

**Universidad de las Ciencias Informáticas**

**Facultad 6**



# **Título: Sistema para Distribuir los Cálculos de la Aplicación Vega sobre la plataforma Java Distributed Computing System (JDACS)**

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

**Autores:**

Irina Sosa Cedeño

Ernesto Amek Duharte Peña

**Tutores:**

Ing. Liesner Acevedo Martínez

Ing. Rafael Arturo Trujillo Rasúa

**Ciudad de La Habana, Cuba**

**Junio, 2008**

**Declaración de Autoría**

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

Para que así conste firmo la presente a los \_\_\_\_ días del mes de \_\_\_\_\_ del año \_\_\_\_\_.

\_\_\_\_\_

Ernesto Amek Duharte Peña (Autor)

\_\_\_\_\_

Irina Sosa Cedeño (Autora)

\_\_\_\_\_

Lic. Rafael Arturo Trujillo Rasúa(Tutor)

\_\_\_\_\_

Lic. Liesner Acevedo Martínez (Tutor)



*"...La Revolución se hace a través del hombre, pero el hombre tiene que forjar día a día su espíritu revolucionario..."*

**Datos del Contacto**

**Tutor:** Lic Liesner Acevedo Martínez

[frodo@uci.cu](mailto:frodo@uci.cu)

**Tutor:** Lic. Rafael Arturo Trujillo Rasúa

[trujillo@uci.cu](mailto:trujillo@uci.cu)

**Autor:** Ernesto Amek Duharte Peña

[eaduharte@estudiantes.uci.cu](mailto:eaduharte@estudiantes.uci.cu)

**Autora:** Irina Sosa Cedeño

[isosa@estudiantes.uci.cu](mailto:isosa@estudiantes.uci.cu)

## *Agradecimientos*

*Difícil es el camino para llegar a dónde soñamos desde nuestros inicios escolares. Por fin ha llegado el momento de demostrar que somos capaces de enfrentarnos al mundo como profesionales. Este trabajo de diploma es la transición entre la vida de estudiantes a la vida profesional, y para su realización tuvimos que pasar momentos difíciles, y es precisamente en estos momentos difíciles donde hubo muchas personas que nos dieron su apoyo incondicional, es por ello que le dedicamos estas líneas como agradecimientos.*

*Agradezco a mis padres por sus buenos consejos y brindarme su apoyo incondicional.*

*A mi abuelita Bertha, que ha estado esperando siempre este momento, por quererme tanto y por guiarme por el buen camino.*

*A Wilmer por su amor, su apoyo y su paciencia. Gracias por aparecer en mi vida.*

*A mi tío, mi hermano y mi suegra Nelsa por preocuparse por mí y brindarme su ayuda en todo momento.*

*A mis abuelitos Puchi y Ramona que aunque no estén hoy presente, hicieron todo lo posible para que cada día fuera una mejor persona.*

*A todos mis familiares y amistades.*

*A todos, muchas gracias.*

*Irina*

*Agradezco especialmente a mis padres Paula y Ernesto por su apoyo y amor.*

*A todos mis familiares por su cariño.*

*A mi abuela por su dedicación.*

*A mis amistades, en especial a Lester, Roberto, Carlos, Nestor, Yixander, Yoamel, Yadir, Keiler, Yendry, Roberto por su apoyo y sus buenos consejo.*

*A todos muchas gracias porque sin ustedes no hubiese llegado hasta donde estoy hoy.*

*Ernesto*

### *Dedicatoria*

*Dedicamos este trabajo a nuestro Comandante en Jefe Fidel Castro, por la creación de tan importante proyecto y por la confianza que ha depositado en nosotros.*

*A nuestros padres por ser nuestros guías.*

*A todos los que esperaron ansiosos este momento e hicieron posible de una forma u otra que estubieramos hoy aquí.*

*A nuestras amistades por brindarnos lo mejor de ellos.*

## **Resumen**

Las investigaciones científicas en el campo de la bioinformática evolucionan constantemente en busca de soluciones óptimas para un rango grande de problemas que no pueden ser resueltos manualmente. Para soportar este desarrollo se necesitan de aplicaciones que sean más productivas y que aprovechen eficientemente los recursos computacionales con que se cuente. En el presente trabajo se propone un sistema para realizar cálculos químicos-teóricos a un número considerable de ficheros moleculares de forma distribuida utilizando la versión 2.2.0 de la aplicación Vega para Linux. El sistema se desarrolló en un ambiente multiplataforma, utilizando, para su implementación, el lenguaje de programación Java.

Durante todo el desarrollo del trabajo se utilizaron herramientas libres. Una de estas herramientas es Eclipse, que fue el entorno de desarrollo donde se realizó la programación y para la modelación gráfica se utilizó Visual Paradigm.



**Índice**

<b>Agradecimientos.....</b>	<b>I</b>
<b>Dedicatoria .....</b>	<b>III</b>
<b>Resumen.....</b>	<b>IV</b>
<b>Introducción .....</b>	<b>1</b>
<b>Capítulo 1: Estudios Preliminares.....</b>	<b>4</b>
<b>1.1 Sistemas Distribuidos .....</b>	<b>4</b>
1.1.1 Ventajas y Desventajas de los Sistemas Distribuidos .....	5
1.1.2 Ejemplos de Sistemas Distribuidos.....	6
<b>1.2 Herramienta VEGA.....</b>	<b>7</b>
<b>Capítulo 2: Materiales y Métodos .....</b>	<b>9</b>
<b>2.1 Lenguaje de modelado y metodología.....</b>	<b>9</b>
2.1.1 Metodología OPENUP (Basic Unified Process).....	9
2.1.2 Metodología RUP (Rational Unified Process).....	10
2.1.3 Lenguaje Unificado de Modelado (UML).....	11
<b>2.2 Herramientas de Modelado y Desarrollo .....</b>	<b>11</b>
2.2.1 Herramienta de modelado gráfico Rational Rose .....	11
2.2.2 Herramienta de modelado gráfico Visual Paradigm.....	11
2.2.3 Características Principales de Java.....	12
2.2.4 Entorno de Desarrollo Eclipse.....	12
<b>2.3 Plataformas Distribuidas.....</b>	<b>13</b>
2.3.1 Plataforma JDCS .....	13
2.3.2 BOINC (Infraestructura Abierta de Berkeley para la Computación en Red).....	13
<b>2.4 Java RMI (Remote Method Invocación) Invocación a Métodos Remotos. ....</b>	<b>13</b>
<b>2.5 Modelos Distribuidos.....</b>	<b>14</b>
2.5.1 Modelo Cliente-Servidor.....	14
<b>2.6 Conclusiones .....</b>	<b>15</b>
<b>Capítulo 3: Desarrollo y Resultados .....</b>	<b>16</b>
<b>3.1 Ingeniería del Software.....</b>	<b>16</b>

3.1.1 Requerimientos .....	16
3.1.2 Actores del Sistema .....	17
3.1.3 Casos de Uso del Sistema.....	17
3.1.4 Diagrama de Casos de Uso del Sistema .....	18
3.1.5 Descripción del Caso de Uso del Sistema .....	18
3.1.6 Patrón de Casos de Uso .....	21
3.1.7. Patrón de Diseño .....	21
3.1.8 Diagrama de Clases del Diseño.....	24
3.1.9 Diagrama de Interacción .....	25
3.1.10 Descripción de las clases.....	28
3.1.11 Diagrama de Despliegue.....	29
3.1.12 Diagramas de Componentes. ....	29
3.1.13 Algoritmos. ....	31
<b>3.2 Resultados Experimentales. ....</b>	<b>34</b>
3.2.1 Prueba para demostrar la eficiencia de los cálculos de forma distribuida. ....	34
<b>Conclusiones Generales .....</b>	<b>38</b>
<b>Recomendaciones .....</b>	<b>39</b>
<b>Bibliografías .....</b>	<b>40</b>
<b>Referencias Bibliográficas .....</b>	<b>42</b>
<b>ANEXOS.....</b>	<b>43</b>
<b>GLOSARIO DE TÉRMINOS.....</b>	<b>58</b>

## Índice de Figuras

<b>Figura 1: Centralizado: un servidor centraliza el acceso.</b> .....	4
Figura 2: Anillo: paso de un token. ....	4
Figura 3: Multicast: espera confirmación de todos. ....	5
Figura 4: Quórum: espera confirmación de “algunos”. ....	5
Figura 5: Topología Típica de Internet.....	7
Figura 6: Principales funciones en OPENUP y su interacción.....	9
Figura 7: Capas en OPENUP: micro-incrementos, iteración del ciclo de vida y proyecto de vida. ....	10
Figura 8: Fases y Flujos de trabajo propuestos por RUP. ....	10
Figura 9: Arquitectura Cliente/Servidor. ....	14
Figura 10: Diagrama de Caso de Uso. ....	18
Figura 11: Ejemplo del patrón Experto. ....	22
Figura 12: Ejemplo del patrón Creador.....	22
Figura 13: Ejemplo del patrón Alta Cohesión .....	23
Figura 14: Ejemplo del patrón Controlador .....	24
Figura 15: Ejemplo del patrón Bajo Acoplamiento.....	24
Figura 16: Diagrama de clases del diseño.....	25
Figura 17: Diagrama de secuencia. Sección Realizar Cálculo químico teórico.....	26
Figura 18: Diagrama de secuencia. Buscar Resultados Obtenidos. ....	27
Figura 19: Diagrama de secuencia. Buscar Problemas en Ejecución. ....	28
Figura 20: Diagrama de Despliegue .....	29
Figura 21: Diagrama de Componentes.....	30
Figura 22: Implementación del Método processUnit de la clase VegaAlgorithm.....	31
Figura 23: Implementación del Método getStatus de la clase VegaDataManager. ....	32
Figura 24: Implementación del Método generateWorkUnit de la clase VegaDataManager. ....	33
Figura 25: Implementación del Método processresult de la clase VegaDataManager.....	34
Figura 26: Tiempo de Procesamiento por Granularidad.....	35
Figura 27: Speed-Up obtenido en las pruebas realizadas.....	35
Figura 28: Procesamiento en 1 PC y Distribuido con glanularidad de 200.....	36
Figura 29: Prototipo de Interfaz de la CI_ Loguin. ....	55
Figura 30: Prototipo de Interfaz de la CI_Vega Visual.....	55
Figura 31: Prototipo de Interfaz de la CI_ Problemas en ejecución.....	56
Figura 32: Prototipo de Interfaz de la CI_Resultados.....	56
Figura 33: Prototipo de Interfaz de la CI_ Estado de Ejecución .....	57

**Índice de Tablas**

Tabla 1: Actores del Sistema ..... 17

Tabla 2: Caso de Uso del Sistema ..... 18

Tabla 3: Descripción del CU Gestionar Cálculo.....21

Tabla 5: Descripción de la Clase: Vega.....45

Tabla 6: Descripción de la Clase: CC\_Vegadatamanager. ....47

Tabla 7: Descripción de la Clase: CC\_Vegaalgorithm.....49

Tabla 8: Descripción de la Clase: CE\_Molecule.....50

Tabla 9: Descripción de la Clase: CE\_Resultado. ....51

Tabla 10: Descripción de la Clase: CI\_Resultado. ....52

Tabla 11: Descripción de la Clase: CI\_Execution\_problems. ....52

Tabla 12: Descripción de la Clase: CI\_Vegavisual. ....52

Tabla 13: Descripción de la Clase: CI\_loguin. ....53

Tabla 14: Descripción de la Clase: CI\_Ayuda. ....53

Tabla 15: Descripción de la clase: CC\_Visualcontrol .....54

Tabla 16: Descripción de la clase: CI\_Estado de ejecución. ....54

## **Introducción**

Las necesidades actuales de los proyectos vinculados a la Bioinformática requieren de una gran capacidad de almacenamiento y de cálculo. Para cubrir estas necesidades a nivel internacional, se han puesto en marcha grandes proyectos que utilizan la tecnología GRID y que tienen asociados temas relacionados con la bioinformática, tales como el proyecto Data Grid, InforSense BioSense Grid, PRAGMA (Pacific Rim Applications and Grid Middleware Assembly), IRISGrid, EGEE (Enabling GRIDS for e-Science in Europe), HealthGrid, entre otros.

Existen varias áreas en las investigaciones científicas que necesitan de la utilización de esta tecnología como es el caso del Análisis de secuencias y notación del genoma, la expresión de proteínas y las mutaciones de cáncer, la Biología Evolutiva computacional, la predicción de estructuras y funciones de biomoléculas, la preservación de la biodiversidad, la modelación de sistemas biológicos, entre otros. Para desarrollar estas investigaciones existen software o paquetes especializados en el cálculo químico-teóricos tales como el Vega, Gaussian, Mopac, Gamess, NWChem, entre otros [\[9\]](#).

Vega es una aplicación libre que permite trabajar con casi todos los ficheros moleculares sin importar el tamaño que estos tengan, incluyendo las proteínas. En nuestro país se ha utilizado esta aplicación en la facultad de Química de la Universidad de la Habana y se quiere incorporar al proyecto Grato que está patrocinado por el CITMA. La plataforma distribuida en la que se quiere montar el Vega es la plataforma JDCS desarrollada por nuestra facultad como una alternativa de cómputo que aglutina en un solo sistema computacional a un conjunto de estaciones de trabajo.

En nuestro país se le presta especial atención al estudio de los problemas que están relacionados con la medicina y la salud, por esta razón se cuenta con varios centros científicos que tienen la misión fundamental de resolver con calidad y rigor científico, problemas biomédicos y tecnológicos de gran importancia económica y social para el país y crear productos científicos de avanzada con capacidad competitiva en el mercado mundial. Como parte de la Batalla de Ideas se ha creado la Universidad de Ciencias Informáticas (UCI) con el objetivo de formar ingenieros informáticos bien preparados, así como la creación de software para la exportación y el desarrollo del país.

La Facultad de Bioinformática de la UCI está trabajando en el proyecto "Biogrid" en el cual se desarrolla el montaje de aplicaciones que permitan disminuir el costo y el tiempo de realización de cálculos químico-teóricos. Estos cálculos resultan muy costosos computacionalmente cuando se necesita procesar demasiadas moléculas, esto se ha convertido en un problema real para las investigaciones científicas, pues trae consigo pérdida de tiempo, lo cual retrasa y encarece los resultados.

Para dar solución al problema del tiempo y a los altos costos se ha venido aplicando los sistemas computacionales distribuidos. Aprovechando las ventajas que brindan estos sistemas en el uso de los recursos computacionales y la distribución de grandes volúmenes de información, se piensa que si se distribuyen los datos químicos que procesa la aplicación Vega sobre una red de computadoras es posible reducir los tiempos de respuesta.

Como parte del proyecto "Biogrid" se decide realizar un sistema para distribuir los datos químicos que procesa la aplicación Vega, aprovechando el estado ocioso y los recursos de varias máquinas conectadas en red para la realización de los cálculos químicos. Por esta razón se toma como **objeto de estudio** del presente trabajo los Sistemas Distribuidos para el procesamiento y cálculo masivo de información, y como **campo de acción** del mismo Plataforma JDCS para el cálculo químico-teórico.

En este ámbito, se define como **objetivo general** de la investigación: Desarrollar un sistema que permita distribuir los datos químicos que procesa la Aplicación Vega sobre la plataforma JDCS. Para dar cumplimiento a dicho objetivo es necesario alcanzar los siguientes **objetivos específicos**:

- Diseñar el sistema.
- Implementar el sistema.
- Evaluar la eficiencia del sistema implementado.

Para alcanzar dichos objetivos se planteó desarrollar las siguientes tareas:

- Estudio de la aplicación Vega para la realización del cálculo químico-teórico.
- Realización de búsquedas bibliográficas sobre los diferentes sistemas distribuidos.
- Estudio de la tecnología Java RMI (Invocación a Métodos Remotos).
- Estudio de la plataforma JDCS para el trabajo distribuido.
- Levantamiento de requisitos.
- Diseño del modelo de clases.
- Implementación del sistema.
- Realización de pruebas de tiempo de ejecución .

El documento está conformado por un Resumen, Introducción, tres capítulos que constituyen el cuerpo de la tesis, Conclusiones Generales, Referencias Bibliográficas, Bibliografías y Anexos. Los capítulos son los siguientes:

**Capítulo 1: Estudios Preliminares.**

Se presenta una breve descripción sobre los Sistemas Distribuidos para la solución de problemas que requieren el procesamiento de grandes volúmenes de información. Además se hace una caracterización de la herramienta de cálculo químico-teórico que se va a utilizar.

**Capítulo 2: Materiales y Métodos.**

Se presenta una breve descripción del lenguaje de modelado y la metodología que se va a emplear para el desarrollo del sistema, así como las herramientas de modelado y desarrollo. También se hará una descripción de la plataforma distribuida sobre la cual se ejecutará la aplicación Vega, así como del protocolo cliente-servidor.

**Capítulo 3: Desarrollo y Resultados.**

Se presenta todo el desarrollo de la ingeniería de software, incluyendo los patrones de diseño empleados para el desarrollo del sistema, así como el resultado de las pruebas realizadas al sistema para demostrar su eficiencia en cuanto a la reducción del tiempo de procesamiento.

## Capítulo 1: Estudios Preliminares

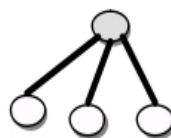
En este capítulo se presenta una breve historia de los Sistemas Distribuidos, así como algunas características fundamentales y ejemplos de estos sistemas. Además se hará una caracterización de la versión 2.2.0 de la aplicación Vega.

### 1.1 Sistemas Distribuidos

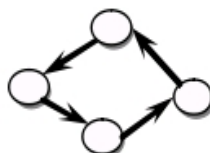
Los Sistemas Distribuidos son sistemas que utilizan los componentes de hardware y software de las computadoras conectadas a la red, para la realización de una tarea dada [1]. Estas computadoras se comunican mediante un protocolo cliente-servidor (ver Figura 9). Desde el inicio de la era de la computadora moderna (1945), hasta cerca de 1985, solo se conocía la computación centralizada. A partir de la mitad de la década de los ochentas aparecen dos avances tecnológicos fundamentales, el desarrollo de microprocesadores poderosos y económicos con arquitecturas de 8, 16, 32 y 64 bits y el desarrollo de redes de área local (LAN) de alta velocidad, con posibilidad de conectar cientos de máquinas a velocidades de transferencia de millones de bits por segundo (mb/seg). El surgimiento de estas redes trajo consigo el desarrollo de los Sistemas Distribuidos. El avance de estos sistemas ha propiciado un aumentando de la disponibilidad de computadoras personales de altas prestaciones y servidores para los Sistemas Distribuidos.

Estos Sistemas presentan varias características entre las que se encuentran:

- **Concurrencia:** Permite la utilización simultánea de los recursos disponibles en la red. El acceso a estos recursos se realiza a través de la Exclusión Mutua (ver desde Figura 1 hasta Figura 4).

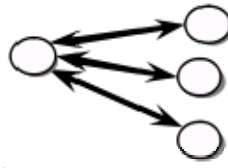


**Figura 1:** Centralizado: un servidor centraliza el acceso.

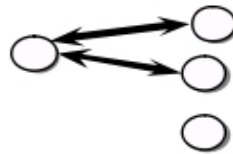


**Figura 2:** Anillo: paso de un token.





**Figura 3:** Multicast: espera confirmación de todos.



**Figura 4:** Quórum: espera confirmación de "algunos".

- **Carencia de reloj global:** cuando los programas necesitan cooperar, coordinan sus acciones mediante el intercambio de mensajes. La coordinación estrecha depende de una idea compartida del instante en el que ocurren las acciones de los programas. Pero hay límites donde las computadoras conectadas en la red pueden sincronizar sus relojes, no hay una única noción global del tiempo correcto. Esto es una consecuencia directa del hecho que la única comunicación se realiza enviando mensajes a través de la red.
- **Fallos independientes de los componentes:** Los fallos en la red producen el aislamiento de los computadores conectados a él, pero eso no significa que detengan su ejecución. De hecho, los programas que se ejecutan en ellos pueden no ser capaces de detectar cuando la red ha fallado o está excesivamente lenta. De forma similar, la parada de un computador o la terminación inesperada de un programa en alguna parte del sistema (crash) no se da a conocer inmediatamente a lo demás componentes con los que se comunica.

### 1.1.1 Ventajas y Desventajas de los Sistemas Distribuidos

#### Ventajas

##### Con respecto a Sistemas Centralizados:

- Una de las ventajas de estos sistemas es la economía porque los microprocesadores ofrecen mejor proporción precio/rendimiento que los mainframes.
- El trabajo en conjunto [\[1\]](#).
- Mayor confiabilidad porque si una máquina se descompone, el sistema puede sobrevivir como un todo.
- Capacidad de crecimiento incremental. Se puede incrementar la potencia del sistema adicionando procesadores al sistema según las necesidades.

##### Con respecto a PCs Independientes:

- Se pueden compartir recursos [\[1\]](#).

- El trabajo es más rápido porque un sistema distribuido puede tener mayor poder de cómputo que un mainframes.

#### **Desventajas**

- Existen una gran diversidad de criterios sobre el diseño, implementación y uso del software distribuido.
- Existen problemas en la red de comunicación debido a la pérdida de mensajes y la saturación en el tráfico.
- Pueden surgir problemas en la seguridad de la máquina al compartir los recursos.

#### **1.1.2 Ejemplos de Sistemas Distribuidos**

##### **INTERNET**

Internet es también un sistema distribuido muy grande. Permite a los usuarios, donde quiera que estén, hacer uso de servicios como el World Wide Web (www), el correo electrónico, y la transferencia de archivos. El conjunto de servicios es abierto, puede ser extendido por la adición de servidores y nuevos tipos de servicios.

Los proveedores de servicios de Internet (ISP's)[\[11\]](#) son empresas que proporcionan enlaces de módem y otros tipos de conexión a usuarios individuales y pequeñas organizaciones, permitiéndoles el acceso a servicios desde cualquier parte de Internet, así como proporcionando servicios como correo electrónico y páginas Web. Las intranets están enlazadas conjuntamente por conexiones troncales (backbones). Una conexión o red troncal es un enlace de red con una gran capacidad de transmisión, que puede emplear conexiones de satélite, cables de fibra óptica y otros circuitos de gran ancho de banda [\[12\]](#).

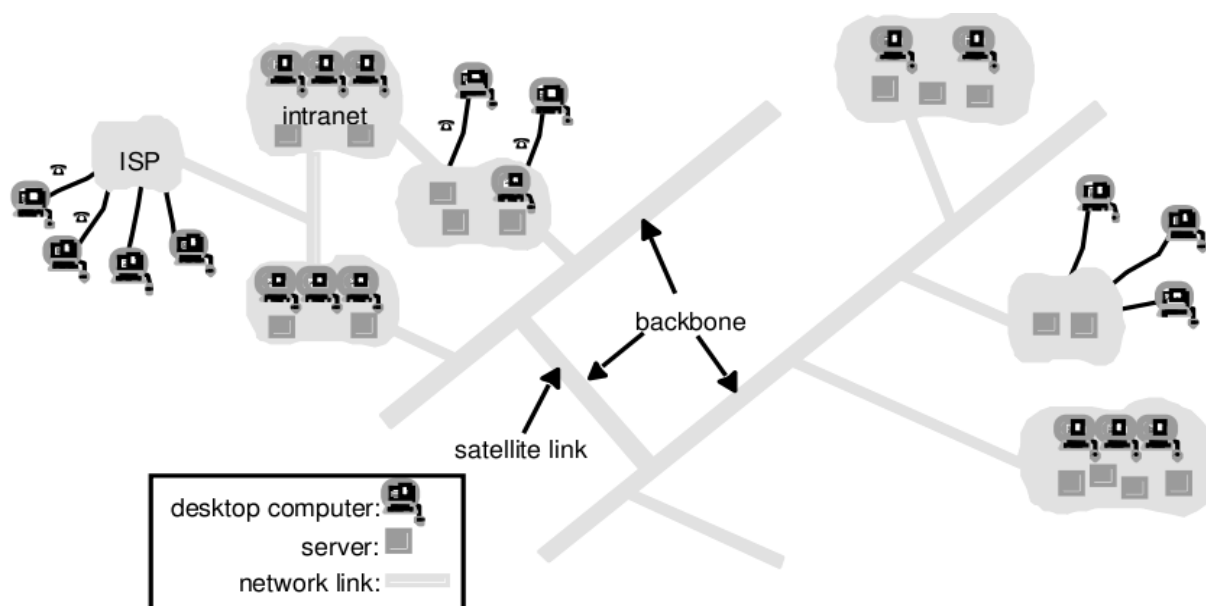


Figura 5: Topología Típica de Internet

## 1.2 Herramienta VEGA

La aplicación VEGA fue desarrollado por Alessandro Pedretti y Giulio Vistoli para crear un puente entre la mayoría de los paquetes de software moleculares como BioDock, Quanta/CHARMm, Insight II, Mopac. Ésta herramienta ha ido desarrollándose en el transcurso de los años hasta ser diseñada para completar modelos moleculares. VEGA está escrito en código portable (lenguaje estándar C) y puede ejecutarse en una gran cantidad de sistemas de hardware, simplemente recompilado el código fuente. El programa ya se ha probado en los siguientes sistemas operativos: IRIX (Silicon Graphics), Windows 9x/NT/2000/XP, Linux, FreeBSD, NetBSD [6].

Entre las funciones que tiene esta herramienta se encuentran:

- Cargar una plantilla.
- Calcular la energía de interacción de una constante dieléctrica.
- Calcular la energía para un residuo.
- Crear un archivo de salida en un formato de archivo específico.
- Calcular un mapa de la superficie con una sonda de radio diferente a la predeterminada.
- Mostrar una ayuda.
- Simular la solvatación de una molécula en cualquier disolvente.
- Pasar el control de la Información de las palabras clave para InfoXML y MopInt.
- Simular la adición y eliminación de hidrógenos.

- Puede especificar palabras clave para el análisis de la trayectoria.
- Normalizar las coordenadas atómicas.
- Puede especificar el nombre del archivo de salida.
- Definir el campo de fuerza a aplicar.
- Fijar el orden de enlace (ALL, RINGS).
- Cambiar el punto de densidad de superficie de un mapa.
- Cambiar la estructura secundaria de las proteínas.
- Agregar las cadenas laterales a una proteína.
- Simular la eliminación de todas las moléculas de agua presentes en el fichero molecular.

**Los formatos de entrada que permite el Vega son los siguientes:**

Alchemy, AMMP, Arc, BioDock, CAR, CIF, CHARMM CRD, CML, CML 2.0, CPMD XYZ, CRT, CHARMM DCD, Chem3D, ChemSol, CSSR, ESCHER NG, Fasta, GAMESS, Gaussian In/Out, Gromacs/Gromos mol, Gromacs XTC, HIN, IFF, Mol2, Mopac cartesian, Mopac internal, MDL, MDL V3000, MSF, PDB, PDBA, PDBF, PDBL, PQR, PQRXML, PSFX, QMC, Quanta CSR, RIFF, TINKER XYZ, XYZ.

**Los formatos de salida son los siguientes:**

- **Formato de Cálculo:** CVFF, Info, InfoXML.
- **Formatos de Mapas:** BiosymSrf, ComfaFld, CsvIIm, CsvLogP, CsvMep, CsvSrf, Quantallm, QuantaLogP, QuantaMep, QuantaSrf.
- **Formato de Moléculas:** Alchemy, AMMP, Biosym, ChemSol, CIF, CML, CML2, CPMDXYZ, CRD, CRT, CSSR, Fasta, GAMESS, GaussIn, Gromos, GromosNm, IFF, MdIMol, MdIMol3, mmCIF, Mol2, MopCar, MopInt, MSF, PDB, PDB2, PDBQ, PDBA, PDBF, PDBL, PDBNOTSTD, PQR, PQTXML, PSFX, QMC, OldBiosym, RIFF, XYZ.
- **Formato de Parcelas:** BinPlt, CSV, QuantaPlt. (Estos formatos de salida son útiles para el análisis de la trayectoria).
- **Virtual Reality Modeling Language (VRML):** Vrml, VrmlPts, VrmlCpk, VrmlSol (Estos formatos se crean con el fin de apoyar alguna publicación Web).
- **Formatos Packer:** bz2 (bzip2), gz (gzip), pp (PowerPacker), z (Z-Comprimir).

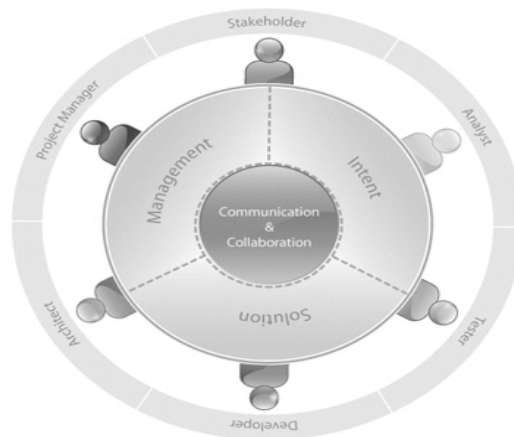
## Capítulo 2: Materiales y Métodos

En este capítulo se realiza un estudio de algunas herramientas y metodologías que pudieran ser usadas en la confección del sistema, además se hace una breve descripción de los lenguajes de modelado y programación con que se va a trabajar.

### 2.1 Lenguaje de modelado y metodología

#### 2.1.1 Metodología OPENUP (Basic Unified Process)

OPENUP es una versión simplificada de la RUP que está donado por IBM Rational para su inclusión en el Marco del Proceso de Eclipse (EPF) como fuente abierta. OPENUP conserva las características esenciales de RUP/Proceso Unificado, que incluyen el desarrollo iterativo, casos de uso y escenarios de conducción de desarrollo, la gestión de riesgos, y el enfoque centrado en la arquitectura. Debemos estar claros que el uso de OPENUP se debe realizar cuando se tiene un equipo pequeño, cuando se quiere evitar ser cargado excesivamente con metodologías formales improductivas.



**Figura 6:** Principales funciones en OPENUP y su interacción.

OPENUP proporciona una poderosa herramienta de aprendizaje y hace que sea más fácil encontrar la orientación pertinente, al señalar cuándo son mayores las posibilidades de llevar a cabo diversas tareas. Esto se hace a través de una visualización del proceso de aplicación que proporciona un tiempo base de organización de las tareas en el marco de un proyecto de vida. OPENUP estructura el ciclo de vida del proyecto en cuatro fases: Inicio, Elaboración, Construcción, y Transición.

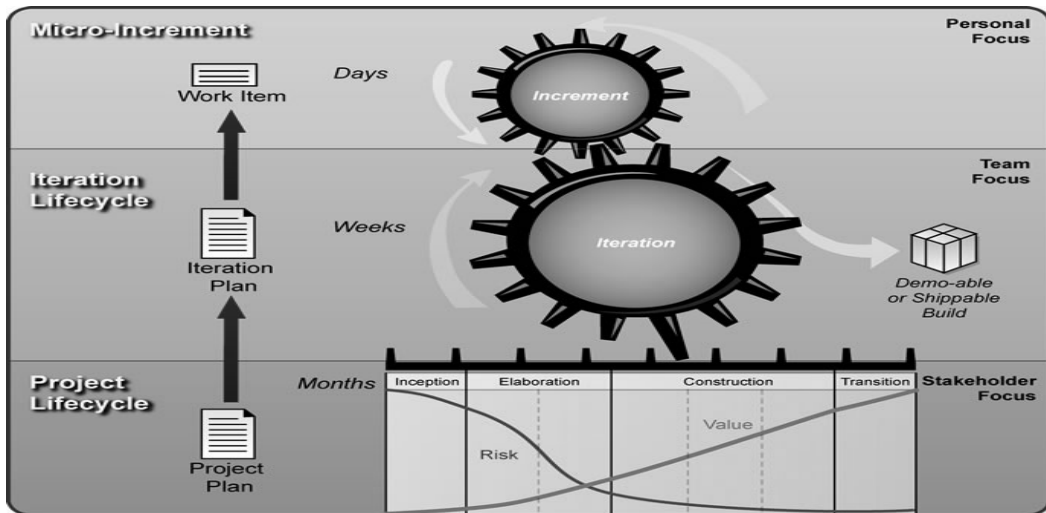


Figura 7: Capas en OPENUP: micro-incrementos, iteración del ciclo de vida y proyecto de vida.

### 2.1.2 Metodología RUP (Rational Unified Process)

La Metodología RUP es uno de los procesos más generales de los existentes actualmente, ya que está diseñado para adaptarse a cualquier proyecto [4]. RUP se basa en casos de uso para describir lo que se espera del software y está muy orientada a la arquitectura del sistema y es iterativo e incremental. RUP a diferencia de la metodología OPENUP es más apropiado para proyectos grandes, dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. Agrupa las actividades en grupos lógicos definiendo nueve flujos de trabajo principales (ver Figura 1), donde los seis primeros son conocidos como flujos de ingeniería y los tres últimos de apoyo.

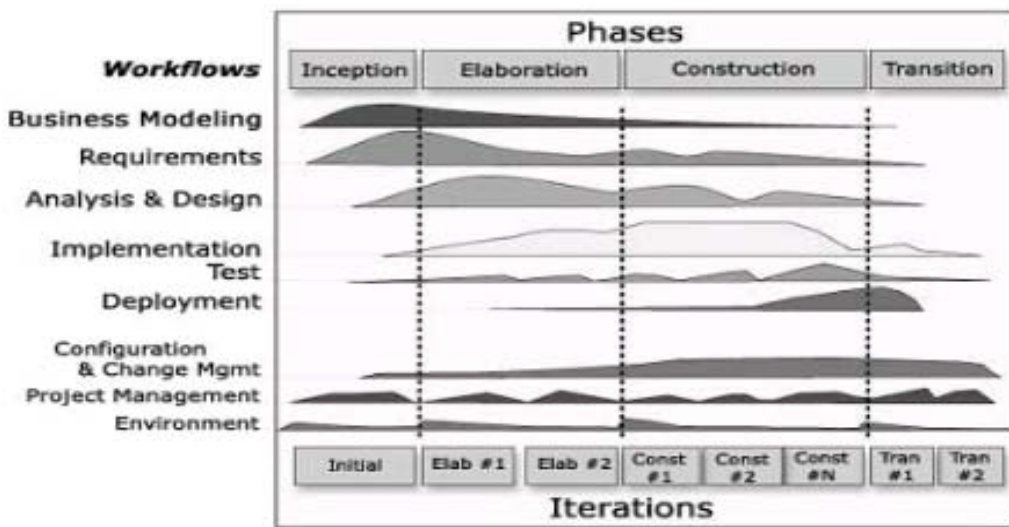


Figura 8: Fases y Flujos de trabajo propuestos por RUP.

### **2.1.3 Lenguaje Unificado de Modelado (UML)**

**UML** es un lenguaje que permite la modelación, construcción y documentación de los elementos que forman un sistema software orientado a objetos. UML ofrece un estándar para describir modelos, incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables. UML también puede considerarse como un lenguaje de modelado visual que permite una abstracción del sistema y sus componentes. Uno de los objetivos principales de la creación de UML era dar la posibilidad de intercambiar modelos entre las distintas herramientas CASE orientadas a objetos del mercado.

## **2.2 Herramientas de Modelado y Desarrollo**

### **2.2.1 Herramienta de modelado gráfico Rational Rose**

Rational es una herramienta que agrupa metodologías y herramientas que abarcan todos los aspectos del desarrollo de software, desde su concepción hasta la elaboración del producto. Permite la generación de código en Ada, ANSI C++, C++, CORBA, Java™/J2EE™, Visual C++ y Visual Basic a partir de modelos. Como todos los demás productos Rational Rose, proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente. Aunque Rational Rose es considerada la herramienta líder en el mundo de la modelación visual para el proceso de modelación del negocio, análisis de requerimientos y diseño de arquitectura de componentes, es un software propietario lo cual constituye una desventaja con respecto a Visual Paradigm.

### **2.2.2 Herramienta de modelado gráfico Visual Paradigm**

Visual Paradigm para UML es una herramienta profesional que soporta el ciclo de vida completo del desarrollo de software: Análisis y Diseño orientados a objetos, Construcción, Pruebas y Despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso para JAVA, genera código desde diagramas y genera documentación. Además, se integra con los entornos de desarrollo Eclipse, Borland, JBuilder, NetBeans entre otros.

Visual Paradigm ofrece:

- Diseño centrado en casos de uso y enfocado al negocio lo cual contribuye a que genere un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidad de ingeniería directa (en su 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 a los principales IDE.
- Disponibilidad en múltiples Plataformas.

### 2.2.3 Características Principales de Java.

Java es un lenguaje de programación orientada a objetos desarrollado por Sun Microsystems [8] a principios de la década de 1990. Entre las características de Java se tiene que una misma aplicación puede funcionar en diversos tipos de computadoras y sistemas operativos: Windows, Linux, Solaris, MacOS-X así como en otros dispositivos inteligentes. Además, los programas en Java pueden ser aplicaciones independientes (que corren en una ventana propia) o en javascript que se encuentran incrustados en una página Web y pueden funcionar con cualquier tipo de navegador: Internet Explorer, Netscape, Opera, etc. Java trabaja con sus datos como objetos y con interfaces a esos objetos, y soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo [2].

### 2.2.4 Entorno de Desarrollo Eclipse

La plataforma Eclipse, cuando se combina con el JDT<sup>1</sup>, ofrece muchas de las características que cabría esperar de un IDE<sup>2</sup> de calidad comercial: editor con sintaxis resaltada, compilación incremental, un depurador que tiene en cuenta los hilos a nivel fuente, un navegador de clases, un controlador de ficheros/proyectos, e interfaces para el control estándar de código fuente, como CVS<sup>3</sup> y ClearCase. También incluye varias características únicas como la refactorización de código, la actualización/instalación automática de código (mediante Update Manager). Eclipse es una herramienta de código abierto y es neutral y adaptable a cualquier tipo de lenguaje de programación como C/C++, Cobol, C#, XML entre otros, además la depuración e implementación de aplicaciones resultan sencillas. Es bueno destacar que corre en una gran cantidad de sistemas operativos incluyendo Windows y Linux y le provee a los desarrolladores herramientas que facilitan la creación de plugins<sup>4</sup>.

---

1 JDT es el IDE de JAVA

2 IDE es el Entorno de Desarrollo Integrado

3 Los CVS son los sistemas de control de versiones.

4 Un plugin o componente enchufable es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad muy específica.



Se decidió que Eclipse fuera el IDE seleccionado para desarrollar el sistema por sus grandes potencialidades, además de que corra en diferentes sistemas operativos.

### **2.3 Plataformas Distribuidas**

#### **2.3.1 Plataforma JDCS**

Es una plataforma distribuida como alternativa para realizar cálculos intensos que demandan determinados proyectos relacionados con la bioinformática, aprovechando los diversos recursos computacionales que se encuentran disponibles en la institución, a pesar de su diversidad y heterogeneidad.

En la plataforma hay que programar 2 clases:

- **DataManager:** clase que contiene las funciones necesarias para especificar como distribuir los datos para el cómputo, además de procesar los resultados de las distintas ejecuciones.
- **Algorithm:** clase donde se especifica el algoritmo a ejecutarse en una máquina cliente.

#### **2.3.2 BOINC (Infraestructura Abierta de Berkeley para la Computación en Red).**

BOINC es una infraestructura para la computación distribuida, actualmente se utiliza para diversos campos como física, medicina nuclear, climatología. La intención de este proyecto es obtener una capacidad de computación enorme utilizando computadoras personales. Los proyectos en los que trabaja este software tienen un denominador común, y es que requieren una gran capacidad de cálculo. El BOINC toma la información y la divide en fragmentos y la envía a los usuarios, que buscan una relación matemática en los datos recibidos, cosa que requiere cálculos muy complejos que son resueltos por PCs domésticos que tengan este software instalado. Esta plataforma de software está desarrollada bajo la filosofía de ser código abierto <sup>5</sup> y disponible bajo la licencia GNU LGPL <sup>6</sup>. Está disponible en varias plataformas incluyendo Windows Vista y diversos tipos de Unix como Linux, FreeBSD o Mac OS X.

### **2.4 Java RMI (Remote Method Invocación) Invocación a Métodos Remotos.**

RMI es el mecanismo ofrecido por Java para invocar o ejecutar procedimientos remotos en computadoras o servidores distribuidos, es el mecanismo ofrecido en Java. Permite a un procedimiento (método, clase, aplicación) poder ser invocado remotamente. Una de las ventajas al diseñar un

---

<sup>5</sup> Código Abierto (en Ingles open source) es el término con el que se conoce al software distribuido y desarrollado libremente.

<sup>6</sup> Las siglas LGPL significan Licencia Pública General Limitada de GNU.

procedimiento con RMI es la interoperabilidad, ya que RMI forma parte de todo JDK (Java Development Kit), por ende, cualquier plataforma que tenga acceso a un JDK también tendrá acceso a estos procedimientos (método, clase, aplicación). Al ser RMI parte estándar del entorno de ejecución Java, usarlo provee un mecanismo simple en una aplicación distribuida que solamente necesita comunicar servidores codificados para Java.

Por medio de RMI, un programa Java puede exportar un objeto. A partir de esa operación este objeto está disponible en la red esperando conexiones en un puerto TCP. Un cliente puede entonces conectarse e invocar métodos. La invocación consiste en el "marshaling"<sup>7</sup> de los parámetros utilizando la funcionalidad de "serialización" que provee Java, luego se sigue con la invocación del método que se realiza en el servidor.

## 2.5 Modelos Distribuidos

### 2.5.1 Modelo Cliente-Servidor

El modelo Cliente-Servidor es un modelo para construir sistemas de información, que se sustenta en la idea de repartir el tratamiento de la información y los datos por todo el sistema informático, permitiendo mejorar el rendimiento del sistema global de información. En esta arquitectura la capacidad de proceso está repartida entre los clientes y los servidores, aunque son más importantes las ventajas de tipo organizativo debidas a la centralización de la gestión de la información y la separación de responsabilidades, lo que facilita y clarifica el diseño del sistema.

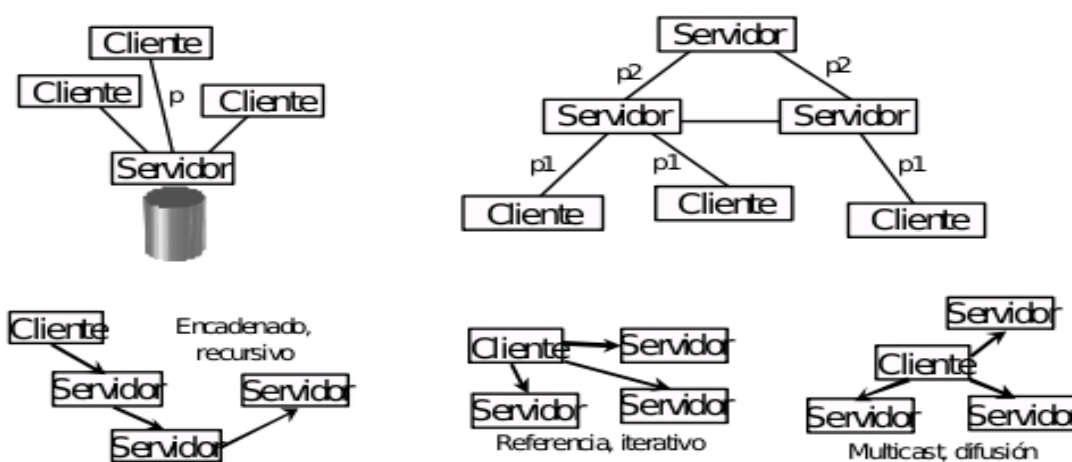


Figura 9: Arquitectura Cliente/Servidor.

<sup>7</sup> Marshaling: puesta en orden, ordenación.

¿Que es un Cliente?

Es el que inicia un requerimiento de servicio. El requerimiento inicial puede convertirse en múltiples requerimientos de trabajo a través de redes LAN o WAN. La ubicación de los datos o de las aplicaciones es totalmente transparente para el cliente.

¿Que es un Servidor?

Es cualquier recurso de cómputo dedicado a responder a los requerimientos del cliente y pueden estar conectados a los clientes a través de redes LANs o WANs, para proveer de múltiples servicios a los clientes y ciudadanos tales como impresión, acceso a bases de datos, fax, procesamiento de imágenes, etc.

El Cliente y el Servidor pueden actuar como una sola entidad y también pueden actuar como entidades separadas, realizando actividades o tareas independientes. Las funciones de Cliente y Servidor pueden estar en plataformas separadas, o en la misma plataforma.

Un servidor da servicio a múltiples clientes en forma concurrente. Cada plataforma puede ser escalable independientemente. Los cambios realizados en las plataformas de los Clientes o de los Servidores, que sean por actualización o por reemplazo tecnológico, se realizan de una manera transparente para el usuario final.

La separación entre cliente y servidor es una separación de tipo lógico, donde el servidor no se ejecuta necesariamente sobre una sola máquina ni es necesariamente un sólo programa.

## **2.6 Conclusiones**

Para la realización del sistema se va a utilizar como lenguaje de modelado UML y como metodología se va a utilizar OpenUP porque se va a trabajar en un proyecto pequeño que no requiere de una ingeniería de software profunda porque lo que se va a hacer es distribuir la aplicación Vega (que es un software que ya tiene su ingeniería hecha) en la plataforma JDCS (que también tiene hecha su ingeniería), y lo que nos quedaría es realizar el diseño de cómo se va a realizar la distribución. La herramienta de modelado gráfico que vamos a emplear es el Visual Paradigm porque está disponible en múltiples plataformas y puede integrarse a los principales IDE, incluyendo Eclipse que es el entorno donde se va a realizar nuestro sistema porque es neutral y adaptable al lenguaje de programación, que en nuestro caso es Java y además puede correr en una gran cantidad de sistemas operativos incluyendo Linux y Windows. El modelo distribuido con que se va a desarrollar nuestro sistema es el modelo cliente-servidor porque el mecanismo del entorno de ejecución que vamos a utilizar es Java RMI. La plataforma donde se va a distribuir el sistema es la JDCS.

## **Capítulo 3: Desarrollo y Resultados**

En este capítulo se especifican los detalles de la construcción del sistema, brindando como parte de este los requerimientos funcionales y no funcionales. Se realizan descripciones a los casos de uso del sistema. Además muestra el modelado de su funcionamiento, transformando los requerimientos funcionales a diseños del sistema, también se adapta el diseño para hacerlo corresponder con el ambiente de implementación para lograr los resultados satisfactorios esperados.

### **3.1 Ingeniería del Software**

#### **3.1.1 Requerimientos**

Un requerimiento es una condición o capacidad que tiene que ser alcanzada o poseída por un sistema o componente de un sistema para satisfacer un contrato estándar u otro documento impuesto formalmente [5].

##### **Requerimientos Funcionales**

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir [5].

- **R1 Gestionar Cálculo**
  1. **Realizar cálculo químico teórico.**
  2. **Buscar Problema en Ejecución.**
    - 2.1 Interrumpir la Ejecución del Problema.
    - 2.2 Mostrar Estado.
  3. **Buscar Resultado Obtenido.**
    - 3.1 Eliminar el Fichero Resultado.
    - 3.2 Descargar el fichero para un lugar seleccionado.

##### **Requerimientos no Funcionales**

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Representan las características del producto [5].

- **Usabilidad:** El sistema puede ser usado por cualquier persona que posea conocimientos básicos en el manejo del software, se necesita contar con conocimientos especializados en química para entender los resultados dados por la aplicación.
- **Rendimiento:** Al estar concebida para un ambiente cliente-servidor, se trata de garantizar la rapidez de respuesta del sistema ante las solicitudes de los usuarios, al igual que la velocidad de procesamiento de la información.

- **Soporte:** El sistema debe propiciar su mejoramiento y la anexión de otras opciones que se le incorporen en un futuro.
- **Portabilidad:** El sistema puede ser ejecutado sobre los sistemas operativos Linux y Windows.
- **Seguridad:** El especialista tendrá que autenticarse para tener acceso a la aplicación, una vez accedida a esta tendrá control total de la aplicación, sin restricción alguna.
- **Ayuda:** El sistema cuenta con una ayuda donde se exponen las opciones de cálculo que posee la aplicación.
- **Software:** Se debe disponer del sistema operativo Linux para la ejecución de la aplicación Vega. Debe tenerse instalado el Java Runtime Environment (JRE) versión 1.5 o superior.
- **Hardware:** Para el desarrollo y puesta en práctica del sistema se requieren  
 Procesador Pentium 3 o superior  
 256 o mas MB de RAM  
 50 MB de capacidad del disco duro

### 3.1.2 Actores del Sistema

Los actores del sistema pueden representar el rol que juega una o varias personas, un equipo o un sistema automatizado, son terceros fuera del sistema que interactúan con el [5].

Actores del Sistema	Justificación
Especialista	Es el usuario que va a hacer uso del sistema, es el que se encarga de seleccionar los ficheros de entrada, seleccionar el comando y mandar a ejecutar la aplicación, también se encarga de descargar o borrar algún resultado y de interrumpir algún problema en ejecución.
Vega	Es el software encargado de realizar los cálculos químico-teóricos.

**Tabla 1:** Actores del Sistema

### 3.1.3 Casos de Uso del Sistema

Los casos de uso son artefactos narrativos que describen, bajo la forma de acciones y reacciones, el comportamiento del sistema desde el punto de vista del usuario [5].

Nombre del Caso de Uso	Justificación
Gestionar Cálculo	Objetivo principal del sistema que abarca todo el proceso de cálculos químico-teóricos deseados.

Tabla 2: Caso de Uso del Sistema

### 3.1.4 Diagrama de Casos de Uso del Sistema

Un diagrama de caso de uso del sistema representa gráficamente a los procesos y su interacción con los actores [5].



Figura 10: Diagrama de Caso de Uso.

### 3.1.5 Descripción del Caso de Uso del Sistema

Caso de Uso:	Gestionar Cálculo
<b>Actores:</b>	Especialista[inicia], Vega
<b>Propósito:</b>	El Especialista tiene la opción de mandar a realizar los cálculos químicos, de buscar resultados de cálculos anteriores o buscar el estado de un problema que se está ejecutando.
<b>Resumen:</b>	El Especialista es el que inicia el Caso de Uso cuando tiene que autenticarse para poder realizar las operaciones que desea.
<b>Referencia:</b>	R1
<b>Precondiciones:</b>	Que el usuario esté autenticado en el sistema.
<b>Poscondiciones:</b>	
<b>Flujo Normal de Eventos</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
1. El Especialista se autentica.	2. Muestra una interfaz para que escoja la opción de: <ul style="list-style-type: none"> <li>▪ Realizar cálculos químicos teóricos.</li> <li>▪ Buscar Resultados Obtenidos.</li> <li>▪ Buscar Problemas en Ejecución.</li> </ul>

<p>3. Si escoge la opción:</p> <ul style="list-style-type: none"> <li>▪ Realizar cálculos químicos teóricos ir a la sección <b>“Realizar cálculos químicos teóricos”</b>.</li> <li>▪ Buscar Resultados Obtenidos ir a la sección <b>“Buscar Resultados Obtenidos”</b>.</li> <li>▪ Buscar Problemas en Ejecución ir a la sección <b>“Buscar Problemas en Ejecución”</b>.</li> </ul>	
<b>Flujos Alternos</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
	2.1 No permite acceder al sistema y muestra un mensaje de error.
<b>Prototipo de Interfaz</b>	
Ver (Figura 29 y Figura 30 del Anexo 2)	
<b>Sección “Realizar cálculos químicos teóricos”.</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
1. El Especialista la opción <b>“Ficheros”</b> .	2. Sistema abrirá un explorador donde el Especialista podrá localizar sin dificultad el directorio donde se encuentran sus ficheros.
3. El Especialista selecciona el directorio donde se encuentran sus ficheros.	4. El sistema guarda en memoria la dirección de los ficheros seleccionados.
5. El usuario selecciona los comandos y selecciona la opción <b>“Crear comando”</b> .	6. El sistema muestra un mensaje verificando la correcta entrada de los datos.
7. El Especialista selecciona la opción <b>“Ejecutar”</b> .	8. El Sistema sube los ficheros seleccionados para la plataforma y manda a ejecutar la aplicación.

9. El software químico procede a realizar los cálculos químicos indicados.	10. El sistema muestra el resultado de los cálculos.
<b>Flujos alternos Sección “Realizar cálculos químicos teóricos.”</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
<b>Sección “Buscar Resultados Obtenidos”.</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
	1. El sistema muestra una interfaz con una lista de resultados de los cálculos realizados.
2. Selecciona el fichero.	
3. Si desea eliminar el fichero selecciona la opción “Eliminar”.	4. El sistema elimina el fichero y actualiza el listado de ficheros.
5. Si desea descargar el fichero selecciona la opción de “Descargar”	6. El sistema abre un explorador para que el usuario seleccione la dirección donde desee descargar el fichero.
7. El usuario selecciona la dirección y selecciona la opción “Aceptar”.	8. El sistema descarga el fichero.
<b>Flujos alternos Sección ” Buscar Resultados Obtenidos”</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
<b>Prototipo de Interfaz</b>	
Ver(Figura 32 del Anexo 2)	
<b>Sección “Buscar Problemas en Ejecución”.</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
	1. El sistema abre una interfaz con una lista con todos los problemas que se están ejecutando.
2. El usuario selecciona el problema.	
3. Si desea interrumpir la ejecución del problema selecciona la opción “Interrumpir”.	4. El sistema termina la ejecución y actualiza el listado de ejecuciones.
5. Si el usuario desea mostrar el estado de la ejecución selecciona la	6. El sistema muestra una ventana con el estado de la ejecución seleccionada.



opción "Estado".	
<b>Flujos alternos Sección "Buscar Problemas en Ejecución".</b>	
<b>Acción del Actor</b>	<b>Respuesta del Sistema</b>
<b>Prototipo de Interfaz</b>	
Ver (Figura 31 y Figura 33 del Anexo 2)	
<b>Prioridad:</b>	Crítico

**Tabla 3:** Descripción del CU Gestionar Cálculo

### 3.1.6 Patrón de Casos de Uso

**Patrón CRUD (Creating, Reading, Updating and Deleting):** El patrón CRUP propone identificar un CU, llamado "Gestionar Cálculo", que modela las operaciones que se pueden realizar sobre una parte de información de cierto tipo (o sea en una misma entidad), como es el caso de crearla, leerla, actualizarla y eliminarla.

### 3.1.7. Patrón de Diseño

Los Patrones de Diseño son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Para el Diseño de nuestro sistema se decidió seguir los basamentos e ideas del patrón Máster–Slave, donde un componente maestro (master) distribuye el trabajo a los componentes esclavos (slaves). El componente maestro calcula el resultado final a partir de los resultados arrojados por los componentes esclavos.

También se aplicaron los patrones GRASP. Estos patrones describen los principios fundamentales de asignación de responsabilidades a los objetos, expresados en forma de patrones.

El patrón **EXPERTO** plantea que la responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Un ejemplo de este patrón se puede observar en la Figura 2.

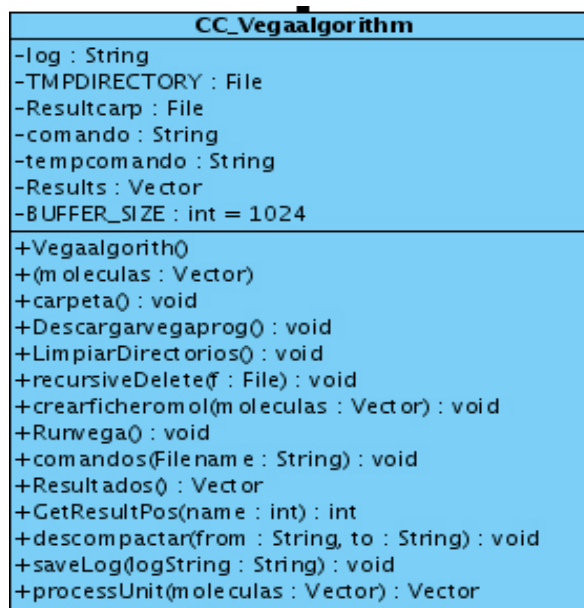


Figura 11: Ejemplo del patrón Experto.

El patrón **CREADOR**: Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. Los que define este patrón es que una instancia de un objeto la tiene que crear el objeto que tiene la información para ello. En la Figura 3 se puede evidenciar la presencia de este patrón.

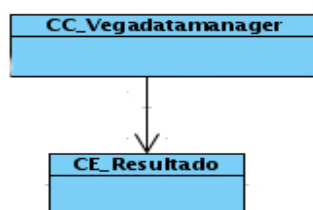


Figura 12: Ejemplo del patrón Creador

El patrón **ALTA COHESIÓN**: Asigna una responsabilidad de modo que la cohesión sea alta. Los elementos de un componente (clase) colaboran para producir algún comportamiento bien definido. Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. La responsabilidad de este patrón es evitar asignar demasiadas responsabilidades a las clases. En la figura 4 se puede evidenciar la presencia de este patrón.

```

CC_Vegadatamanager
-Buffer : Vector
-moleculasInProcess : int
-Granularity : int
-totalmolecules : int
-TMPDIRECTORY : File
-PROBLEMLOG : File
-StartTime : long
-EndTime : long
-TOTAL_WALL_CLOCK_TIME : long
-storageResults : int
-collectedResults : int
-clientlog : String
+Vegadatamanager()
+fillBuffer() : void
+(arg0 : ClientInfo)
+processResults(arg0 : Long, arg1 : Vector) : boolean
+(percent : double)
+closeResources() : void
+endCalculation() : void
+limpiarTMPDIRECTORY() : void
+recursiveDelete(f : File) : void
+saveLog(logString : String) : void
+comand() : Molecule
+generateWorkUnit(arg0)
+getStatus() : String
+adjustGranularity(percent : double) : void
    
```

Figura 13: Ejemplo del patrón Alta Cohesión

El patrón **CONTROLADOR**: Asigna la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad, etc.). El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de alta cohesión. En la siguiente figura se muestra un ejemplo de este patrón.

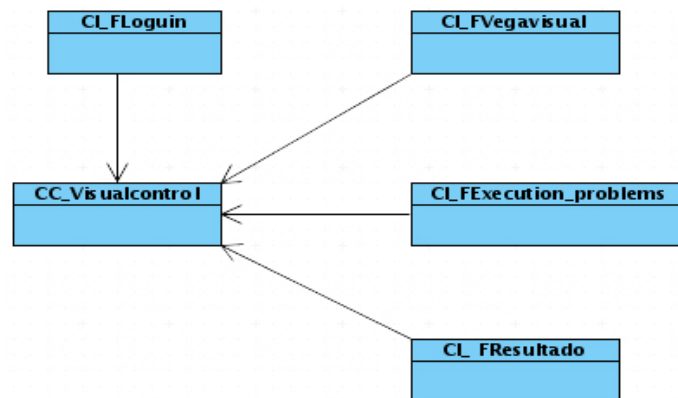


Figura 14: Ejemplo del patrón Controlador

El patrón de **BAJO ACOPLAMIENTO**: Es un patrón evaluativo que el diseñador aplica al jugar sus decisiones de diseño. Plantea que debe haber pocas dependencias entre las clases. La presencia de este patrón se puede evidenciar en la Figura 6.

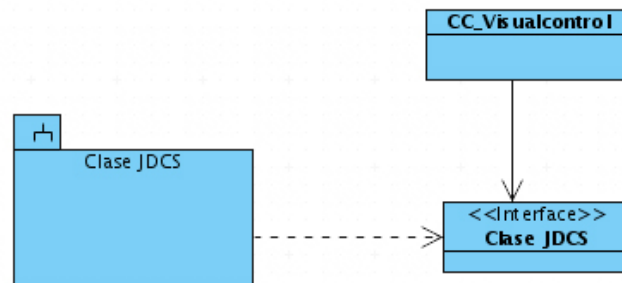


Figura 15: Ejemplo del patrón Bajo Acoplamiento.

### 3.1.8 Diagrama de Clases del Diseño

Los Diagramas de Clases del Diseño describen gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación.

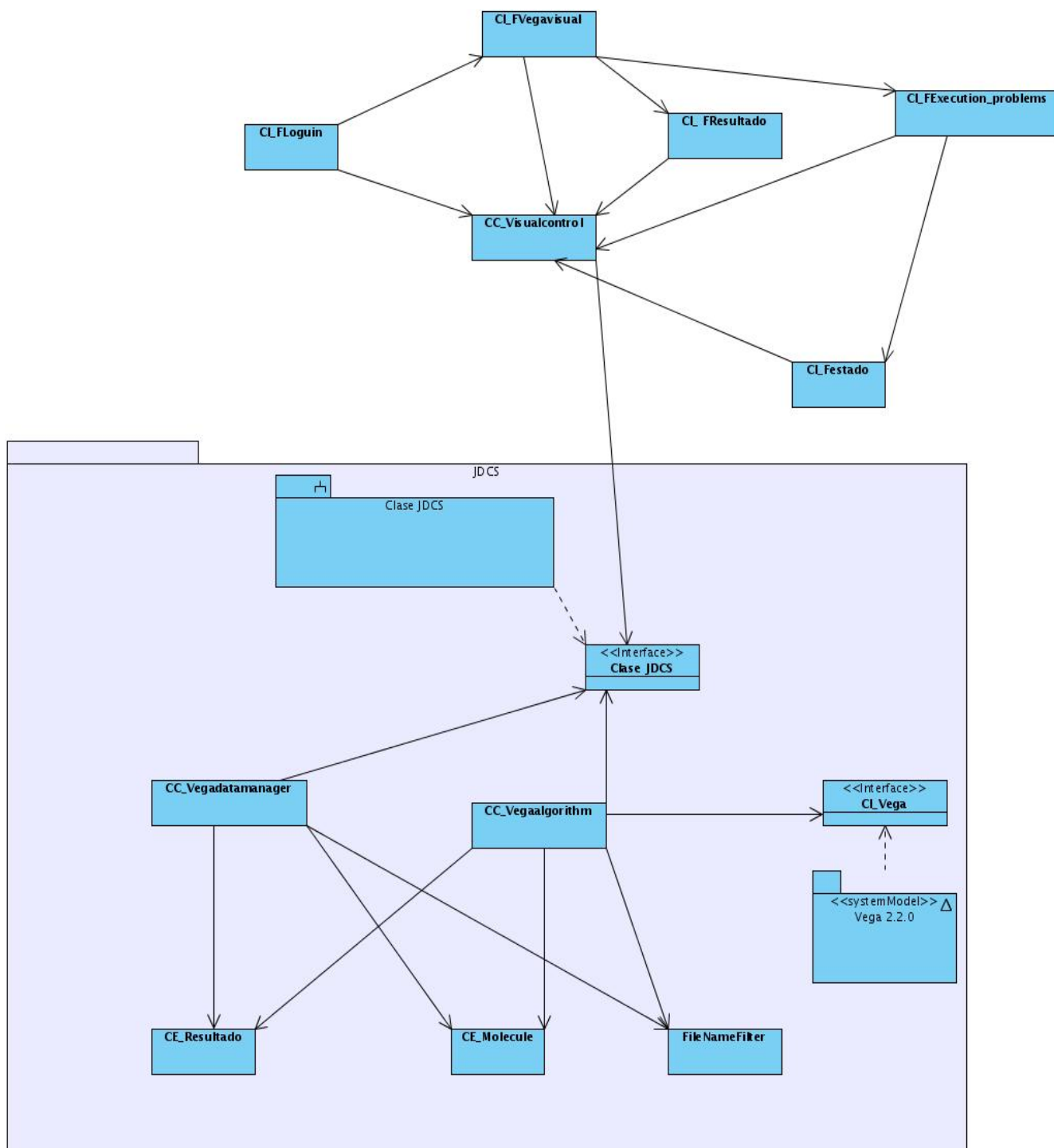


Figura 16: Diagrama de clases del diseño

### 3.1.9 Diagrama de Interacción

Los Diagramas de Interacción modelan el aspecto dinámico del sistema.

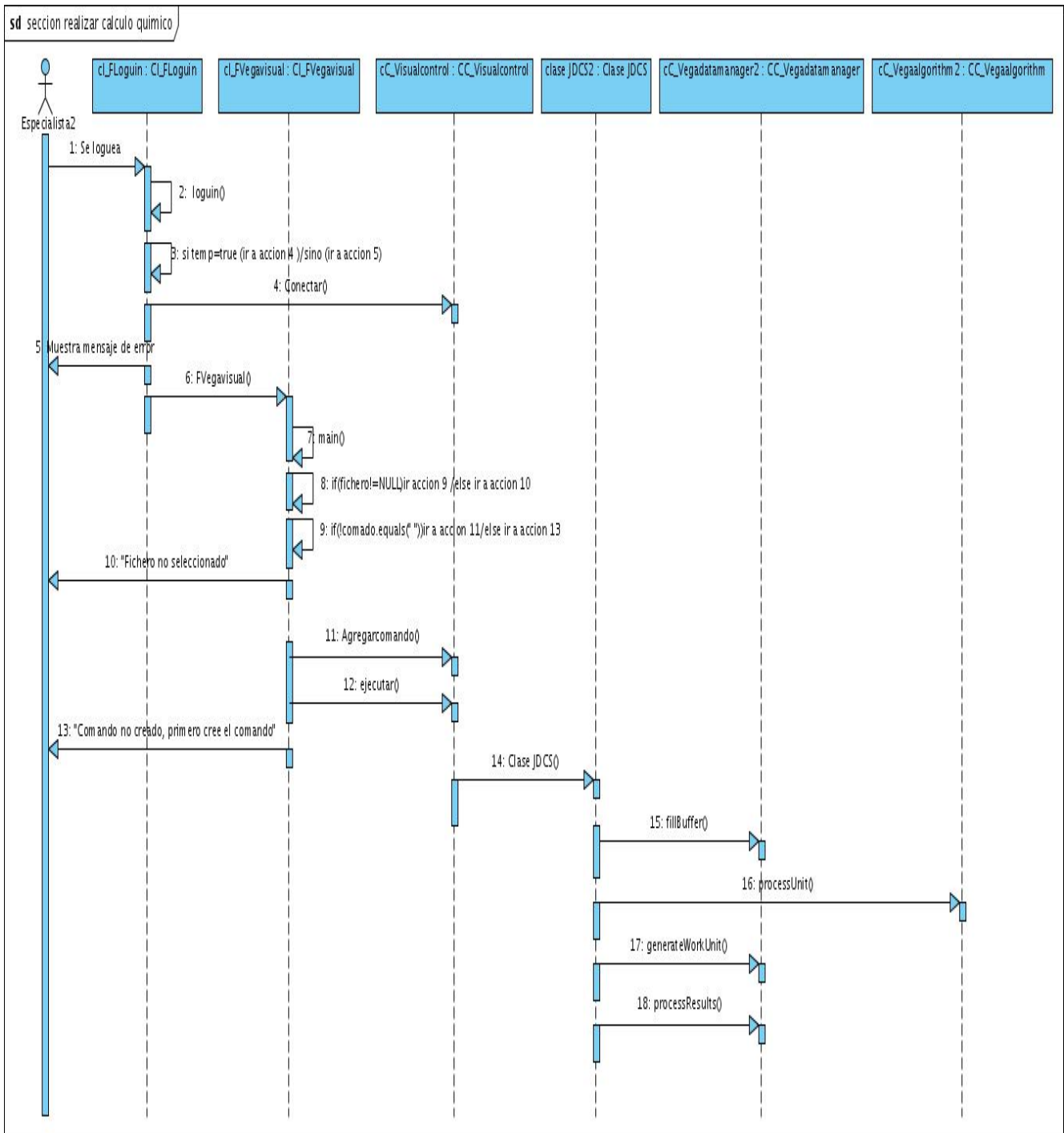


Figura 17: Diagrama de secuencia. Sección Realizar Cálculo químico teórico.

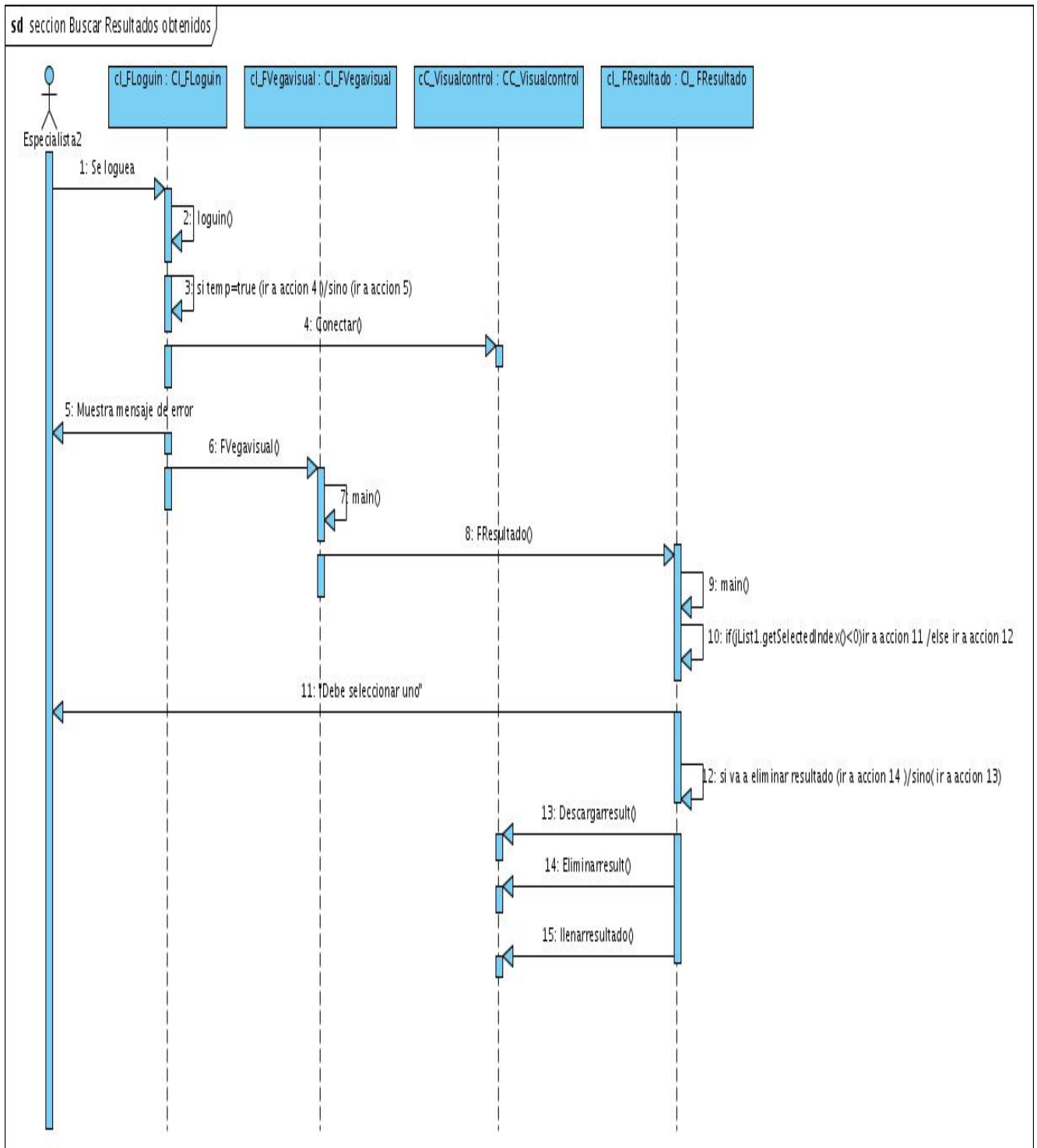


Figura 18: Diagrama de secuencia. Buscar Resultados Obtenidos.

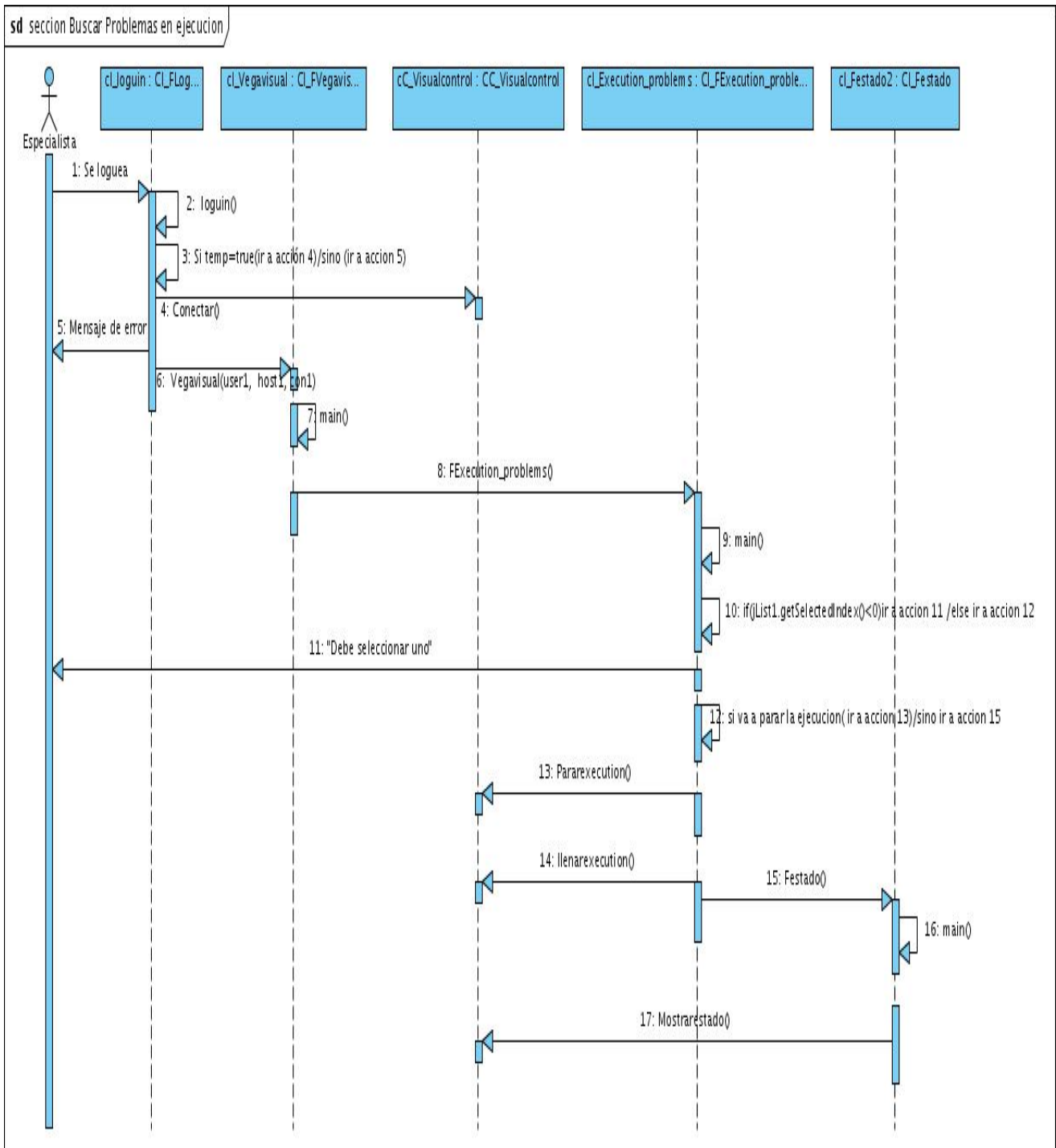


Figura 19: Diagrama de secuencia. Buscar Problemas en Ejecución.

### 3.1.10 Descripción de las clases.

(Ver anexo #1.)



### 3.1.11 Diagrama de Despliegue



Figura 20: Diagrama de Despliegue

### 3.1.12 Diagramas de Componentes.

Los diagramas de componentes son usados para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre los elementos de implementación.

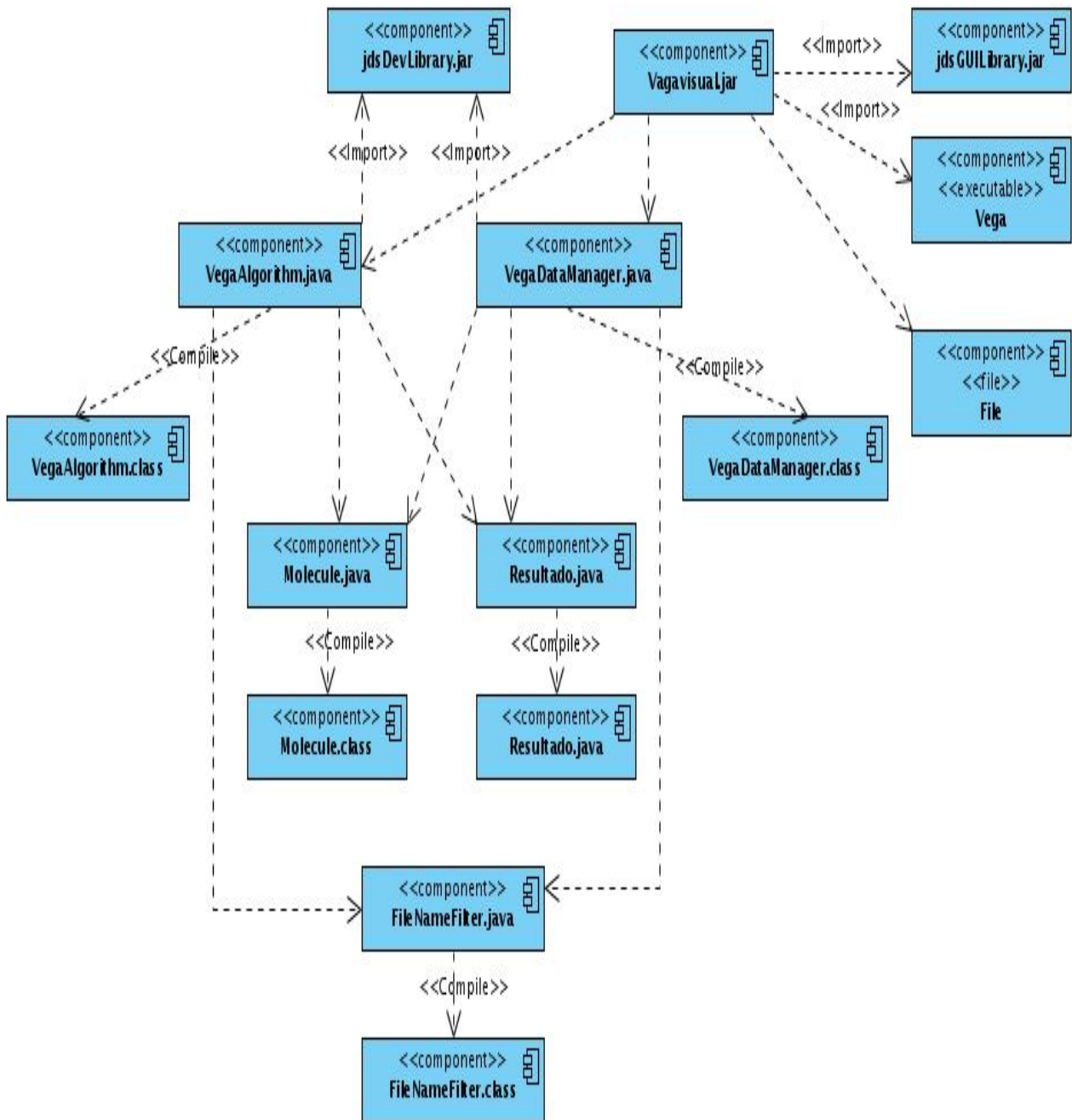


Figura 21: Diagrama de Componentes

### 3.1.13 Algoritmos.

```

public class VegaAlgorithm extends Algorithm
{
    ...
    File TMPDIRECTORY;
    String tempcomando;
    ...
    |
    public VegaAlgorithm()
    {
        Results=new Vector<Resultado>();
    }
    public Vector processUnit( Vector molculas ) throws Throwable
    {
        ...
        // recoge el comando con que se va a ejecutar el Vega.
        Molecule com=(Molecule) molculas.elementAt(0);
        tempcomando = com.getData();
        molculas.remove(0);
        ...
        // manda a relizar todas las operaciones.
        if(molculas!=null)
        {
            carpeta();
            crearficheromol(molculas);
            Descargavegaprog();
            Runvega();
            resultado=Resultados();
            Resultado result = new Resultado(0,log,".mol",100);
            Results.add(result);
        }
        return resultado;
    }...
}

```

**Figura 22:** Implementación del Método processUnit de la clase VegaAlgorithm.

```
public class VegaDataManager extends DataManager
{
...
    private int moleculesInProgress;
    private int totalmolecules;
        int storageResults;
        int collectedResults;
...
public VegaDataManager() throws Throwable
{
    //se inicializan las variables.
    ...
        totalmolecules=0;
        moleculesInProgress = 0;
        collectedResults = 0;
        storageResults = 0;
    ...

public String getStatus() throws Throwable
{
    String aux;
    aux="Total de moléculas: "+totalmolecules+"\r\n";
    aux+="Moléculas procesadas: "+collectedResults+"\r\n";
    aux+="Moléculas falladas: "+(collectedResults-storageResults)+"\r\n";
    aux+="Moléculas en proceso: "+moleculesInProgress+"\r\n";
    return aux;
}
}
```

Figura 23: Implementación del Método getStatus de la clase VegaDataManager.

```

public Vector generateWorkUnit( ClientInfo arg0 ) throws Throwable
{
    // verifica si hay mas moleculas para precesar y si el sistema
    // operativo es linux para mandarle un Vector <Molecule> y manda
    // a ejecutar todas las operaciones.
    boolean acepto = true;
    if(Buffer.isEmpty())
        acepto=false;
    if(acepto && arg0.getOS().contains("Linux"))
    {
        Vector<Molecule> workUnit = new Vector<Molecule>();
        //comando a ejecutar del vega
        Molecule aux=comand();
        workUnit.addElement(aux);
        int workSize = Granularity;
        if(Buffer.size() < workSize )
            workSize=Buffer.size();
        // se hace un log para saber que maquinas estan solicitando trabajo, su arquitectura
        // y las operaciones que se han ido realizando.
        // adiciona una unidad de trabajo
        for (int i = 0; i < workSize; i++)
            {
                workUnit.addElement( Buffer.get(0) );
                ...
                Buffer.removeElementAt(0);
                moleculesInProcess++;|
            }
        //verifica la cantidad de moleculas que envio para cada maquina.
        ...
        saveLog(log);
        return workUnit;
    }
    return null;
}

```

**Figura 24:** Implementación del Método generateWorkUnit de la clase VegaDataManager.

```

public boolean processResults( Long uID, Vector arg1 ) throws Throwable
{
    try
    {
        if(Results != null)
        {
            for (Object object : Results)
            {
                Resultado result = (Resultado)object;
                if(result.getName()!=0)
                {
                    if(result.getData()!=null)
                    {
                        PARTIAL_TIME += result.getProcessTime(); ...
                    }
                    else
                    {
                        NotProcessed++;
                        PARTIAL_TIME += result.getProcessTime();
                        if(firstnot)
                            notprocessed += String.valueOf(result.getName());
                    }
                    else
                    {
                        notprocessed += ", " + String.valueOf(result.getName());
                        firstnot = false;
                    }
                }
            }
            storageResults += Processed;
            collectedResults += Processed + NotProcessed;}

        // se lanza una excepcion de error ...
        TOTAL_WALL_CLOCK_TIME += PARTIAL_TIME;
        processlog += collectedResults;
        workUnitTime += PARTIAL_TIME + " seconds";
        clientstime += TOTAL_WALL_CLOCK_TIME + " seconds";
        long partial = System.currentTimeMillis();
        long divider = 1000;
        long currenttime = (partial - StartTime) / divider;
        jdsTime += currenttime + " seconds";
        if(collectedResults >= totalmolecules)
        {
            endCalculation();
            return true;}
        return false; }
    }
}

```

Figura 25: Implementación del Método processresult de la clase VegaDataManager.

## 3.2 Resultados Experimentales.

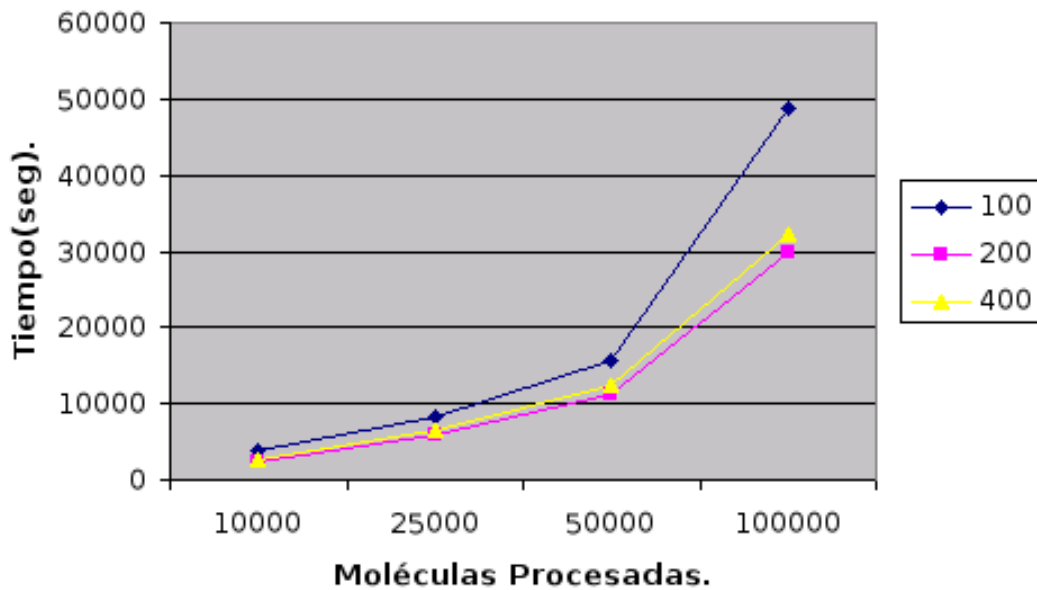
### 3.2.1 Prueba para demostrar la eficiencia de los cálculos de forma distribuida.

Para demostrar la eficiencia del sistema desarrollado se realizaron pruebas para comparar el tiempo que demora el proceso de realización de los cálculos en una computadora personal como se realiza actualmente, y el tiempo que demoran estos mismos cálculos al distribuir los ficheros moleculares en la plataforma.

Se tomaron muestras de 100000, 50000 y 25000 y 10000 moléculas con extensiones **.mol** que tenían entre 2 y 235 átomos cada una, el tiempo de procesamiento de estos ficheros en la aplicación Vega fue de 10h 35min, 3h 55min, 2h 12min y 0h 7min respectivamente en una máquina personal. Para realizar

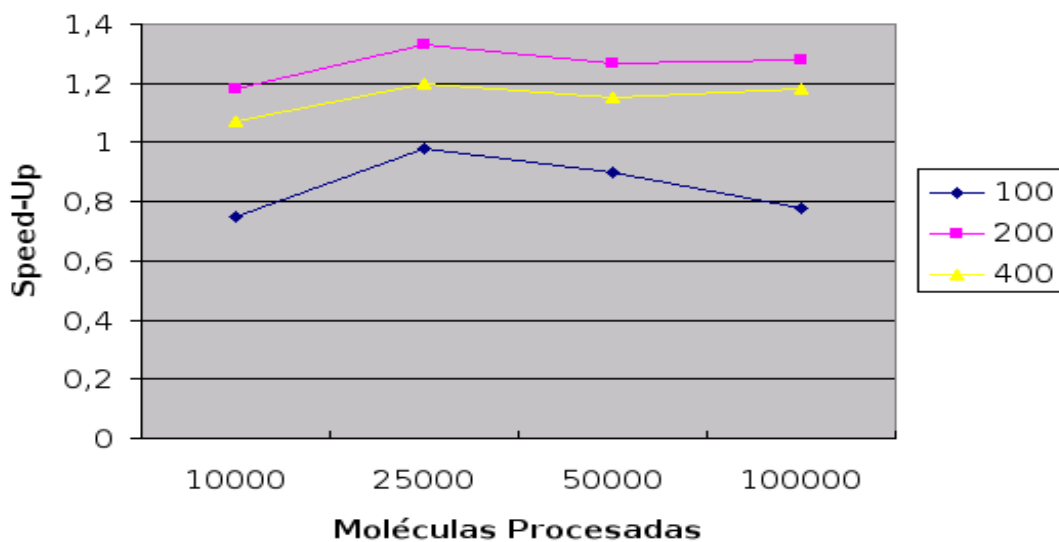
éstos cálculos de manera distribuidas se disponían de 22 a 30 clientes Pentium IV con el sistema operativo Linux y 512Mb y 712Mb de memoria RAM, conectados a una red con una velocidad de 100 Mbps. En el momento en que se realizaron los cálculos se estaban ejecutando otros procesos en la plataforma por lo que el proceso en las pruebas que se hicieron utilizó de 4 a 6 máquinas.

Se realizaron pruebas para verificar la granularidad de mejor rendimiento y se llegó a la conclusión de que la más favorable era 200 moléculas por máquina, esto se puede evidenciar en la siguiente figura.



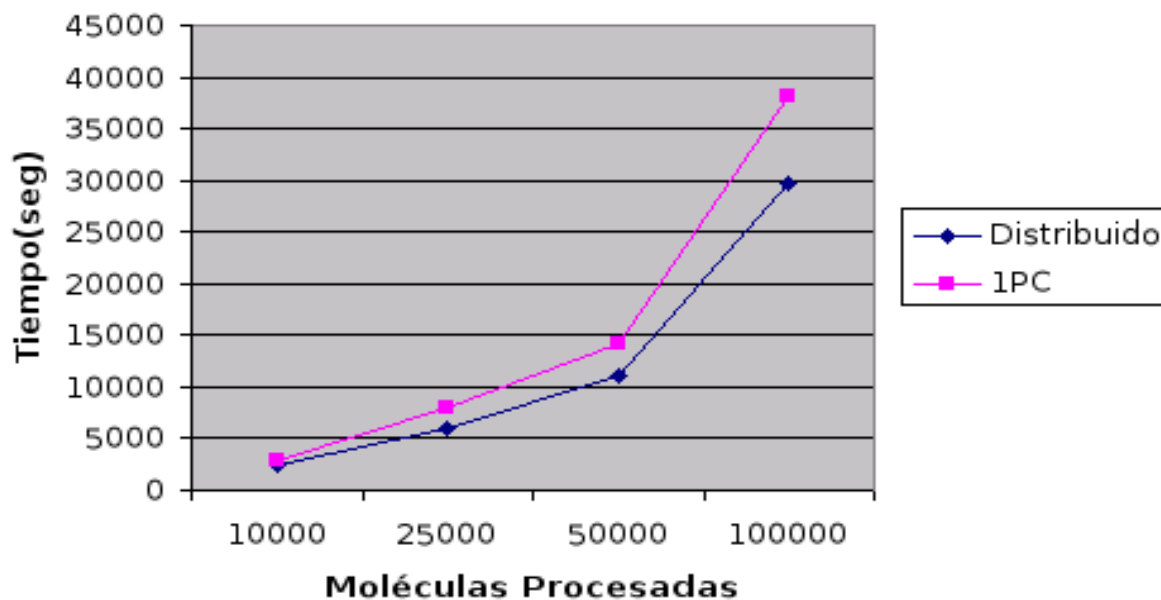
**Figura 26:** Tiempo de Procesamiento por Granularidad.

La siguiente gráfica muestra la ganancia de velocidad (*Speed-Up*) obtenida en las distintas pruebas realizadas:



**Figura 27:** Speed-Up obtenido en las pruebas realizadas.

Como se puede apreciar, el *Speed-Up* para una granularidad de 200 siempre se comportó por encima de 1.2, es decir, el sistema distribuido obtiene una ganancia de velocidad al menos 1.2 veces mayor, esto se puede apreciar en la siguiente figura.



**Figura 28:** Procesamiento en 1 PC y Distribuido con granularidad de 200.

El tiempo de procesamiento distribuido hubiese sido mejor si se hubiesen realizado las pruebas con todas las máquinas con que contaba la plataforma, y no con menos de 6 clientes como fue el caso.

Las causas de la demora en el procesamiento de algunas unidades de trabajo, puede estar dada por varias razones, que en el tiempo de inicio de procesamiento solo hicieron peticiones de trabajo algunos clientes y el resto lo hicieron mucho tiempo después o puede ocurrir que en un primer momento estas unidades de trabajo se enviaron a los clientes y luego de un tiempo de no reportarse al servidor se consideraron expiradas y fueron enviadas nuevamente a otros clientes. Otra posible razón es que en algunas estaciones de trabajo la prioridad de ejecutar este proceso sea muy baja, o que tenga muchos procesos corriendo y como consecuencia sea más lenta la ejecución. En las pruebas realizadas algunos clientes se reportaron al servidor y no empezaron a procesar las unidades de trabajo hasta después de un tiempo prolongado luego de haber realizado la petición, incluso más de 15 minutos.



### **3.3 Conclusiones**

En este capítulo se definieron los requisitos que debe cumplir el sistema para su correcto funcionamiento, determinado así las diferentes funcionalidades del mismo resumidas como casos de uso, de los que se realizó una descripción detallada para su mejor comprensión. Se describieron las diferentes clases que forman parte de los diagramas de clases de diseño. De la misma manera los diferentes diagramas de componentes modelan la forma en que fue implementado el sistema para su correcto funcionamiento. También se trataron los principios de diseño que se siguieron para el desarrollo del sistema. El diagrama de despliegue describió la organización entre nodos físicos en los que funciona el sistema. Se realizaron pruebas de tiempo de ejecución al sistema para mostrar y comprobar su correcto funcionamiento de forma eficiente.

### **Conclusiones Generales**

- Se diseñó e implementó un sistema capaz de gestionar cálculos químico a grandes bancos de moléculas a través de una red para el sistema operativo Linux y Windows.
- Se demostró la eficiencia del sistema implementado ya que se redujo el tiempo procesamiento en pruebas realizadas a 100000, 50000, 25000 y 10000 ficheros moleculares que disminuyeron sus tiempos de cálculo de 10h:35min, 3h:55min, 2h:12min y 47min a 8h:15min, 3h:05min, 1h:39min y 40min respectivamente, lo cual demostró tener una velocidad 1.2 veces mayor que en una computadora personal.

### **Recomendaciones**

- Tener presente que el sistema fue realizado específicamente para que funcionara en la Universidad, que en caso de ponerlo en práctica en cualquier otro lugar, habría que implementar un sistema Grid similar al implementado en el centro.
- Ir mejorando la Interfaz gráfica a medida que vayan saliendo las actualizaciones de la aplicación Vega.

## Bibliografías

1. *Sistemas Distribuidos. Nuevos paradigmas de los Sistemas de Información.* Universidad Carlos III. Madrid : s.n. Doctorado en ingeniería Informática.
2. Tutorial de Java.
3. **Andrés Camejo Isaac, Adnier Turro.** *Módulo para el Cálculo Distribuido de Mecánica Molecular Mecánica Cuántica.* Universidad de las Ciencias Informaticas. 2007. Trabajo de Diploma para optar por el título de ingenieros en las Ciencias Informaticas.
4. **Molpeceres., Alberto.** Proceso de Desarrollo: Rup, XP, FDD.
5. Ayuda Extendida de Rational Rose Interprise Edition. 2003.
6. Vega ZZ. *Vega ZZ.* [Online] <http://www.ddl.unimi.it/vega/manual/pages/index.htm> .
7. Metodología de Booch. . [Online] <http://es.wikipedia.org/wiki/Booch..>
8. SunMicrosystems. [Online] [http://www.sun.com/..](http://www.sun.com/)
9. Quanchem Quantum Chemistry Packages. . [Online] <http://www.chemsoft.ch/qc/qce.htm..>
10. **Fuentes, J. L. Vázquez, E. Huedo, R. S. Montero e I. M. Llorente.** *Beneficios del uso de la tecnología grid computing en bioinformática usando la infraestructura de IRISGrid.*
11. **Education., Ed. Pearson.** “*La programación extrema en la práctica*”. Madrid : s.n., 2002.
12. **Sanchez., María A. Mendoza.** *Metodologías De Desarrollo De Software. Ing. Informático – UNT Microsoft Certified Professional – MCP Analista y Desarrolladora – TeamSoft . . Perú S.A.C. : s.n.*
13. **(RUP)., Rational Unified Process.**
14. **Pekka Abrahamsson, Outi Salo, Jussi Ronkainen & Juihani Warsta.** *Agile Software Development Methods.*
15. Taller de Requerimientos, Análisis y Diseño con UML. . *Guía del alumno.* .
16. **Galves., Ing Jorge.** *Fundamentos de la Metodología RUP.* Universidad Tecnológica de Pereira. : s.n.
17. **Bioinformática., Lic. Logendri Aguilera Mendoza.** *Sistema de Cómputo Distribuido aplicado a la Bioinformática.* Tesis en Opcion al grado de Master en Master en Bioinformática. .
18. *Process, Rational Unified.* Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia. Valencia : s.n., 2005.
19. **Cao., Ing. José Ignacio.** *El Ciclo de Vida Orientado a Objetos según el Método de Grady Booch.*
20. SunMicrosystems. *SunMicrosystems.* [Online] <http://www.sun.com/> .
21. Universidad Politécnica de Catalunya. [En línea] 9 de 2005. [Citado el: 20 de 4 de 2008.] **FUNDAMENTOS DE SISTEMAS DISTRIBUIDOS;** Departamento de Arquitectura de Computadoras.

<http://64.233.169.104/search?q=cache:ec6SSG7XW88J:studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf+%22Conceptos+Generales+de+Sistemas+distribuidos%22%2B&hl=es&ct=clnk&cd=1&gl=cu>.

22. **Rodas, Prof. Guillermo Gonzáles.** *Caracterización de los Sistemas Distribuidos. Capítulo 1.* Universidad Nacional de Asunción. Facultad Politécnica de Ingeniería Informática., Paraguay: s.n.

23. ISP.com. *ISP.com.* [Online] <http://www.isps.com/>.

### Referencias Bibliográficas

1. *Sistemas Distribuidos. Nuevos paradigmas de los Sistemas de Información*. Universidad Carlos III. Madrid : s.n. Doctorado en ingeniería Informática.
2. Tutorial de Java.
3. **Andrés Camejo Isaac, Adnier Turro**. *Módulo para el Cálculo Distribuido de Mecánica Molecular Mecanica Cuántica*. Universidad de las Ciencias Informaticas. 2007. Trabajo de Diploma para optar por el título de ingenieros en las Ciencias Informaticas.
4. **Molpeceres., Alberto**. Proceso de Desarrollo: Rup, XP, FDD.
5. Ayuda Extendida de Rational Rose Interprise Edition. 2003.
6. Vega ZZ. *Vega ZZ*. [Online] <http://www.ddl.unimi.it/vega/manual/pages/index.htm> .
7. Metodología de Booch. . [Online] <http://es.wikipedia.org/wiki/Booch..>
8. Sun Microsystems. [Online] [http://www.sun.com/..](http://www.sun.com/)
9. Quanchem Quantum Chemistry Packages. . [Online] <http://www.chemsoft.ch/qc/qce.htm..>
10. Universidad Politécnica de Catalunya. [En línea] 9 de 2005. [Citado el: 20 de 4 de 2008.] FUNDAMENTOS DE SISTEMAS DISTRIBUIDOS; Departamento de Arquitectura de Computadoras. <http://64.233.169.104/search?q=cache:ec6SSG7XW88J:studies.ac.upc.edu/EPSC/FSD/FSD-ConceptosGenerales.pdf+%22Conceptos+Generales+de+Sistemas+distribuidos%22%2B&hl=es&ct=clnk&cd=1&gl=cu>.
11. **Rodas, Prof. Guillermo Gonzáles**. *Caracterización de los Sistemas Distribuidos. Capítulo 1*. Universidad Nacional de Asunción. Facultad Politécnica de Ingeniería Informática., Paraguay: s.n.
- 12 . ISP.com. *ISP.com*. [Online] <http://www.isps.com/>.

## ANEXOS

## Anexo1: Descripción de las clases

<b>Nombre:</b> Vega	
<b>Tipo de Clase:</b> <<Interface>>.	
<b>Responsabilidades:</b> Interface brindada por la aplicación Vega en su versión 2.2.0 para el sistema operativo Linux.	
1. Nombre	-a [RESNUM]
Descripción:	Esta función renumera todos los residuos a partir de [RES_NUM].
2. Nombre	-b
Descripción:	Con este cambio, se puede desactivar el ahorro de conectividad para los formatos moleculares que pueden almacenar este tipo de información.
3. Nombre	-d [DIELECTRIC]
Descripción:	Función necesaria para calcular la energía de interacción.
4. Nombre	-e [NOMBRE: NUM]
Descripción:	Función necesaria para especificar la cantidad de residuos.
5. Nombre	-f [OUT.PACK]
Descripción:	Con este parámetro, puede crear un archivo de salida en un formato de archivo específico.

6. Nombre	-g [RADIUS]
Descripción:	Función para calcular un mapa de superficie con una sonda de radio diferente a la predeterminada sin cambiar el archivo de preferencias.
7. Nombre	-k [PALABRAS CLAVE]
Descripción:	Esta función es útil para pasar el control de la Información de las palabras clave cuando XML o Mopac es el formato de salida seleccionado.
8. Nombre	-l [MOLTYPE]
Descripción:	Esta función añade el hidrógenos a la molécula cargada / s, saturar todas las valencias de átomo.
9. Nombre	-m [PALABRAS CLAVE]
Descripción:	Esta función es necesaria si se quiere hacer una medida específica en una trayectoria dinámica molecular.
10.Nombre	-n
Descripción:	Esta función permite a la normalización de las coordenadas atómicas.
11.Nombre	-o [salida]
Descripción:	Con esta función se puede especificar el nombre del archivo de salida.
12.Nombre	-q [FORMA]
Descripción:	Con esta función se puede fijar el orden de bonos utilizando el método especificado (ALL, RINGS).



13.Nombre	-r
Descripción:	Esta función elimina todos los átomos de hidrógeno y es muy útil para crear un archivo de AP adecuada para subir a las bases de datos moleculares.
14.Nombre	-s [puntos]
Descripción:	Con esta función se puede cambiar el punto de densidad de superficie de un mapa.
15.Nombre	-w
Descripción:	Esta función elimina todas las moléculas de agua presentes.
16.Nombre	-u
Descripción:	Esta función agrega las cadenas laterales de una proteína.
17.Nombre	-t[SECSTRUCT]
Descripción:	Esta función permite cambiar la estructura secundaria de una proteína.

**Tabla 4:** Descripción de la Clase: Vega.

**Nombre:** CC\_VegaDataManager

**Tipo de Clase:** Controladora

Atributo:	Tipo:
Buffer	Vector<Molecule>
moleculesInProcess	int
Glanularity	int
totalmolecules	int

TMPDIRECTORY	File
PROBLEMLOG	File
StarTime	long
EndTime	long
TOTAL_WALL_CLOCK_TIME	long
storageResults	int
collectedResults	int
clientlog	String
<b>Responsabilidades</b>	
1. Nombre	fillBuffer()
Descripción:	Encargado de dar valores al buffer que contiene los datos a ser procesados.
2. Nombre	generateWorkUnit(ClientInfo arg0)
Descripción:	Encargado de repartir la unidad de trabajo a un cliente, mediante un vector con el contenido o el tipo que objeto que procesará.
3. Nombre	processResults(Long arg0, Vector arg1)
Descripción:	Encargado de procesar los resultados, terminar de realizar las operaciones indicadas con los mismos.
4. Nombre	adjustGranularity( double percent )

Descripción:	Encargado de ajustar la cantidad de trabajo por cliente. Dependiendo del tiempo de respuesta o de la eficiencia con que lo haga.
5. Nombre	closeResources()
Descripción:	Encargado de terminar el trabajo con un cliente determinado.
6. Nombre	getStatus()
Descripción:	Retomara el estado real en que se encuentre la solución del problema.
7. Nombre	endCalculation()
Descripción:	Recoge y calcula el tiempo consumido por la ejecución.
8. Nombre	limpiarTMPDIRECTORY()
Descripción:	Borra la carpeta temporal que se crea con la ejecución del programa.
9. Nombre	recursiveDelete(File f)
Descripción:	Encargado de borrar el fichero que se le pase en f.
10. Nombre	saveLog(String logString)
Descripción:	Registra todos los mensajes.
11. Nombre	comand()
Descripción:	Se encarga de leer el comando que entre el usuario.

**Tabla 5:** Descripción de la Clase: CC\_Vegadatamanager.

**Nombre:** CC\_VegaAlgorithm

**Tipo de clase:** Controladora

<b>Atributo:</b>	<b>Tipo:</b>
TMPDIRECTORY	File
Resultcarp	File
comando	String
tempcomando	String
Results	Vector<Resultado>
BUFFER_SIZE= 1024	static final int
log	String
<b>Responsabilidades:</b>	
1.Nombre	processUnit ( Vector moléculas )
Descripción:	Responsable del proceso de los datos que se reciben en los clientes.
2.Nombre	carpeta()
Descripción:	Encargado de crear una carpeta necesaria para guardar los ficheros .mol y los resultados de los cálculos realizados.
3.Nombre	Descargarvegaprog()
Descripción:	Responsable de descargar la aplicación Vega para que sea ejecutado en la PC cliente.
4.Nombre	LimpiarDirectorios()
Descripción:	Encargado de borrar los ficheros de las carpetas temporales que cree.
5.Nombre	crearficheromol(Vector moléculas)
Descripción:	Encargado de crear los ficheros .mol en el

	cliente
6.Nombre	recursiveDelete(File f)
Descripción:	Encargado de borrar el fichero que se le pase en f.
7.Nombre	RunVega()
Descripción:	Es el responsable de mandar a ejecutar la aplicación Vega.
8.Nombre	comandos(String Filename)
Descripción:	Encargado de crear el comando con que se va a ejecutar el Vega.
9.Nombre	Resultados()
Descripción:	Es el encargado de poner en un Vector los ficheros resultados.
10.Nombre	getResultPos(int name)
Descripción:	Es el encargado de ver que posición tiene un resultado dado su nombre.
11.Nombre	descompactar(String from, String to)
Descripción:	Es el responsable de descompactar el Vega pasándole la dirección donde se encuentra el .zip y el lugar donde lo vas a descompactar.
12.Nombre	saveLog(String logString)
Descripción:	Es el encargado de salvar los sucesos que ocurren cuando se ejecuta el Vegadatamanager o el Vegaalgorithm.

**Tabla 6:** Descripción de la Clase: CC\_Vegaalgorithm.

<b>Nombre: CE_Molecule</b>	
<b>Tipo de clase:</b> Entidad	
<b>Atributo:</b>	<b>Tipo:</b>
Name	String
Data	String
serialVersionUID=1L	static final long
<b>Responsabilidades:</b>	
1. Nombre	getData()
Descripción:	Encargado de devolver el valor de los datos que conforman a la molécula.
2. Nombre	setData(String data)
Descripción:	Encargado de dar valores a las moléculas.
3. Nombre	getName()
Descripción:	Encargado de devolver el nombre de la molécula.
4. Nombre	setName(String name)
Descripción:	Encargado de darle nombre a la molécula.

**Tabla 7:** Descripción de la Clase: CE\_Molecule.

<b>Nombre: CE_Resultado</b>	
<b>Tipo de clase:</b> Entidad	
<b>Atributo:</b>	<b>Tipo:</b>
Name	int
Data	String

Extension	String
ProcessTime	long
<b>Responsabilidades:</b>	
1. Nombre	getData()
Descripción:	Encargado de devolver el valor de los datos que conforman el resultado.
2. Nombre	setData(String data)
Descripción:	Encargado de dar valores a los resultados.
3. Nombre	getName()
Descripción:	Encargado de devolver el nombre del resultado.
4. Nombre	setName(int name)
Descripción:	Encargado de darle nombre a los resultados.
5.Nombre	getExtension()
Descripción:	Responsable de devolver la extensión de los resultados.
6.Nombre	setExtension(String extension)
Descripción:	Encargado de ponerle la extensión a los resultados.

**Tabla 8:** Descripción de la Clase: CE\_Resultado.

<b>Nombre:</b> CI_Resultado
<b>Tipo de Clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz del Especialista, brindada para que el usuario vea los resultados que se han obtenido hasta el momento y darle la posibilidad de descargar o eliminar dicho resultado.

**Tabla 9:** Descripción de la Clase: CI\_Resultado.

<b>Nombre:</b> CI_Execution_problems
<b>Tipo de clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz del Especialista, brindada para que el usuario vea todos los problemas que están en ejecución, y además para dar la posibilidad de mostrar el estado de estas ejecuciones o para mandar a interrumpir una ejecución deseada.

**Tabla 10:** Descripción de la Clase: CI\_Execution\_problems.

<b>Nombre:</b> CI_Vegavisual
<b>Tipo de clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz Principal del Especialista, brindada para que el usuario realice todas las operaciones que desee, como es el caso de mandar a ejecutar un problema, mandar a ver todos los problemas que están ejecutándose así como mandar a ver todos los resultados que se han obtenido hasta el momento, también da la posibilidad de mostrarle una ayuda al usuario.

**Tabla 11:** Descripción de la Clase: CI\_Vegavisual.



<b>Nombre:</b> CI_loguin
<b>Tipo de clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz del Especialista, brindada para que el usuario se autentique y pueda conectarse a la plataforma.

**Tabla 12:** Descripción de la Clase: CI\_loguin.

<b>Nombre:</b> CI_Ayuda
<b>Tipo de clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz del Especialista, brindada para que el usuario vea la ayuda que brinda el Vega.

**Tabla 13:** Descripción de la Clase: CI\_Ayuda.

<b>Nombre:</b> CC_Visualcontrol	
<b>Tipo de Clase :</b> Controladora	
<b>Atributo:</b>	<b>Tipo:</b>
user	UserAuthentication
host	ResquestingHost
con	SystemConnection
<b>Responsabilidades:</b>	
1. Nombre	Conectar(String nick,String pass,String server,int port1,int port2)
Descripción	Encargado de crear la conexión con la plataforma.
2. Nombre	Agregarcomando(File comand,String comando,File[] ficheros)
Descripción	
3. Nombre	ejecutar(File[] ficheros)

Descripción	Encargado de mandar a ejecutar la aplicación.
4. Nombre	Pararexecution(Myexecution ew)
Descripción	Encargado de para la ejecución de un problema seleccionado.
6. Nombre	Descargarresult(Myolution sw)
Descripción	Encargado de descargar un resultado deseado.
6. Nombre	Eliminarresult(Myolution sw)
Descripción	Encargado de eliminar un resultado deseado.

**Tabla 14:** Descripción de la clase: CC\_ Visualcontrol

<b>Nombre:</b> CI_Estado de ejecución.
<b>Tipo de Clase:</b> Interfaz
<b>Responsabilidades:</b>
Interfaz del Especialista, brindada para que el usuario vea el estado en que se encuentran los problemas que están ejecutándose.

**Tabla 15:** Descripción de la clase: CI\_Estado de ejecución.

## Anexo 2: Prototipos de Interfaz



Loguin

Usuario

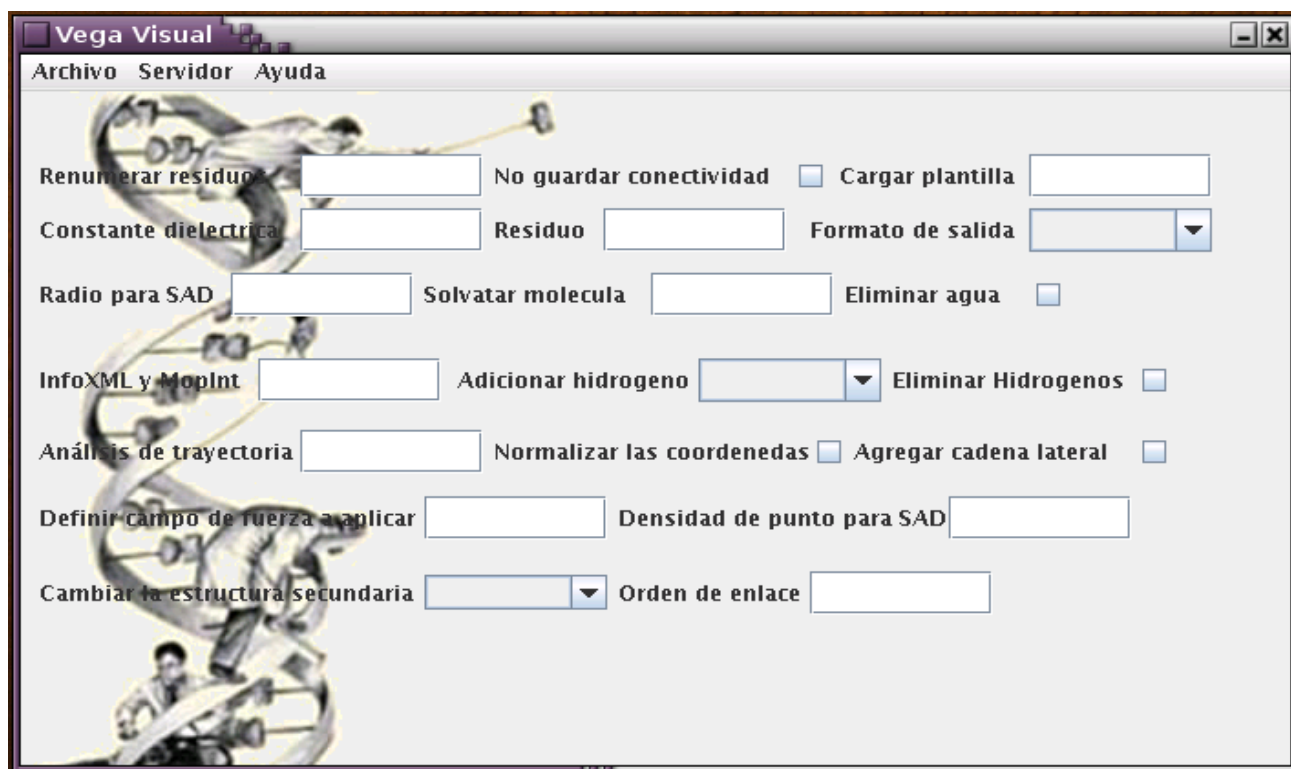
Contraseña

Servidor

Puerto RMI

Puerto socket

Figura 29: Prototipo de Interfaz de la CI\_Loguin.



Vega Visual

Archivo Servidor Ayuda

Renumerar residuo  No guardar conectividad  Cargar plantilla

Constante dieléctrica  Residuo  Formato de salida

Radio para SAD  Solvatar molecula  Eliminar agua

InfoXML y MopInt  Adicionar hidrogeno  Eliminar Hidrogenos

Análisis de trayectoria  Normalizar las coordenadas  Agregar cadena lateral

Definir campo de fuerza a aplicar  Densidad de punto para SAD

Cambiar la estructura secundaria  Orden de enlace

Figura 30: Prototipo de Interfaz de la CI\_Vega Visual.

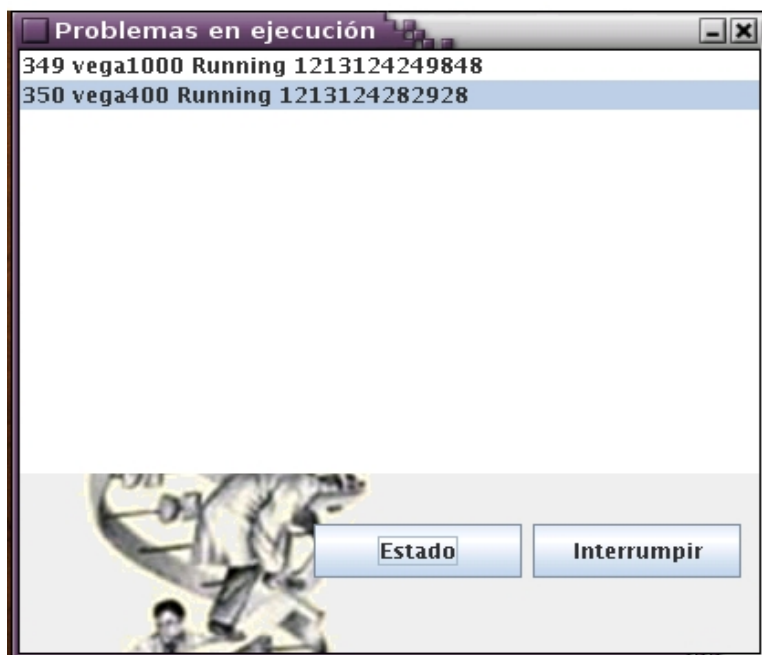


Figura 31: Prototipo de Interfaz de la CI\_Problemas en ejecución.

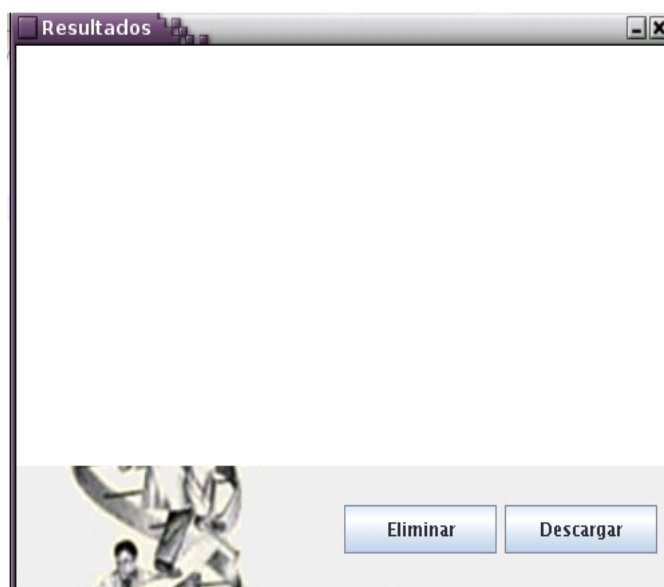
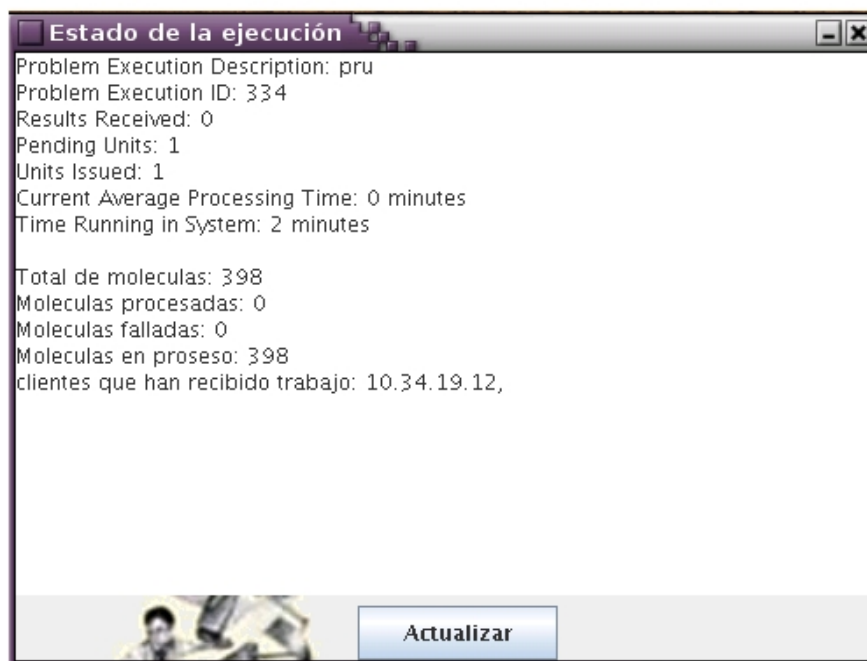


Figura 32: Prototipo de Interfaz de la CI\_Resultados.



**Figura 33:** Prototipo de Interfaz de la CI\_ Estado de Ejecución

## **GLOSARIO DE TÉRMINOS**

**JDT:** es el IDE de Java.

**IDE:** Es el Entorno de Desarrollo Integrado.

**CVS:** son los Sistemas de Control de Versiones.

**Código Abierto:** es el término con el que se conoce al software distribuido y desarrollado libremente.

**LGPL:** significa Licencia Pública General Limitada de GNU.

**Marshaling:** puesta en orden, ordenación.

**Herramienta CASE:** (**C**omputer **A**ided **S**oftware **E**ngineering, Ingeniería de Software Asistida por Ordenadores). Son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.

**Plugin:** Un plugin o componente enchufable es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad muy específica.

**GRID:** Grilla. Función que divide un área de trabajo en cuadrículas.