Java ha sido diseñado poniendo un énfasis especial en el tema de la seguridad, y se ha conseguido lograr cierta inmunidad en el aspecto de que un programa realizado en Java no puede realizar llamadas a funciones globales y acceder a recursos arbitrarios del sistema, por lo que el control sobre los programas ejecutables no es equiparable a otros lenguajes.

Niveles de seguridad:

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de bytecode que asegura que no se viole ninguna construcción del lenguaje.

- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles

ORIENTADO A OBJETOS

Java fue diseñado en este aspecto partiendo de cero, no es un derivado de otro lenguaje anterior y no tiene compatibilidad con ninguno de ellos. Proporciona un mecanismo de clasificación simple y presenta un modelo de interfaz dinámica intuitiva en caso de necesidad.

ROBUSTO

Java verifica su código al mismo tiempo que lo escribe, y una vez más antes de ejecutarse, de manera que se consigue un alto margen de codificación sin errores. Se realiza un descubrimiento de la mayor parte de los errores durante el tiempo de compilación, ya que Java es estricto en cuanto a tipos y declaraciones. Respecto a la gestión de memoria, Java libera al programador del compromiso de tener que controlar especialmente la asignación que de ésta. Este lenguaje posee una gestión avanzada de memoria llamada gestión de basura, y un manejo de excepciones orientado a objetos integrados. Estos elementos realizarán muchas tareas que antes se tornaban tediosas y a la vez obligadas para el programador.

INTERACTIVO

Uno de los requisitos de Java desde sus inicios fue la posibilidad de crear programas en red interactivos, por lo que es capaz de hacer varias cosas a la vez sin perder el rastro de lo que debería suceder y cuándo. Para que esto sea posible cuenta con múltiples hilos de utilización sencilla que le permiten pensar en el comportamiento específico que se intenta codificar, sin tener que integrarlo en un modelo de programación global.

ARQUITECTURA NEUTRAL

Java está diseñado para que un programa escrito en este lenguaje sea ejecutado correctamente independientemente de la plataforma en la que se esté actuando (Macintosh, PC, UNIX…). Para conseguir esto utiliza una compilación en una representación intermedia que recibe el nombre de código de byte de Java, que puede interpretarse en cualquier sistema con intérprete de Java. La desventaja de un sistema de este tipo es el rendimiento; sin embargo, el hecho de que Java fuese diseñado para funcionar razonablemente bien en CPU de escasa potencia, unido a la sencillez de traducción a código máquina hacen que Java supere esa desventaja sin problemas.

FÁCIL APRENDIZAJE

Java es más complejo que un lenguaje simple, pero más sencillo que cualquier otro entorno de programación. El único obstáculo que se puede presentar es la consecución de la comprensión de la programación orientada a objetos.

LENGUAJE

Java fue desarrollado basándose en C++, pero eliminando rasgos del mismo poco empleados, optándose por una codificación comprensible. Básicamente, encontramos las siguientes diferencias con C++:

- Java no soporta los tipos struct, union y pointer
- No soporta typedef ni #define
- Se distingue por su forma de manejar ciertos operadores y no permite una sobrecarga del operador
- No soporta herencia múltiple
- Java maneja argumentos en la línea de comandos de forma diversa a como lo hacen C o C++
- Tiene una clase string que es parte del paquete java.lang package y se diferencia de la matriz de caracteres terminada con un nulo que usan C y C++
- Java cuenta con un sistema automático para asignar y liberar memoria, con lo que no es necesario utilizar las funciones previstas con este fin en C y C++.
- En Java podemos crear los siguientes tipos de construcciones:
- Miniaplicaciones (applets)
- Aplicaciones, estas se ejecutan sin necesidad de un navegador
- Manipuladores de protocolo (interpretan protocolos).
- Manipuladores de contenido (interpretan archivos).
- Métodos nativos (métodos declarados en una clase Java, pero implementados en C).

2.2.3 ECLIPSE

Eclipse es un entorno de desarrollo de Java, una poderosa herramienta que permite integrar diferentes aplicaciones para construir un entorno integrado de desarrollo (IDE). Usa java como lenguaje de programación aunque permite plugins para otros lenguajes, soporta programación orientada a objetos (POO), la depuración e implementación de aplicaciones resultan mucho más sencillas. Tiene varios beneficios:

- Es una herramienta de código abierto.
- Soporta herramientas que manipulan diferentes tipos de lenguajes, como por ejemplo Java, C, C++,
- Se ejecuta en varios sistemas operativos incluyendo Windows y Linux.
- Provee a los desarrolladores, herramientas que facilitan la creación de plugins.

2.3 PLATAFORMA DISTRIBUIDA

2.3.1 JAVA RMI (REMOTE METHOD INVOCATION) INVOCACIÓN A MÉTODOS REMOTOS

RMI es un paquete de JAVA que permite manejar objetos (y sus respectivos métodos) de manera remota, para utilizar los recursos de un servidor de manera transparente para el usuario local.

Las aplicaciones RMI normalmente comprenden dos programas separados: un servidor y un cliente. Una aplicación servidor crea una serie de objetos remotos, hace accesibles unas referencias a dichos objetos remotos, y espera a que los clientes llamen a estos métodos u objetos. Una aplicación cliente obtiene una referencia remota de uno o más objetos en el servidor y llama a sus métodos. RMI proporciona el mecanismo por el que se comunican y se pasan información del cliente al servidor y viceversa. Cuando es una aplicación, algunas veces nos referimos a ella como Aplicación de Objetos Distribuidos.

Las aplicaciones de objetos distribuidos necesitan:

- *Localizar Objetos Remotos*

  Las aplicaciones pueden utilizar uno de los dos mecanismos para obtener referencias a objetos remotos. Puede registrar sus objetos con la facilidad de nombrado de RMI: rmiregistry. O puede pasar y devolver referencias de objetos como parte de su operación normal.

- *Comunicar con Objetos Remotos*

Los detalles de la comunicación entre objetos remotos son manejados por el RMI; para el programador, la comunicación remota se parecerá a una llamada estándar a un método Java.

- *Cargar Bytecodes para objetos que son enviados.*

Como RMI permite al llamador pasar objetos Java a objetos remotos, RMI proporciona el mecanismo necesario para cargar el código del objeto, así como la transmisión de sus datos.

Una de las principales y únicas características de RMI es la habilidad de descargar los bytecodes de una clase de un objeto si la clase no está definida en la máquina virtual del recibidor. Los tipos y comportamientos de un objeto, anteriormente sólo disponibles en una máquina virtual, ahora pueden ser transmitidos a otra máquina virtual, posiblemente remota. RMI pasa los objetos por su tipo verdadero, por eso el comportamiento de dichos objetos no cambia cuando son enviados a otra máquina virtual. Esto permite que los nuevos tipos sean introducidos en máquinas virtuales remotas, y así extender el comportamiento de una aplicación dinámicamente.

Una aplicación distribuida construida utilizando RMI de Java, al igual que otras aplicaciones Java, está compuesta por interfaces y clases. Las interfaces definen métodos, mientras que las clases implementan los métodos definidos en las interfaces y también definen algunos métodos adicionales. En una aplicación distribuida, se asume que algunas implementaciones residen en diferentes máquinas virtuales. Los objetos que tienen métodos que pueden llamarse por distintas máquinas virtuales son los objetos remotos.

Un objeto se convierte en remoto implementando una interface remota, que tenga estas características.

- ✓ Una interface remota desciende de la interface java.rmi.Remote.
- ✓ Cada método de la interface declara que lanza una java.rmi.RemoteException además de cualquier excepción específica de la aplicación.

Metas del Sistema RMI de Java

- Proporcionar invocación remota de objetos que se encuentran en máquinas virtuales diferentes.
- Soportar llamadas a los servidores desde los applets.
- Integrar el modelo de objetos distribuidos en el lenguaje Java de una manera natural,

conservando en medida de lo posible la semántica de los objetos Java.

- Hacer tan simple como sea posible la escritura de aplicaciones distribuidas.
- Preservar la seguridad proporcionada por el ambiente Java.
- Proporcionar varias semánticas para las referencias de los objetos remotos (persistentes, no persistentes y de activación retardada).

Elementos

Toda aplicación RMI normalmente se descompone en 2 partes:

- ✓ Un *servidor*, que crea algunos objetos remotos, crea referencias para hacerlos accesibles, y espera a que el cliente los invoque.
- ✓ Un *cliente*, que obtiene una referencia a objetos remotos en el servidor, y los invoca.

Ventajas de usar RMI:

Permite distribuir una aplicación de forma muy transparente, es decir, sin que el programador tenga que modificar apenas el código. Además las invocaciones remotas son más eficientes que las peticiones vía http que se usan con los cgis o los servlets.

Inconvenientes de usar RMI:

El paso de parámetros por valor implica tiempo para hacer la serialización, enviar los objetos serializados a través de la red y luego volver a deserializarlos en el destino; sobretodo si los parámetros son objetos complejos. [9]

2.3.2 TECNOLOGÍA GRID

Permite utilizar de forma coordinada todo tipo de recursos (cómputo, almacenamiento y aplicaciones específicas) para gestionar y distribuir la potencia de cálculo disponible hacia todos los usuarios conectados a la red. Las tecnologías Grid permiten que el marco en el que se comparten los recursos sea:

- Seguro, mediante el uso de certificados de usuario y listas de autorización centralizada por organización virtual.
- Robusto, permitiendo la redundancia de recursos y datos el relanzamiento automático de procesos.

- Escalable, mediante una estructura jerarquizada y automáticamente reconfigurable.
- Eficiente, a través del uso de sistemas de balance de carga e identificación de los recursos más adecuados. [10]

Entre las ventajas que ofrece la Tecnología Grid tenemos:

- Potencia ilimitada que ofrecen multitud de ordenadores conectados en red ofreciendo su capacidad de proceso.
- Ahorra tiempo, ya que evita los cuellos de botella de algunos procesos de computación.
- Se integra fácilmente con las aplicaciones y procesos de la compañía.
- Permite ahorrar costes, ya que la inversión se amortiza al 100 por ciento inmediatamente y a diferencia de la compra de grandes equipos, nunca queda obsoleta.

2.3.3 PLATAFORMA DISTRIBUIDA JDCS

La plataforma Java Distributed Computing System (Sistema Java de Cómputo Distribuido) se ha diseñado para la implementación simplificada de aplicaciones en un ambiente distribuido.

El sistema está representado por una arquitectura MIMD y dentro de sus características esenciales se encuentran:

- Transparencia.
- Eficiencia.
- Flexibilidad.
- Escalabilidad.
- Fiabilidad.
- Conexión desde otro software.
- Grupos de usuarios con privilegios según el rol que desempeñan.
- Autorización de elementos de cómputos.

Para llevar a cabo un cómputo distribuido en esta plataforma se necesitan componentes especiales como:

- Máquina Virtual de JAVA: La plataforma está desarrollada con este lenguaje de programación.
- Servidor: Encargado de dirigir el proceso.
- Cliente: Son los encargados de procesar los cálculos.

- Interface: Actúa de intermediaria entre el cliente y el sistema especificándole a este último los elementos necesarios para realizar el cómputo distribuido.

Para definir completamente un cómputo distribuido en el sistema, se necesita solamente extender dos clases de Java:

- La clase DataManager que funciona en el servidor
- La clase Algorithm que funciona en el cliente.

Para la ejecución del cómputo distribuido deben implementarse varios métodos propios de las clases DataManager y Algorithm.

CLASE DATA MANAGER:

La clase DataManager se implementa en el servidor. Dicha clase tiene como propósito generar las unidades de trabajo que se enviarán a los clientes para que estos realicen los cálculos correspondientes, además debe procesar los resultados que se obtienen una vez realizados los cálculos, también deben implementarse funcionalidades que logren monitorear y ajustar el particionamiento de las unidades de trabajo, generar el estado actual del problema que se mostrará mediante la interfaz remota y terminar el cómputo distribuido. El desarrollador debe implementar las funcionalidades que se mencionan a continuación para que el cómputo sea aceptado por el sistema:

- generateWorkUnit.
- processResults.
- adjustGranularity.
- getStatus.
- closeResources.

CLASE ALGORITHM:

Algorithm es parte del código que corre en el cliente y procesa las unidades de trabajo generadas por el método generateWorkUnit del DataManager. Solamente un método se debe implementar en la clase Algorithm:

- *processUnit.*

2.4 MODELO DISTRIBUIDO CLIENTE – SERVIDOR

El modelo cliente-servidor es un modelo para construir sistemas de información, que se basa en la idea de repartir toda la información y los datos por todo el sistema informático, permitiendo mejorar el rendimiento de este. Este modelo proporciona al usuario el acceso transparente a las aplicaciones,

datos, servicios de cómputo o cualquier otro recurso del grupo de trabajo mediante la organización o en múltiples plataformas. El modelo soporta un medio ambiente distribuido donde los requerimientos de servicio hechos por los clientes pueden convertirse en múltiples requerimientos de trabajo a través de redes, terminan en un trabajo realizado por los servidores que son cualquier recurso de cómputo dedicado a responder a los requerimientos del cliente. Los servidores pueden estar conectados a los clientes a través de redes LANs o WANs, para proveer de múltiples servicios a los clientes tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc. Aunque el modelo cliente-servidor no satisface todos los requisitos necesarios para todas las aplicaciones distribuidas, es adecuado para muchas de ellas y provee una base efectiva para los sistemas operativos distribuidos de propósito general. Es un elemento esencial, la presencia de un medio físico de comunicación entre las máquinas, y dependerá de la naturaleza de este medio la viabilidad del sistema. [11]

A continuación se presenta una lista de los servidores más comunes:

- *Servidores de archivos:* Proporciona archivos para clientes. Dado a que los archivos son muy grandes y los usuarios que los comparten son muchos, esto no constituye una buena opción de almacenamiento y procesamiento de archivos. En este tipo de servidores los clientes solicitan el archivo y el servidor los busca y se los envía.
- *Servidores de Base de Datos*: Almacenan gran cantidad de datos. A diferencia de los servidores de archivos, la información que se envía del servidor al cliente ya está resumida en la base de datos por lo que este solo tiene que ubicar la información pertinente y enviar esa respuesta al cliente.
- *Servidores de Software de Grupo*: Permite organizar el trabajo de un grupo. El servidor proporciona datos que soportan las tareas a realizar. Un ejemplo de esto es al crear una lista de correo electrónico donde el cliente puede indicar que se ha terminado la tarea y el servidor es el encargado de informarlo al resto del grupo.
- *Servidores WEB:* Son los que guardan y brindan páginas HTML. El cliente desde un browser o link hace un llamado de la página y el servidor recibe el mensaje y envía la página que fue solicitada.
- *Servidores de correo:* Se responsabilizan con el envío y recepción de correos de un grupo de usuarios. El servidor solo debe utilizar un protocolo de correo.
- *Servidor de objetos:* Permite almacenar objetos que pueden ser activados a distancia. Los

clientes pueden ser capaces de activar los objetos que se encuentran en el servidor.

- *Servidores de impresión:* Son los encargados de atender las solicitudes de impresión realizadas por los clientes, este envía la solicitud, el servidor la recibe, da la orden a la impresora para que comience el proceso y al terminar avisa a la PC cliente que ha finalizado la tarea que le fue asignada.
- *Servidores de aplicación:* Se dedica a una única aplicación. Es básicamente una aplicación a la que pueden acceder los clientes.

Se distinguen tres componentes básicos de software:

Presentación: Consiste en presentar al usuario un conjunto de objetos visuales y llevar a cabo el procesamiento de los datos producidos por dicho usuario y los devueltos por el servidor.

Lógica de aplicación: Esta capa es la responsable de procesar toda la información que tiene lugar en la aplicación.

Base de datos: Está compuesta por los archivos que contienen los datos de la aplicación.

A continuación se muestran las arquitecturas cliente-servidor más conocidas:

ARQUITECTURA CLIENTE-SERVIDOR DE DOS CAPAS:

Consiste en una capa de presentación y lógica de la aplicación; y la otra de la base de datos. Normalmente esta arquitectura se utiliza en las siguientes situaciones:

- Se requiere poco procesamiento de datos en la organización.
- Se tiene una base de datos centralizada en un solo servidor.
- La base de datos es relativamente estática.
- Se requiere un mantenimiento mínimo. [11]

ARQUITECTURA CLIENTE-SERVIDOR DE TRES CAPAS:

Consiste en una capa de Presentación, otra capa de lógica de la aplicación y otra capa de base de datos. Normalmente esta arquitectura se utiliza en las siguientes situaciones:

- Se requiere mucho procesamiento de datos en la aplicación.
- Aplicaciones donde la funcionalidad este en constante cambio.

- Los procesos no están relativamente muy relacionados con los datos.
- Se requiere aislar la tecnología de la base de datos para que sea fácil de cambiar.
- Se requiere separar el código del cliente para que se facilite el mantenimiento.
- Está muy adecuada para utilizarla con la tecnología orientada a objetos. [11]

Los sistemas cliente/servidor se pueden clasificar de acuerdo al nivel de abstracción del servicio que ofrecen:

Representación distribuida: La interacción con el usuario se realiza en el servidor, el cliente hace de puente entre el usuario y el servidor.

Representación Remota: La lógica de la aplicación y la base de datos se encuentran en el servidor. El cliente recibe y formatea los datos para interactuar con el usuario.

Lógica Distribuida: El cliente se encarga de la interacción con el usuario y de algunas funciones triviales de la aplicación. Por ejemplo controles de rango de campos, campos obligatorios, etc. Mientras que el resto de la aplicación, junto con la base de datos, están en el servidor.

Gestión Remota de Datos: El cliente realiza la interacción con el usuario y ejecuta la aplicación y el servidor es quien maneja los datos.

Base de Datos Distribuidas: El cliente realiza la interacción con el usuario, ejecuta la aplicación, debe conocer la topología de la red, así como la disposición y ubicación de los datos. Se delega parte de la gestión de la base de datos al cliente.

Cliente servidor a tres niveles: El cliente se encarga de la interacción con el usuario, el servidor de la lógica de aplicación y la base de datos puede estar en otro servidor. [11]

2.5 PATRONES DE DISEÑO

Los patrones describen un problema que ocurre una y otra vez en nuestro entorno, para describir después la solución a ese problema, de tal manera que esa solución pueda ser usada más de una vez sin hacerlo ni siquiera dos veces de la misma forma. Ya que nuestro sistema estará implementado sobre la programación orientada a objetos los patrones de diseños a utilizar serán los GRASP (Patrones de Software para la asignación General de Responsabilidad) los cuales se describen a continuación:

- *Alta Cohesión:* Una Clase con baja cohesión hace muchas cosas no relacionadas mientras que una Clase con alta cohesión hace lo que uno podría esperar que hiciera. Si el sistema fallara por alguna razón es mucho más fácil encontrar responsabilidades si las Clases del sistema son cohesivas.
- *Bajo Acoplamiento:* Una clase con alto acoplamiento depende de muchas otras Clases para llevar a cabo su trabajo mientras que una Clase con bajo acoplamiento no depende de demasiadas Clases para hacer su trabajo. Las Clases se pueden reutilizar con mayor facilidad y flexibilidad.
- *Creador:* Consiste en asignar a un Objeto la responsabilidad de crear otro Objeto. Lo ideal es decidir estas cuestiones cuando se está diseñando y no cuando se está desarrollando y uno está inmerso entre miles de líneas de código.
- *Experto en Información:* Expresa la intuición de que los Objetos hacen las cosas según la información que tienen, refuerzan el encapsulamiento y esto redunda en bajo acoplamiento.
- *Controlador:* Un Controlador es un Objeto que no pertenece a la interfaz de usuario. Es aconsejable utilizar Controladores de Fachada cuando no existen demasiados eventos del sistema es aconsejable además utilizar Controladores de Casos de uso cuando la aplicación es muy extensa. De esta forma en vez de tener un solo Controlador Fachada y saturarlo, tenemos Controladores más pequeños especializados en cada Caso de uso especificado por el analista. [12]

2.6 GAUSSIAN98

Gaussian funciona tomando la información que necesita para realizar los cálculos de un fichero de entrada cuyas extensiones puedes ser .inp, .com ó .gjf. El resultado de estos cálculos es guardado en un fichero de salida con extensión .log. El químico debe ser capaz de crear los ficheros de entrada e interpretar el de salida. En el fichero de salida se encuentra toda la información que arroja el Gaussian luego de la terminación del cálculo, si existe algún error durante la ejecución este queda plasmado en el fichero .log. Se debe definir en el fichero de entrada la creación de un fichero chk (checkpoint) donde se almacenará paso a paso lo que vaya aconteciendo durante la ejecución, de manera que quede siempre guardado el último procesamiento, esto es de suma importancia pues en caso de que exista algún problema con la computadora donde se esta ejecutando el cálculo no se perderá toda la información y se podrá retomar el cálculo a partir de lo guardado en el fichero chk. Hay que tener en cuenta a la hora de analizar los resultados devueltos que estos deben ser salvados o de lo contrario se perderán los mismos cuando se realice un nuevo cálculo.

CONCLUSIONES DEL CAPÍTULO

Para obtener un sistema que sea capaz de funcionar de forma rápida y eficiente, que cumpla con todos los requerimientos exigidos, además de que debe ser flexible a cambios, se deben utilizar las herramientas correctas para su desarrollo. Luego de analizar todas las posibilidades y valorar las necesidades se decide trabajar con la metodología y las herramientas siguientes:

- La herramienta CASE escogida para modelar el programa será Visual Paradigm ya que es de soporte multiplataforma, proporciona generación de código, ingeniería inversa, generación de informes y apoya los estándares mas recientes de JAVA y UML.
- Para mantener una compatibilidad con las normas de calidad del centro (UCI) y por ser un proceso de desarrollo de software capaz de ser aplicado a cualquier proyecto sin importar la magnitud del mismo, de abordaje ágil al proceso de desarrollo, de código abierto y aplicable a un conjunto amplio de plataformas se utilizará como metodología de desarrollo Basic Unified Process.
- Por consiguiente el lenguaje de modelado será UML, ya que implementa un lenguaje de modelado común para todos los desarrollos y la documentación que crea también es común, por lo que puede ser entendida por cualquier desarrollador que tenga conocimientos del lenguaje de modelado.
- Como IDE se escogió Eclipse pues utiliza como lenguaje de modelado JAVA aunque permite plugins para otros lenguajes, soporta programación orientada a objetos, es una herramienta de código libre y puede ser ejecutado en diferentes sistemas operativos como Windows y Linux.
- El lenguaje de programación que se utilizará será Java, aprovechando RMI y sus posibilidades, para la invocación a las tareas remotas, además por su posibilidad de uso gratuito, por ser simple, potente, seguro, orientado a objetos, robusto e interactivo.
- Se utilizará como patrones de diseño los patrones GRASP pues la implementación del sistema se hará sobre programación orientada a objetos.
- Se utilizará la plataforma JDCS para distribuir los cálculos de la aplicación Gaussian98 compilado en Cuba.

CAPÍTULO 3
DESARROLLO Y RESULTADOS

En el presente capítulo se exponen los elementos imprescindibles para una solución exitosa: requerimientos funcionales y no funcionales, casos de uso del sistema, actores que los inician, se describe la solución y se muestran los resultados estadísticos obtenidos con las pruebas realizadas al sistema.

3.1 INGENIERÍA DEL SISTEMA

Para desarrollar un sistema es necesario realizar el análisis y diseño del mismo, para así tener una base a la hora de realizar la selección de las herramientas a utilizar durante el desarrollo, además deberán analizarse los requisitos funcionales y no funcionales del sistema a implementar que servirán como guía durante el desarrollo del software.

3.1.1 REQUERIMIENTOS FUNCIONALES

El levantamiento de requisitos funcionales se realiza con el objetivo de tener bien claras las funcionalidades que debe tener el sistema para que, basado en la calidad y la eficacia, este sea capaz de satisfacer las necesidades del cliente.

El sistema deberá cumplir con las siguientes funcionalidades:

R1. Gestionar Cálculos.
- 1.1. Realizar cálculos químicos - teóricos.
- 1.2. Buscar problemas en ejecución.
  - 1.2.1. Interrumpir la ejecución.
  - 1.2.2. Mostrar estado de la ejecución.
- 1.3. Buscar resultados obtenidos.
  - 1.3.1. Descargar solución.
  - 1.3.2. Eliminar solución.

3.1.2 REQUERIMIENTOS NO FUNCIONALES

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Representan las características del producto.

Usabilidad:

El sistema puede ser usado por cualquier tipo de personas que posea conocimientos básicos en el manejo de la computadora, se necesita contar con conocimientos especializados en química para entender los resultados dados por la aplicación.

Rendimiento:

El sistema está concebido para lograr un tiempo de respuesta del Gaussian98 más rápido, una mayor velocidad de procesamiento, y un mayor aprovechamiento de los recursos.

Soporte:

El sistema debe propiciar su mejoramiento y la anexión de otras opciones que se le incorporen en un futuro.

Seguridad:

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

Confiabilidad:

El sistema debe ser confiable y preciso en la información que le suministra al usuario para evitar cualquier tipo de error.

Software:

Se debe disponer de sistema operativo Linux para la instalación de la aplicación. Debe tenerse instalado el Java Runtime Environment (JRE) versión 1.5 o superior.

Hardware:

Para el desarrollo y puesta en práctica del proyecto se requieren máquinas clientes con los siguientes requisitos:

- Procesador Pentium 3 o superior.
- 256 MB ó más de RAM.

- 50 MB de capacidad del disco duro.
- Sistema Operativo Linux
- Conexión de red de área local.

3.1.3 Actores del Sistema a automatizar

Los actores del sistema pueden representar el rol que juega una o varias personas, un equipo o un sistema automatizado, interactúan con el sistema ya sea brindándole alguna información necesaria al mismo o tomándola de el.

Actores	Justificación
Químico	Representa el usuario que va a hacer uso del sistema, teniendo la posibilidad de interactuar con todas la funcionalidades de este.
Gaussian	Representa el programa que va a realizar todos los cálculos subidos a la plataforma por el usuario.

Tabla 1 Actores del Sistema

3.1.4 Casos de Usos del Sistema

Las descripciones de las funcionalidades con las que debe cumplir el sistema para garantizar el correcto funcionamiento del mismo son conocidas como casos de uso

Cod.	Nombre de caso de uso	Justificación de la selección.
1	Gestionar Cálculos	Recoge todo el proceso de los cálculos químicos deseados, obtener, guardar, eliminar y visualizar los resultados obtenidos.

Tabla 2 Casos de Uso del Sistema.

3.1.5 Diagrama de Casos de Uso del Sistema

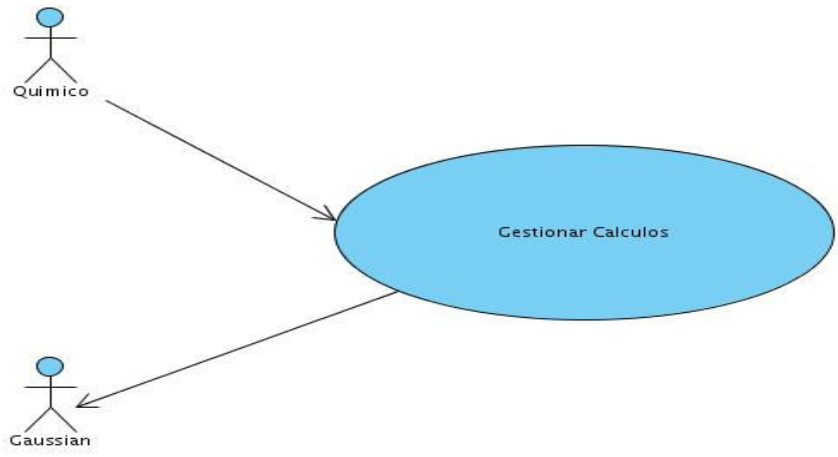


Ilustración 1 Diagrama de Casos de Uso.

3.1.6 Descripción de Casos de Uso del Sistema

Caso de Uso:	Gestionar cálculos.
Actores:	Químico, Gaussian.
Propósito:	Realizar cálculos químico – teóricos, brindar al químico la posibilidad de subir a la plataforma un problema nuevo, conocer el estado actual de la ejecución, detener la misma o ya una vez terminado el proceso de realización de los cálculos descargar o eliminar los resultados.
Resumen:	El caso de uso inicia cuando el químico se autentica y sube un problema a la plataforma JDCS, las PC clientes hacen peticiones de unidades de trabajo al servidor y el sistema selecciona y envía los datos necesarios para la realización de los cálculos, al concluir el proceso y el sistema recoja y muestre los resultados, el químico decide que hace con ellos terminando así el caso de uso.
Referencias:	R1.
Precondiciones:	PC cliente con sistema operativo Linux y Gaussian98 instalado correctamente.
Poscondiciones:	Recoger y entregar resultados a los químicos.
Flujo Normal de Eventos	

Acción del Actor	Respuesta del Sistema
1. El químico se autentica en el sistema. 3. Si el químico desea realizar un cálculo nuevo (Ver sección 1), si desea Buscar los problemas que se encuentran en el servidor y conocer el estado de cualquiera de ellos o detener la ejecución (Ver sección 2) y si decide Buscar los resultados de los cálculos realizados y descargar o eliminar cualquiera de ellos (Ver sección 3).	2. Si la autenticación es válida se le dará la posibilidad de crear un nuevo problema, conocer el estado de los que están en ejecución y si lo desea detener cualquiera de ellos, Buscar los resultados de los cálculos que ya han sido procesados y descargarlos o eliminarlos si así lo decide.
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
	2.1. Si el sistema no logra la conexión con la plataforma no se realiza ninguna operación.
Sección 1	
Acción del Actor	Respuesta del Sistema
2. El químico busca el fichero que desea en un directorio y lo sube a la plataforma. 4. El Gaussian realiza los cálculos.	1. Se muestra una interfaz que da la posibilidad al químico de subir el fichero necesario para la realización de un cálculo nuevo. 3. El servidor genera las unidades de trabajo y las distribuye a las máquinas cliente. 5. La PC cliente envía al servidor los resultados del cálculo realizado. 6. El servidor procesa los resultados y los visualiza.
Flujos Alternos	
Acción del Actor	Respuesta del Sistema

Sección 2	
Acción del Actor	Respuesta del Sistema
1. El químico solicita ver los problemas existentes en el servidor. 3. Si el químico solicita ver el estado de ejecución de un problema determinado. 5. Si el químico desea detener una ejecución, selecciona la opción Terminate.	2. Se muestra una interfaz con los problemas que están en ejecución y brinda la posibilidad de conocer el estado de ejecución del problema que desee o detenerlo. 4. El sistema muestra en una interfaz el estado de ejecución del problema seleccionado por el químico. 6. El sistema detiene la ejecución del problema seleccionado. 7. El sistema actualiza las ejecuciones en curso.
Flujos Alternos	
Acción del Actor	Respuesta del Sistema
Sección 3	
Acción del Actor	Respuesta del Sistema
1. El químico selecciona la opción visualizar resultados de los cálculos realizados. 3. Si el químico decide descargar la solución selecciona la opción Download. 5. Si el químico desea eliminar la solución selecciona la opción Delete.	2. El sistema muestra una interfaz con los resultados. 4. El sistema permite al usuario seleccionar el directorio donde desee descargar la solución, procediendo a efectuar la operación. 6. El sistema elimina del servidor la solución seleccionada y actualiza las soluciones.

Tabla 3 Descripción de Casos de Uso del Sistema.

3.1.7 DESCRIPCIÓN DE LA SOLUCIÓN PROPUESTA

En el presente trabajo, cuyo objetivo fundamental es implementar de forma distribuida la aplicación Gaussian98 debido a que los tiempos de respuesta que se obtienen al procesar los cálculos en una máquina independiente no son los suficientemente rápidos y exactos, se propone una posible solución a este problema científico que aprovecha al máximo los recursos que se poseen y cumpla con las características de la instituciones donde pueda ser utilizado el sistema a implementar. Dicha solución se desarrollará sobre la plataforma distribuida JDCS, en un ambiente multiplataforma, utilizando el lenguaje Java para su implementación, sistema operativo Linux debido a la necesidad de utilizar herramientas libres para garantizar el beneficio gratuito de la aplicación y extendiendo las clases DataManager y Algorithm de Java adaptándolas a lo que se quiere lograr en el sistema a desarrollar. Por lo que se propone realizarlo de la siguiente manera: Una vez que haya sido subido el fichero de entrada a la plataforma, el servidor mediante el método generateWorkUnit implementado en la clase DataManager genera las unidades de trabajo a procesar, cuando una máquina cliente con sistema operativo Linux realice su petición de trabajo el servidor envía, mediante un vector generado por el método generateWorkUnit, la unidad de trabajo a procesar a la máquina que realizó la solicitud, en esta se implementa la clase Algorithm cuya funcionalidad consiste en procesar ese vector enviado desde el servidor mediante el método processUnit, los resultados del procesamiento se envían en un vector al servidor donde son procesados por la funcionalidad processResult implementada en la DataManager, este último recibe como parámetros el ID de la unidad de trabajo procesada y el conjunto de resultados que devuelve processUnit. Si el cómputo termina, processResult debe devolver “True”, si no “False”, de ocurrir el primer caso el servidor comprime el directorio del problema y quita el cómputo del sistema. Al terminar este proceso de cálculo dentro del directorio de la solución se encentra un fichero .log que es donde se encuentran los resultados. En la interfaz visual se le dará al químico la posibilidad de descargar o eliminar estos resultados, así como de ver el estado actual de la ejecución durante el proceso de cálculo o la detención de la misma.

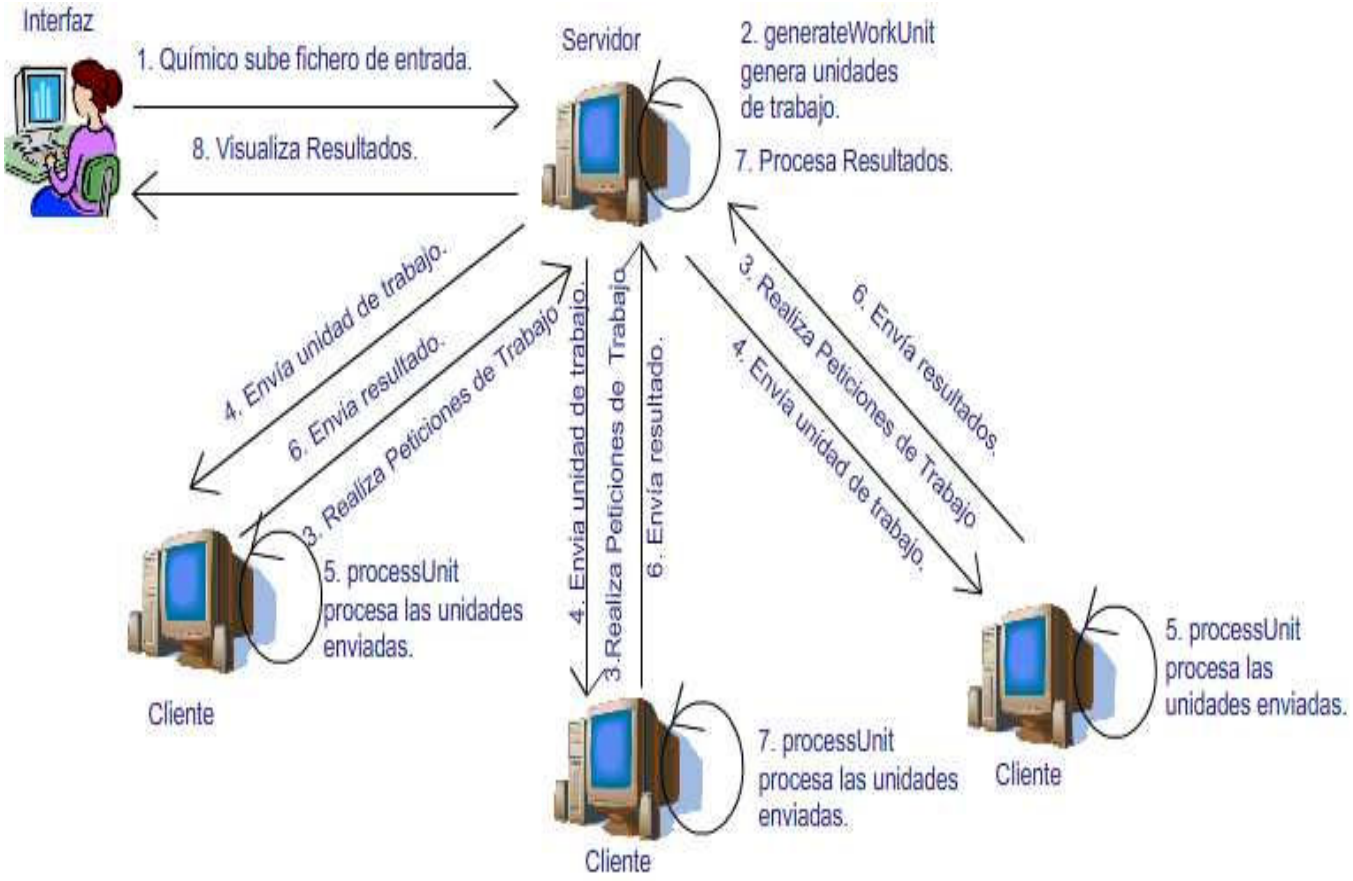
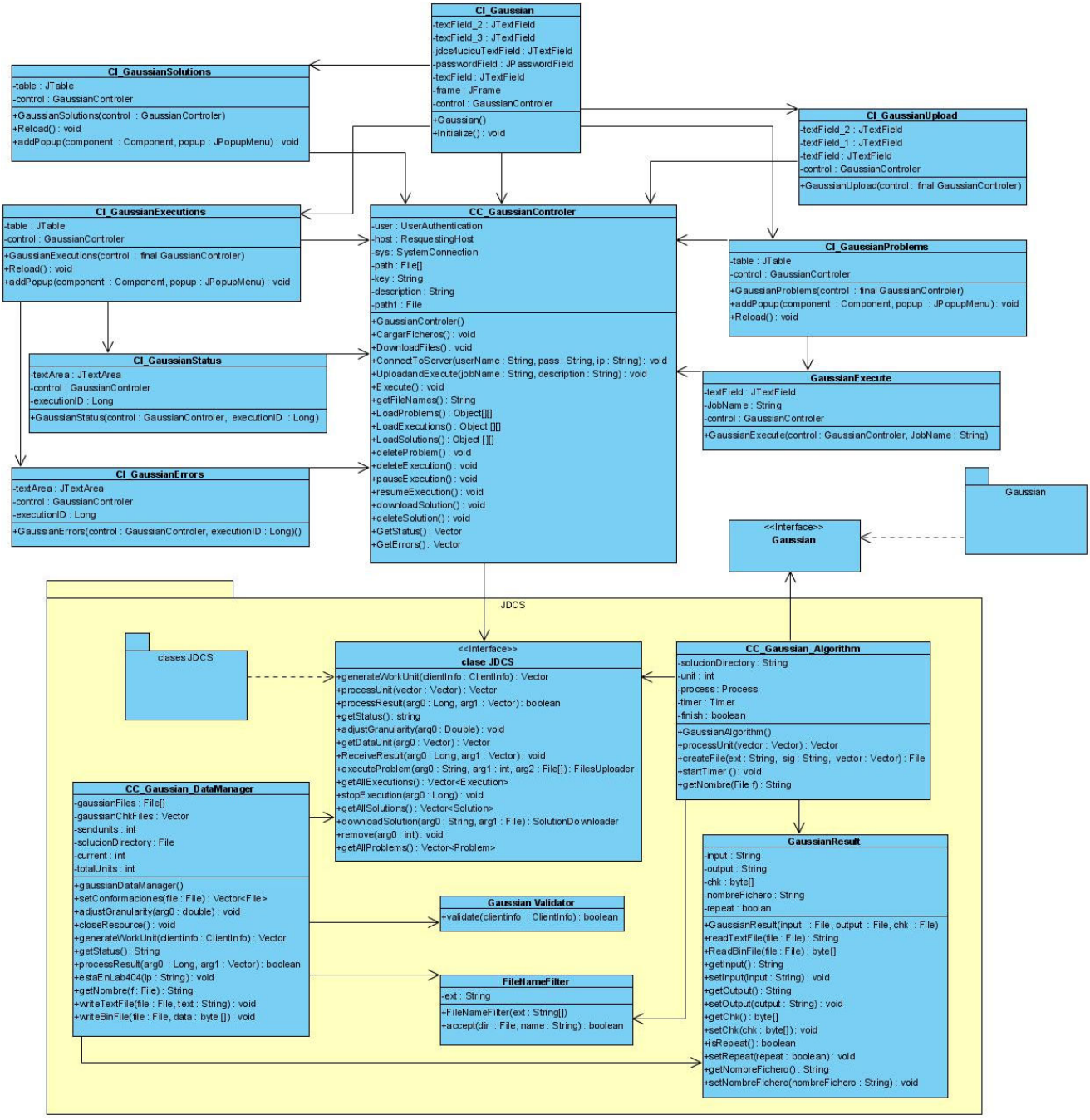


Ilustración 2 Solución Propuesta.

3.1.8 MODELO DE DISEÑO

Es un modelo que se encarga de la realización de los casos de uso. Basándose en los requisitos funcionales y no funcionales del sistema a implementar.

DIAGRAMA DE CLASES DEL DISEÑO.



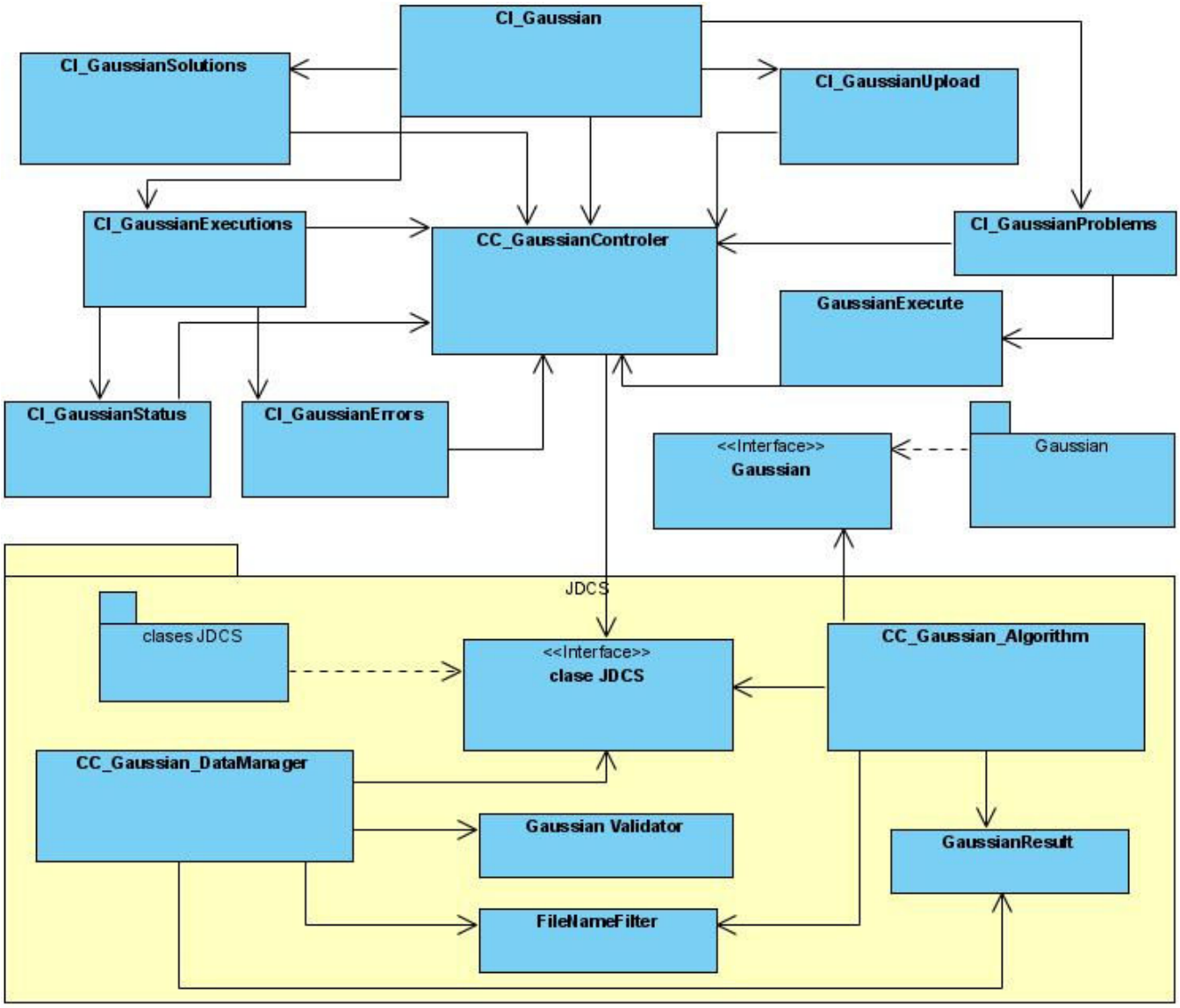


Ilustración 3 Diagrama de Clases del Diseño.

3.1.9 DIAGRAMAS DE SECUENCIA

Los diagramas de secuencia, al estar caracterizados por seguir un orden de pasos, brindan al usuario la posibilidad de entender de manera más sencilla los procesos que van teniendo lugar en cada escenario.

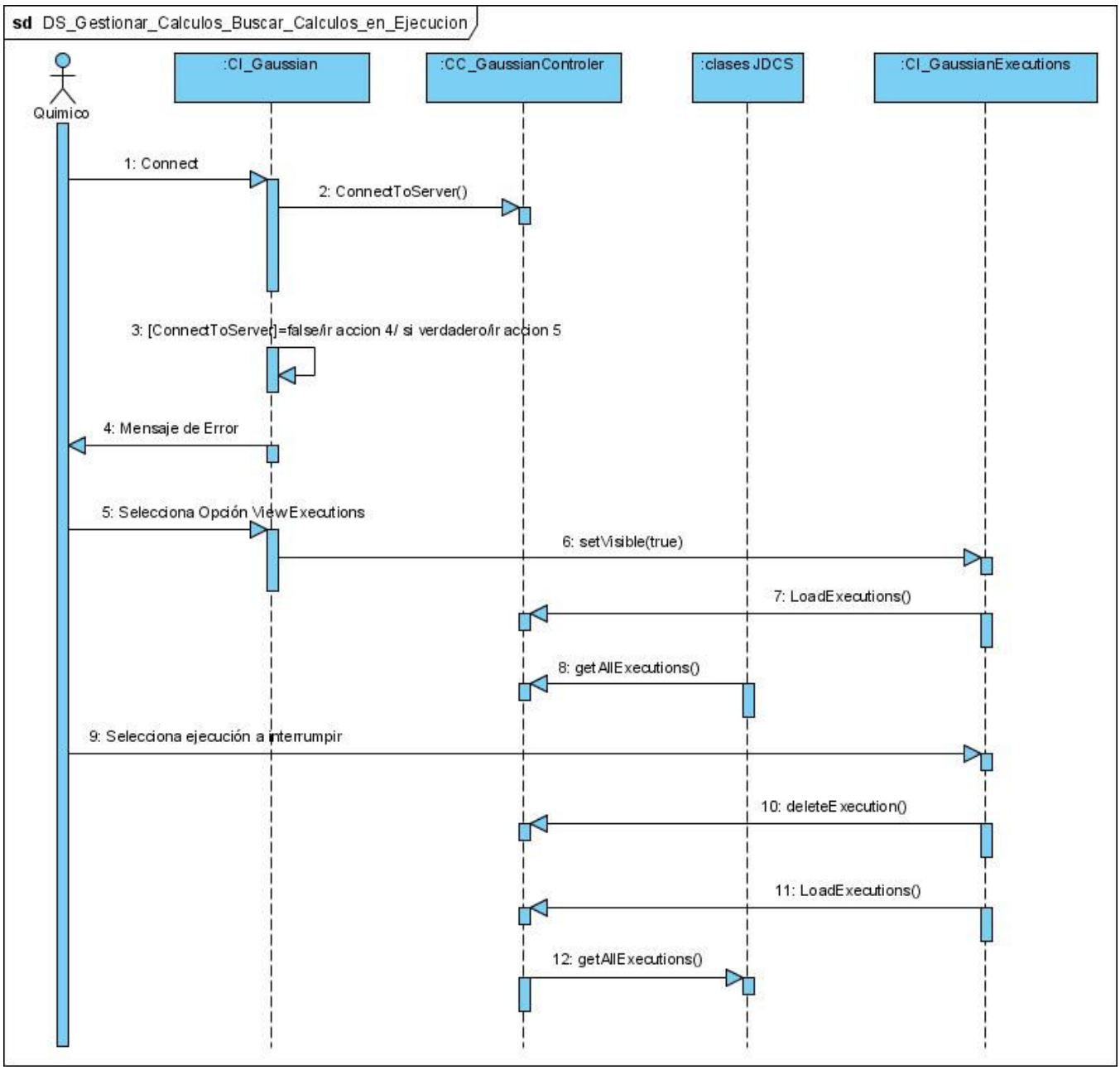


Ilustración 4 Diagrama de Secuencia Buscar_Cálculos_en_Ejecución.

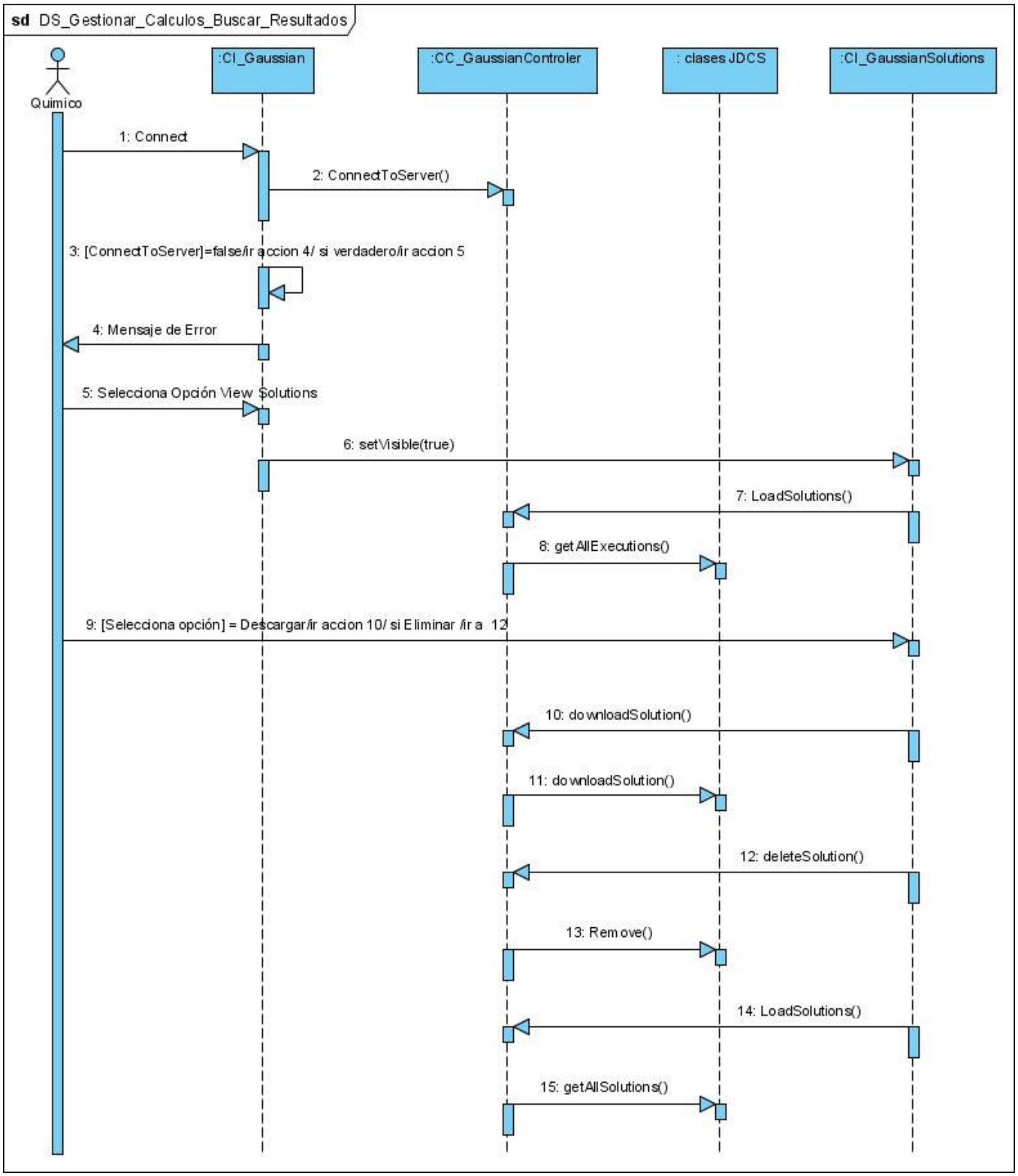


Ilustración 5 Diagrama de Secuencia Buscar Resultados.

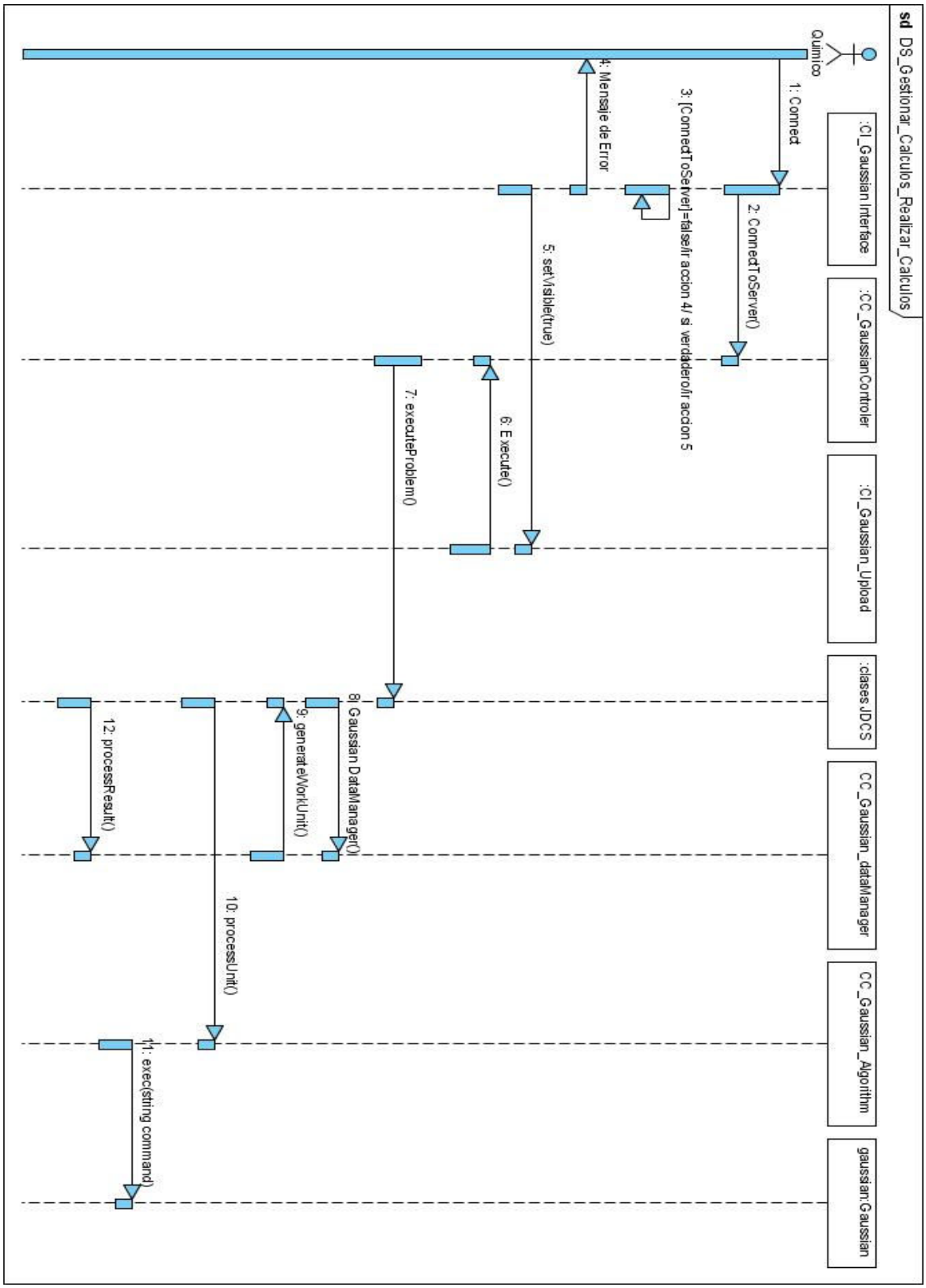


Ilustración 6 Diagrama de Secuencia Realizar Cálculo.

3.1.10 MODELO DE IMPLEMENTACIÓN

DIAGRAMA DE DESPLIEGUE

Ilustración 7 Diagrama de Despliegue.

DIAGRAMA DE COMPONENTES

Un diagrama de componente representa las dependencias entre los ficheros de código fuente además de guiar al equipo de programadores a la hora de comenzar la implementación.

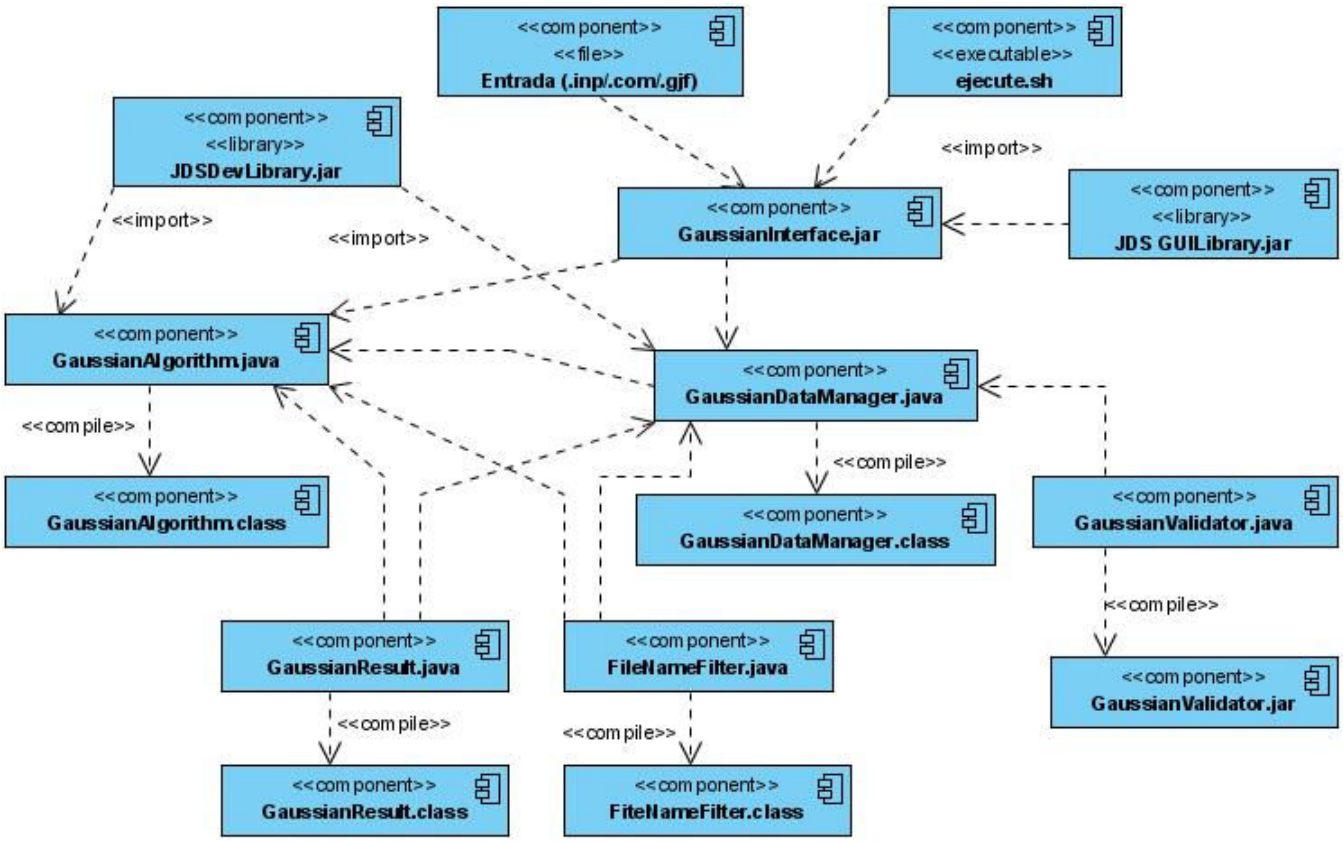


Ilustración 8 Diagrama de Componentes.

3.2 Resultados Experimentales

Con la culminación de la implementación del sistema solo queda por demostrar su rapidez, eficiencia y que realmente un sistema distribuido es la vía de solución correcta para el desarrollo del mismo. Para ello se realizaron pruebas al software donde se ejecutaron los cálculos en una PC independiente, de forma secuencial y luego se realizaron los mismos cálculos utilizando el sistema implementado distribuyéndolos sobre 30 máquinas aproximadamente. Para la ejecución utilizamos 2 ficheros de 100 y 60 conformaciones respectivamente. Al concluir los cálculos quedó evidenciado que en la máquina independiente, como se esperaba, los tiempos de respuestas fueron mayores que los realizados en el sistema implementado, logrando de esta manera afirmar que la utilización de un Sistema Distribuido como vía de solución ante el problema científico logró vencer el objetivo de este trabajo, obteniéndose resultados que lo patentan, los cuales se evidencian en las siguientes tablas de pruebas:

Primeramente se analizó secuencialmente un fichero formado por 100 conformaciones, cuyo tiempo de respuesta fue de 6 horas, posteriormente se realizaron pruebas en el sistema, analizando dicho fichero de forma distribuida en horas de la madrugada (03:30 AM) y en horas de la mañana (9:15 AM). Al concluir el proceso de cálculo se pudo apreciar que los tiempos de respuesta habían sido de 51 minutos en el horario de la madrugada y 1hora en el horario de la mañana, lográndose una reducción de 5 y 4 horas respectivamente, estas diferencias de tiempo pueden estar dadas por diferentes factores, como por ejemplo, que las máquinas con que cuenta la plataforma distribuida no son dedicadas, por lo que todo el tiempo no están disponibles para realizar peticiones de unidades de trabajo, además que la cantidad de ellas con Sistema Operativo Linux es escasa. En horas de la mañana, la tarde y el principio de la noche las ejecuciones demoran más que en el horario de la madrugada en terminar sus cálculos, pues en este tiempo los laboratorios están siendo utilizados en mayor medida, lo que trae como consecuencia la elevación de los tiempos de respuestas de la aplicación Gaussian98. No obstante a esto se logró demostrar que la utilización de un Sistema Distribuido es la vía correcta para dar solución al problema científico. Ver tablas 4 y 5.

Conformación	IP de PC Cliente donde fue procesada	Tiempo de procesamiento en Minutos
1	10.34.19.11	00:03:27:07
2	10.34.19.13	00:03:58:01
3	10.34.19.22	00:03:30:09
4	10.34.19.11	00:03:33:08
5	10.34.19.12	00:03:24:03
6	10.34.19.14	00:03:59:09
7	10.34.19.13	00:02:59:02
8	10.34.19.25	00:04:00:05
9	10.34.19.17	00:03:28:06
10	10.34.19.13	00:03:32:08
11	10.34.19.34	00:03:59:09
12	10.34.19.17	00:03:32:00
13	10.34.19.13	00:03:28:04
14	10.34.19.24	00:03:43:01
15	10.34.19.14	00:03:34:06
16	10.34.19.15	00:03:59:04
17	10.34.19.34	00:03:29:09
18	10.34.19.15	00:03:54:04
19	10.34.19.36	00:03:32:04
20	10.34.19.17	00:02:48:05
21	10.34.19.16	00:03:31:08
22	10.34.19.26	00:03:27:09
23	10.34.19.11	00:03:31:00
24	10.34.19.17	00:03:31:08
25	10.34.19.34	00:03:26:08
26	10.34.19.13	00:02:55:02
27	10.34.19.35	00:03:30:07
28	10.34.19.17	00:03:28:09
29	10.34.19.21	00:03:31:01
30	10.34.19.36	00:03:27:09
31	10.34.19.42	00:03:32:04
32	10.34.19.22	00:03:24:03
33	10.34.19.46	00:02:50:05
34	10.34.19.11	00:04:00:02
35	10.34.19.22	00:03:56:02
36	10.34.19.24	00:03:29:01
37	10.34.19.13	00:03:28:09
38	10.34.19.25	00:03:30:02
39	10.34.19.13	00:03:50:05

40	10.34.19.43	00:03:58:04
41	10.34.19.25	00:03:05:03
42	10.34.19.37	00:03:27:09
43	10.34.19.31	00:03:29:00
44	10.34.19.26	00:03:34:00
45	10.34.19.32	00:03:56:02
46	10.34.19.44	00:03:45:07
47	10.34.19.26	00:03:30:07
48	10.34.19.45	00:04:31:03
49	10.34.19.27	00:03:55:04
50	10.34.19.37	00:03:33:05
51	10.34.19.25	00:03:49:06
52	10.34.19.45	00:03:14:00
53	10.34.19.27	00:03:53:02
54	10.34.19.26	00:03:29:06
55	10.34.19.31	00:04:00:00
56	10.34.19.35	00:03:52:03
57	10.34.19.32	00:03:52:06
58	10.34.19.21	00:02:54:09
58	10.34.19.31	00:03:30:04
59	10.34.19.15	00:03:29:01
60	10.34.19.33	00:03:34:01
61	10.34.19.37	00:03:38:01
62	10.34.19.35	00:03:53:05
63	10.34.19.36	00:03:16:04
64	10.34.19.25	00:03:34:07
65	10.34.19.17	00:03:51:07
66	10.34.19.34	00:03:30:07
67	10.34.19.32	00:03:31:04
68	10.34.19.12	00:03:50:06
69	10.34.19.26	00:03:58:02
70	10.34.19.35	00:03:08:08
71	10.34.19.16	00:03:32:06
72	10.34.19.27	00:03:53:00
73	10.34.19.33	00:03:57:00
74	10.34.19.36	00:03:54:08
75	10.34.19.22	00:03:40:08
76	10.34.19.35	00:03:25:02
77	10.34.19.32	00:04:47:07
78	10.34.19.44	00:03:30:06
79	10.34.19.27	00:03:31:02

80	10.34.19.40	00:03:55:04
81	10.34.19.22	00:03:27:02
82	10.34.19.42	00:03:33:07
83	10.34.19.43	00:03:28:01
84	10.34.19.25	00:02:44:08
85	10.34.19.26	00:03:51:00
86	10.34.19.44	00:03:38:07
87	10.34.19.46	00:03:30:00
88	10.34.19.45	00:03:58:05
89	10.34.19.32	00:03:53:08
90	10.34.19.45	00:03:16:02
91	10.34.19.27	00:03:49:09
92	10.34.19.46	00:03:34:05
93	10.34.19.36	00:03:53:07
94	10.34.19.25	00:04:01:05
95	10.34.19.46	00:03:02:02
96	10.34.19.16	00:02:49:05
97	10.34.19.15	00:03:33:01
98	10.34.19.14	00:03:31:02
99	10.34.19.47	00:03:45:09
100	10.34.19.11	00:02:54:06
Tiempo Total:	**51 minutos**	

Tabla 4 Resultados de Prueba para Cálculo con Granularidad 1

Conformación	IP de PC Cliente donde fue procesada	Tiempo de procesamiento en Minutos
1	10.34.19.11	00:02:15:07
2	10.34.19.47	00:02:31:03
3	10.34.19.12	00:02:16:09
4	10.34.19.15	00:02:16:00
5	10.34.19.12	00:02:10:01
6	10.34.19.23	00:02:31:06
7	10.34.19.13	00:03:10:01
8	10.34.19.14	00:02:29:04
9	10.34.19.47	00:03:51:02
10	10.34.19.14	00:02:16:04
11	10.34.19.22	00:04:04:02
12	10.34.19.15	00:02:23:01
13	10.34.19.47	00:03:38:03
14	10.34.19.16	00:03:06:05
15	10.34.19.12	00:03:48:03

16	10.34.19.17	00:02:37:01
17	10.34.19.24	00:02:46:30
18	10.34.19.26	00:02:19:00
19	10.34.19.32	00:04:03:54
20	10.34.19.27	00:02:21:06
21	10.34.19.34	00:03:34:07
22	10.34.19.17	00:04:15:02
23	10.34.19.36	00:02:13:05
24	10.34.19.17	00:03:24:05
25	10.34.19.21	00:03:35:04
26	10.34.19.13	00:02:03:08
27	10.34.19.22	00:02:25:06
28	10.34.19.36	00:02:04:05
29	10.34.19.22	00:03:02:34
30	10.34.19.26	00:02:02:08
31	10.34.19.17	00:03:39:00
32	10.34.19.23	00:02:13:05
33	10.34.19.17	00:02:02:08
34	10.34.19.24	00:03:33:01
35	10.34.19.26	00:04:05:01
36	10.34.19.24	00:02:10:07
37	10.34.19.25	00:03:28:05
38	10.34.19.32	00:02:04:05
39	10.34.19.25	00:02:06:34
40	10.34.19.26	00:02:32:05
41	10.34.19.27	00:02:16:09
42	10.34.19.33	00:05:01:02
43	10.34.19.36	00:02:04:08
44	10.34.19.26	00:02:04:01
45	10.34.19.17	00:02:19:07
46	10.34.19.26	00.02:32.01
47	10.34.19.27	00:03:33:01
48	10.34.19.25	00:02:47:00
49	10.34.19.27	00:02:16:09
50	10.34.19.17	00:02:14:03
51	10.34.19.27	00:02:29:06
52	10.34.19.31	00:04:01:08
53	10.34.19.26	00:02:15:04
54	10.34.19.17	00:03:04:12
55	10.34.19.32	00:03:59:09
56	10.34.19.33	00:02:18:05

57	10.34.19.26	00:03:00:05
58	10.34.19.34	00:02:04:08
59	10.34.19.15	00:02:04:16
60	10.34.19.34	00:02:04:00
61	10.34.19.14	00:02:24:03
62	10.34.19.33	00:02:34:04
63	10.34.19.36	00:02:34:04
64	10.34.19.22	00:02:01:52
65	10.34.19.36	00:02:19:04
66	10.34.19.22	00:05:53:03
67	10.34.19.36	00:04:36:02
68	10.34.19.43	00:02:13:08
69	10.34.19.16	00:02:35:14
70	10.34.19.44	00:05:22:03
71	10.34.19.36	00:02:27:05
72	10.34.19.37	00:03:25:34
73	10.34.19.42	00:02:28:04
74	10.34.19.36	00:03:21:12
75	10.34.19.43	00:02:29:05
76	10.34.19.36	00:02:16:09
77	10.34.19.25	00:03:01:23
78	10.34.19.44	00:02:48:06
79	10.34.19.31	00:02:29:05
80	10.34.19.46	00:02:04:03
81	10.34.19.34	00:02:07:07
82	10.34.19.45	00:02:35:13
83	10.34.19.16	00:02:03:01
84	10.34.19.22	00:03:01:05
85	10.34.19.32	00:02:04.05
86	10.34.19.25	00:02:16:08
87	10.34.19.36	00:02:04:06
88	10.34.19.11	00:02:11:08
89	10.34.19.24	00:02:10:36
90	10.34.19.33	00:02:19:00
91	10.34.19.45	00:02:13:09
92	10.34.19.35	00:04:02.15
93	10.34.19.32	00:02:18:06
94	10.34.19.46	00:2:06:15
95	10.34.19.45	00:02:36:20
96	10.34.19.44	00:02:13:06
97	10.34.19.26	00:02:35:13:

98	10.34.19.43	00:02:34:01
99	10.34.19.47	00:03:00:05
100	10.34.19.24	00:02:16:09
Tiempo Total:	**1 hora**	

Tabla 5 Resultados de Prueba de Cálculo con Granularidad 1

Para lograr una mayor reducción en los tiempos de respuesta de los cálculos realizados sobre el sistema se particionaron los ficheros granularmente, para ello se escogió un fichero formado por 60 conformaciones, lo cual quedaría resumido en 12 ficheros de 5 conformaciones cada uno, cuando el cliente realiza la petición de trabajo es enviado uno de estos ficheros, por lo que se ejecutarían 5 cálculos en una misma PC. Esto podría ser una vía para no sobrecargar la Plataforma JDCS, las máquinas a utilizar serían menos y se probó que de esta manera el cálculo es reducido a menos tiempo.

Se estimó para el fichero en cuestión que si las PC clientes hicieran sus peticiones al mismo tiempo el sistema demoraría en realizar el cálculo el tiempo que demora una sola partición granulada, quedando en este caso entre 15 y 20 minutos. El proceso de cálculo demoró 46 minutos, demostrando así que la utilización de la granularidad representó una vía factible y rápida para lograr la reducción de los cálculos químico – teóricos. Ver Tabla 6.

Conformación	IP de PC Cliente donde fue procesada	Tiempo de procesamiento en Minutos
1	10.34.19.25	00:13:71
2	10.34.19.46	00:16:68
3	10.34.19.44	00:12:75
4	10.34.19.26	00:14:56
5	10.34.19.11	00:11:34
6	10.34.19.23	00:12:85
7	10.34.19.46	00:14:02
8	10.34.19.15	00:1703
9	10.34.19.34	00:13:47
10	10.34.19.43	00:13:37
11	10.34.19.27	00:11:25
12	10.34.19.26	00:18:01
Tiempo Total:	**46 minutos**	

Tabla 6 Prueba para Cálculo para Granularidad 5.

La siguiente gráfica muestra el comportamiento de la Velocidad de respuesta del Sistema durante la realización de las pruebas, demostrando de esta manera que la vía más factible para reducir los tiempos de respuestas es la aplicación de particionamiento por granularidad.

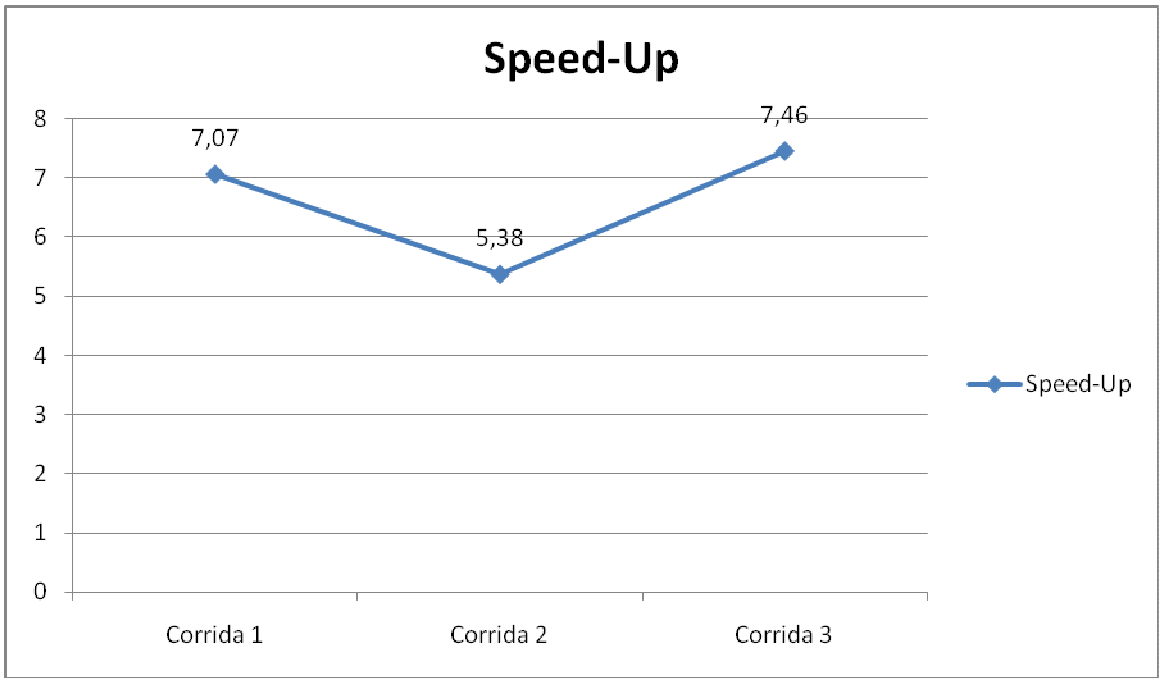


Ilustración 9 Gráfica de Speed-Up

CONCLUSIONES DEL CAPÍTULO.

En el presente capítulo se realizaron los diagramas de diseño, secuencia, despliegue y componentes para el sistema implementado, quedando así claramente expuesto el funcionamiento y la relación entre las diferentes clases que se implementan. Además se realizaron las pruebas que demostraron el correcto funcionamiento del sistema.

CONCLUSIONES GENERALES

Durante la realización de este trabajo se analizaron las posibles vías para dar solución al problema planteado, así como las herramientas, metodologías y lenguajes de programación que facilitarían y harían más eficiente el sistema implementado, atendiendo y siguiendo las normas establecidas se realizó:

- Un sistema que cumpliendo con los requisitos funcionales y no funcionales del mismo fuera capaz de gestionar cálculos químico-teóricos ante grandes volúmenes de datos a través de una plataforma distribuida.

- Mediante el diseño realizado se pudo lograr un sistema que ejecuta los cálculos y realiza diferentes operaciones, como descargar o eliminar los resultados de los cálculos procesados o saber en que estado se encuentra la ejecución y detenerla si es necesario, desde una interfaz gráfica.

- Al realizar las pruebas al sistema se demostró que la implementación distribuida de la aplicación Gaussian sobre la plataforma JDCS logró cubrir su objetivo: reducir los tiempos de respuestas de los cálculos realizados por dicha aplicación.

RECOMENDACIONES

Durante la implementación del sistema surgieron algunos inconvenientes por lo que recomendamos:

- Trabajar en aras de lograr una mayor estabilidad en la plataforma JDCS.
- Incrementar máquinas con sistema operativo Linux a la plataforma JDCS y garantizar la correcta instalación del Gaussian en cada una de ellas.
- Incorporar para una nueva versión del sistema Gaussian para Windows.
- Realizar deferentes pruebas al sistema utilizando el particionamiento por granularidad en mayor escala.

Bibliografía

1. **Vidal, Antonio M.** *Presente y Futuro de la Computación Paralela.*

2. **Keane, Thomas.** *Java Distributed System: Developer.* 2004.

3. **Galicia, Centro de supercomputación de.** *Comparativa Gaussian 03 / Gamess versión Sep. 7 2006 R4 / NWChem 5.0: Métodos y modelos implementados.*

4. **Gilberto Díaz, Vanessa Hamar, Herbert Hoeger, Victor Mendoza, Yubiryn Ramirez, Freddy Rojas.** *Herramientas GRID para la integración y administración de servicios de redes en Latino América.* Mérida, Venezuela : s.n., 2005.

5. **Rojas, Jaime Conde.** *Curso de Java Básico.* 2002.

6. **Álvarez, Ernesto Allén.** grid: la evolución de la informática distribuida.

7. **Rolando Menchaca Méndez, Félix García Carballeira.** *Java RMI.* 2001.

8. **Sun Microsystems, Inc.** *Java Remote Method Invocation.*

9. **John Wiley & Sons, Inc.** *TOOLS AND ENVIRONMENTS FOR PARALLEL AND DISTRIBUTED COMPUTING.* 2004.

10. *El lenguaje de Programación Java™.*

11. **Andrés Yanier Camejo Isaac, Adnier Turro Rodríguez.** *Módulo para el Cálculo Distribuido de Mecánica Molecular y Cuántica.* Julio, 2007.

12. **Informáticas, Grupo de Bioinformática Universidad de las Ciencias.** *Manual del Desarrollador: Plataforma de Cómputo Distribuido.* 2008.

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

14. **Kacsuk, Péter, Fahringer, Thomas y Németh, Zsolt.** *Distributed and Parallel Systems From Cluster to Grid Computing.* 2007.

15. **S. Tanenbaum, Andrew.** *Distributed Operating Systems.*
16. *Impacto de la Bioinformática en las ciencias biomédicas.* **Perezleo Solórzano, Ligeya, y otros.** La Habana : Ciencias Médicas, 2003.

17. LABORATORIO DE QUÍMICA COMPUTACIONAL Y TEÓRICA. [En línea] [Citado el: 8 de noviembre de 2007.] http://karin.fq.uh.cu/mmh/.

18. **E.M, Virgós.** MATEMATICALIA . [En línea] [Citado el: 20 de enero de 2008.] http://www.matematicalia.net/index.php?option=com_content&task=view&id=293&Itemid=191..

19. **Camejo Isaac, Andrés Yanier y Turro Rodríguez, Adnier.** *Módulo para el cálculo distribuido de mecánica molecular y mecánica cuántica.* La Habana : s.n., 2007.

20. **Coulouris, George.** *Sistemas Distribuidos.* Madrid : s.n., 2001.

21. **Aguilera Mendoza, Longendri**. *Sistema de cómputo distribuido aplicado a la Bioinformática.* 2008.

REFERENCIAS BIBLIOGRÁFICAS

1. **Rojo, Oscar J.** *Introducción a los Sistemas Distribuidos.* 2003.

2. Tutorial de Sistemas Distribuidos. [En línea] Instituto Tecnológico de La Paz, Dpto. de Sistemas y Comunicación, Programa Elaboración de Tutoriales., 6 de Junio de 1999. [Citado el: 11 de Diciembre de 2007.] http://sistemas.itlp.edu.mx/tutoriales/sistsdis.

3. **Pedro Fletes Guñido, J. Reyes Benavides Delgado.** Totorial de Sistemas Distribuídos. Fundamentos de Sistemas Distribuídos. [En línea] Instituto Tecnológico de Colima, 2004. http://www.itcolima.edu.mx/profesores/tutoriales/sistemas_distribuidos_I/index.html.

4. **Martín, Ignacio.** *Propuesta para la Creación de un Programa de e-Ciencia.* 2003.

5. CeCalcula. [En línea] 2003. http://www.cecalc.ula.ve/.

6. **Addison Wesley, Ed.James Rumbaugh, Ivar Jacobson y Grady Boonch Manual de Referencia.** *El Lenguaje Unificado de Modelado.* 2000.

7. Artículos CENDESI (Centro de Desarrollo de sistemas de información). [En línea] 30 de Septiembre de 2007. http://www.cendesi.com/site/es/articles.php?lng=es&pg=11.

8. **Vizcaíno, Felix ÓscarGarcía , Ismael Caballero.** *Prácticas Ingeniería del Software 3. Una Herramienta Case para ADOO: Visual Paradigm. Analisis y Diseño Orientado a Obtejos.*

9. **Valls., Guillem Rull Fort y Jose María Rodríguez.** [En línea] http://tryke.blogcindario.com/ficheros/html/rmi.html.

10. **Ignacio Blanquer, Vicente Hernández.** *Las Tecnologías ntacGrid en el ambito de la Salud.*

11. **Valle, Jose Gulliermo. Definición arquitectura cliente servidor.** [En línea] 2005. http://www.monografias.com/trabajos24/arquitectura-cliente-servidor/arquitectura-cliente-servidor.shtml.

12. **Larman, Craig.** *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* 17 de Diciembre del 2007.

13. Percepciones acerca de la programación ágil y xp. [En línea] 14 de Junio de 2007. [Citado el: 25 de Enero de 2008.]

14. **Sánchez, María A. Mendoza.** *Metodologías de Desarrollo de Software.* Junio 4 del 2004.

ANEXOS

Anexo #1 Descripción de las Clases del Sistema

Nombre: CC_GaussianAlgorithm	
Tipo de Clase: Controladora	
Descripción: Clase que corre en el cliente. Es la encargada de procesar las unidades de trabajo generadas por el método generateWorkUnit() de la clase GaussianDataManager.	
Atributo	**Tipo**
solutionDirectory	String
Unit	Int
Process	Process
Timer	Timer
Finish	Boolean
Responsabilidades:	
Nombre	**Descripción**
processUnit(Vector vector)	Recibe como parámetro las unidades de trabajo generada método generateWorkUnit() de la clase GaussianDataManager en un Vector y son procesados. Los resultados de cada unidad de trabajo se devuelven como vector y se retornan posteriormente al método processResults de la clase Gaussian Data Manager.
createFile(String ext, String sig, Vector vector)	Encargado de crear los ficheros de entrada del Gaussian98 en el cliente con la extensión correcta.
startTimer()	Responsable de controlar el tiempo que demora el procesamiento de la unidad

Tabla 7 Descripción de Gaussian Algorithm

Nombre: CC_GaussianDataManager
Tipo de Clase: Controladora
Descripción: Clase que corre en el Servidor. Su propósito es generar todas las unidades de trabajo que se enviarán a los clientes, procesar todos los resultados devueltos por los clientes, monitorear y ajustar el particionado de las unidades del trabajo, generar la información de estado del problema que

se devolverá vía interfaz remota, y terminar el cómputo distribuido.	
Atributo	**Tipo**
gaussianFiles	File[]
gaussianChkFiles	Vector
solutionDirectory	File
problemLog	Logger
Current	Int
sendUnits	Int
totalUnits	Int
Responsabilidades:	
Nombre	**Descripción**
GaussianDataManager()	Constructor de la clase. Crea el fichero donde se guardará el progreso del programa (problemLog), valida que los ficheros de entrada tengan la extensión correcta y que no estén vacíos, y crea el directorio donde se guardarán las soluciones.
generateWorkUnit(ClientInfo clientInfo)	Es llamado por el sistema cada vez que un cliente solicita una unidad del trabajo para procesar. Su misión es generar una unidad del trabajo y guardarla en un Vector que será recibido por la clase GaussianAlgorithm listo para procesar. Si no quedan unidades de trabajo actualmente disponibles, debe devolver null.
processResults(Long aLong, Vector vector)	Es llamado cada vez que un cliente termina de realizar una ejecución. Recibe el ID de la unidad de trabajo procesada y el conjunto de resultados. Una vez terminado el debe retornar true, si no false. Cuando devuelve true, el servidor elimina la ejecución y comprime el directorio de funcionamiento del problema para ser descargado por el usuario.
adjustGranularity(double d)	Es llamado periódicamente por el servidor con un parámetro que corresponde al porcentaje (positivo o negativo) de cuanto debe ajustarse el particionado paralelo para que el promedio de tiempo de procesamiento sea óptimo.
getStatus()	Es llamado cada vez que la interfaz remota hace una petición de estado al servidor sobre un problema particular

closeResources()	Su propósito es cerrar cualquier recurso (ej. archivos, directorios, input/outputstreams) que pueden estar abiertos momentos antes que el problema se quite del sistema.

Tabla 8 Descripción de Gaussian DataManager

Nombre: FileNameFilter	
Tipo de Clase: Auxiliar	
Descripción: Establece un filtro para el tipo de fichero (extensiones) que acepta como entrada el Gaussian98.	
Atributo	**Tipo**
Ext	String[]
Responsabilidades:	
Nombre	**Descripción**
accept(File dir, String name)	Verifica que el/los fichero(s) de entrada cumple con alguna de las extensiones que se tienen en el arreglo de extensiones (ext), devolviendo true en caso positivo.

Tabla 9 Descripción de FileNameFilter

Nombre: GaussianValidator	
Tipo de Clase: Auxiliar	
Descripción: Establece un filtro para el tipo de fichero (extensiones) que acepta como entrada el Gaussian98.	
Atributo	**Tipo**
serialVersionUID	static final long
Responsabilidades:	

Nombre	Descripción
validate(ClientInfo arg0)	Solo acepta PC clientes con Sistema Operativo Linux.

Tabla 10 Gaussian Validator

Anexo #2 Prototipos Funcionales

Ilustración 10 Interfaz_Gaussian

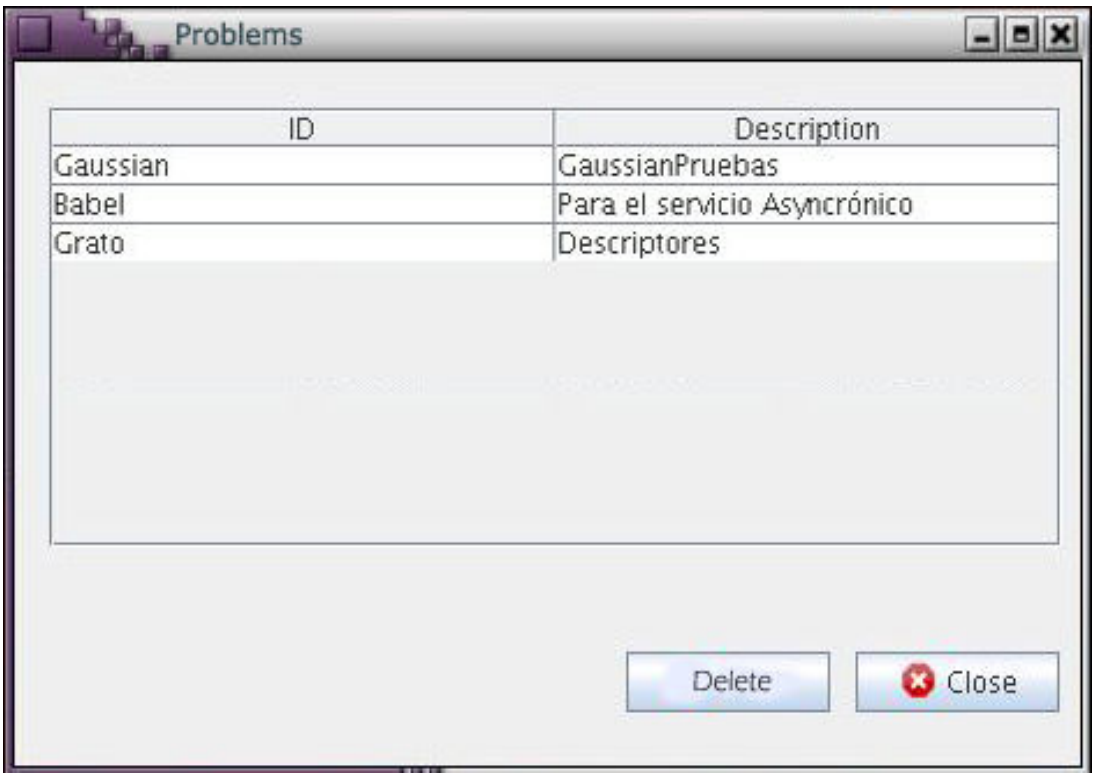


Ilustración 11 Problemas_Gaussian

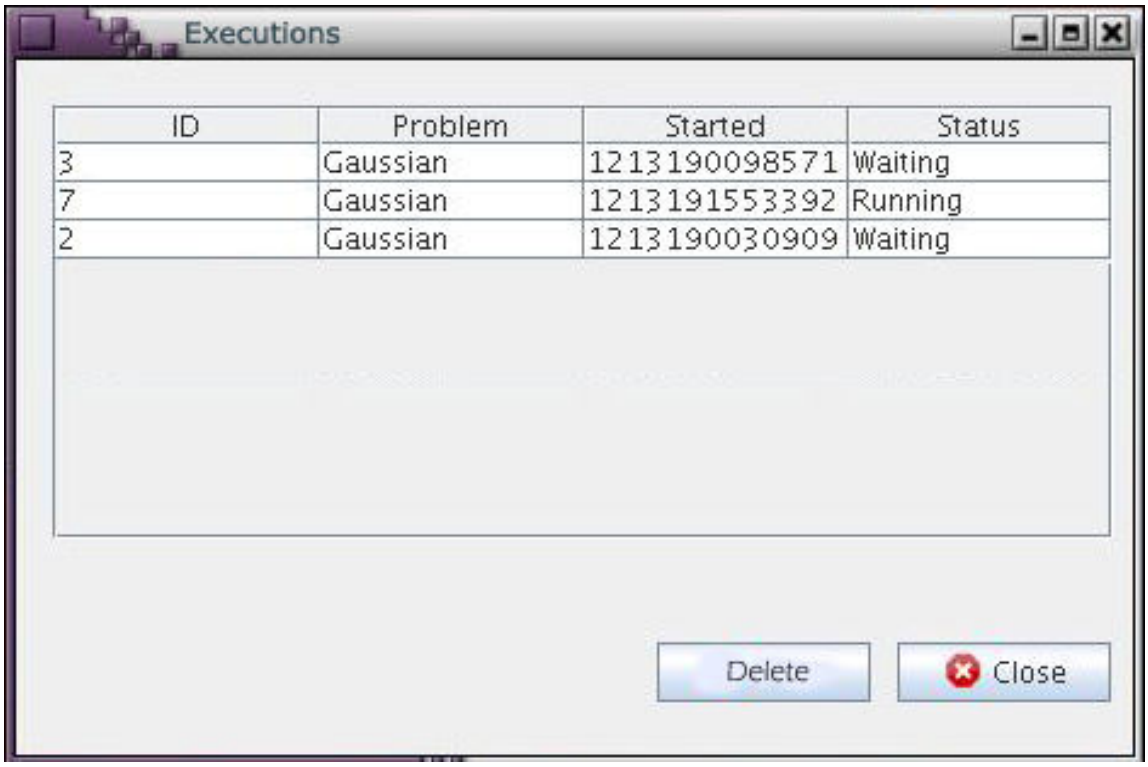


Ilustración 12 Ejecuciones_Gaussian

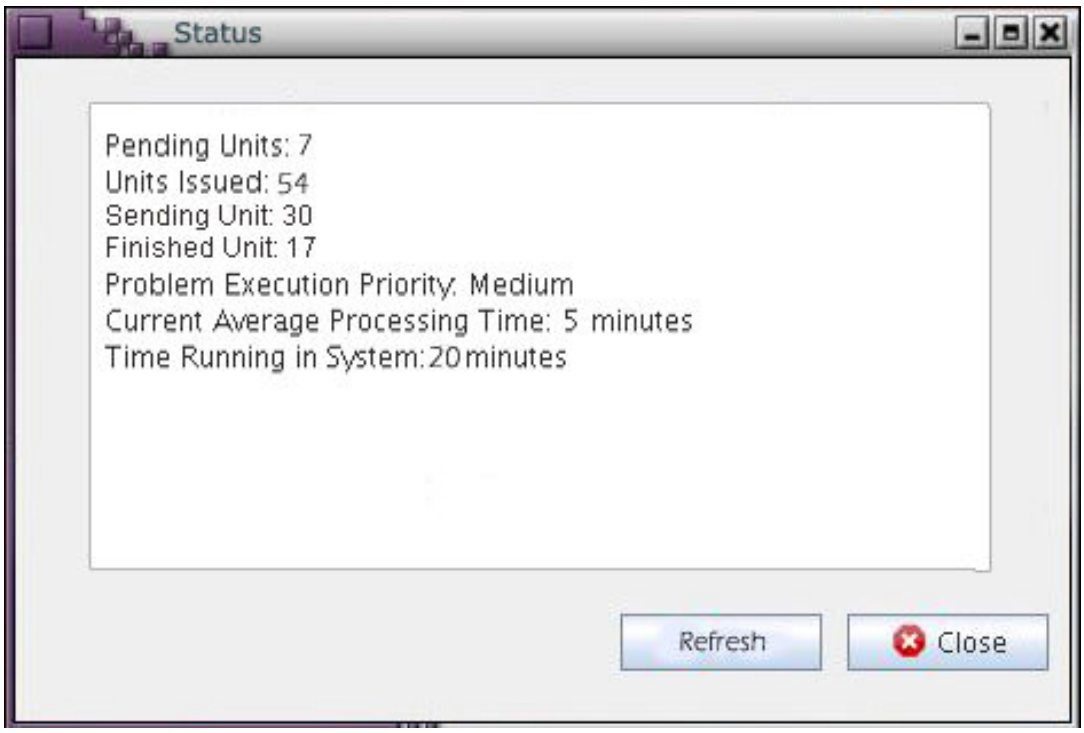


Ilustración 13 Estado_Gaussian

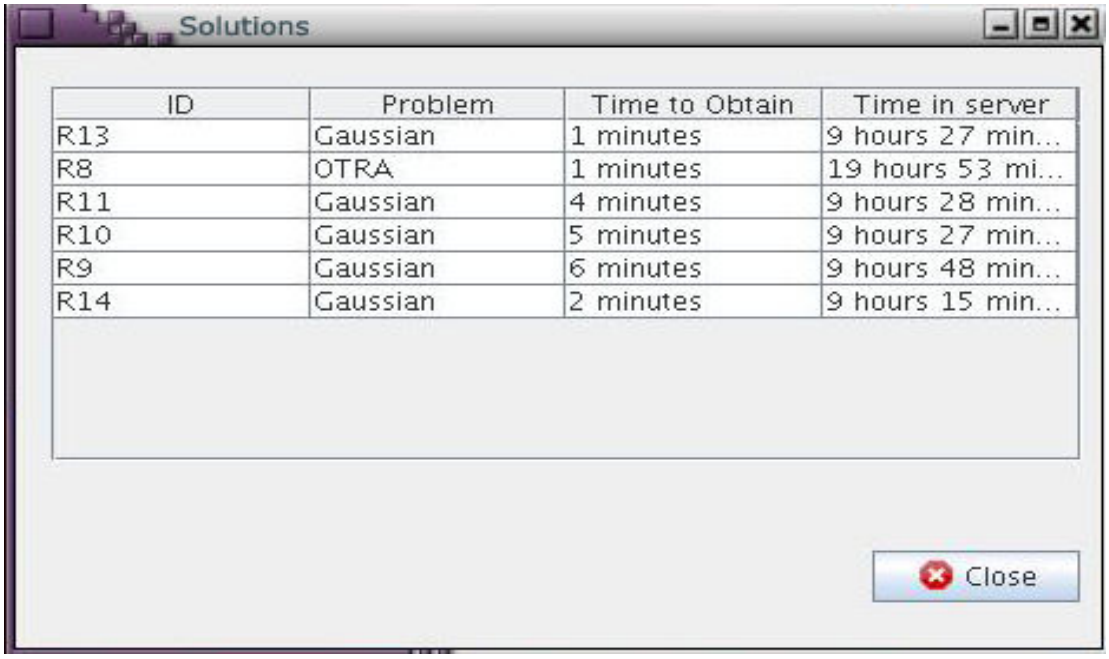


Ilustración 14 Soluciones_Gaussian

GLOSARIO DE TÉRMINOS

Ab-initio: *Son cálculos que se realizan aplicando formulas de la Termodinámica Estadística y la Mecánica Cuántica.*

Applets: *Un applet es un componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El applet debe ejecutarse en un contenedor, que lo proporciona un programa anfitrión, mediante un plugin.*

Browser: *Buscadores.*

Bytecodes: *Código intermedio más abstracto que el código máquina. Habitualmente es tratado como un fichero binario que contiene un programa ejecutable similar a un módulo objeto, que es un fichero binario producido por el compilador cuyo contenido es el código objeto.*

Caching: Procede de CACHE, que es una memoria rápida que se sitúa entre la memoria principal y la de almacenamiento masivo y que guarda referencias a sectores completos de memoria para que se incremente la velocidad del sistema, disminuyendo los accesos al disco duro.

Geant4**:** Es una herramienta informática para la simulación de detectores e interacciones de las partículas elementales con la materia.

Grid Computing**:** Es una tecnología innovadora que permite utilizar de forma coordinada todo tipo de recursos (entre ellos cómputo, almacenamiento y aplicaciones específicas) que no están sujetos a un control centralizado. En este sentido es una nueva forma de computación distribuida, en la cual los recursos pueden ser heterogéneos (diferentes arquitecturas, supercomputadores...) y se encuentran conectados mediante redes de área extensa (por ejemplo Internet).

Internetworks: *interredes.*

Link: *Enlaces.*

Pointer: *Es un tipo de datos de lenguaje de programación cuyo valor se refiere directamente ("a o indica") otro valor almacenado en otra parte en la memoria de ordenador que usa su dirección.*

Refactoring: *Técnica de la Ingeniería de Software para reestructurar un código fuente alterando su estructura interna sin cambiar su comportamiento externo.*

Struct: *Estructura que permite agrupar variables de varios tipos bajo un mismo nombre.*

Tripletes y Singletes: *Estados de excitación.*

typedef: *Es usada para proveer a tipos de datos existentes nombres nuevos, para conseguir que un programa sea más legible para el programador.*

Rmiregistry: *Herramienta del JDK.*

Union: *Es algo como una estructura de clase, pero con la superposición de campos en la memoria, y sólo uno de un juego de campos es activo a la vez. Por lo general hay un campo de etiqueta que dice cual campo es activo.*

#define: *Para la creación de macros.*