Universidad de las Ciencias Informáticas

Facultad 6

Título: "BioSyS: Implementación del Módulo de Simulación."

Trabajo de Diploma para optar por el título de Ingeniero Informático.

Autor(es): Mabel Navarro Bermúdez

Yissel Rodríguez Aldana

Tutor(es): Msc. Noel Moreno Lemus.

Junio, 2008

"El libro de la naturaleza está escrito en lenguaje matemático".

Galileo Galilei

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 26 días del mes de junio del año 2008.

Mabel Navarro Bermúdez

_____

Firma del Autor

Yissel Rodríguez Aldana

_____

Firma del Autor

Noel Moreno Lemus

_____

Firma del Tutor

DATOS DE CONTACTO

Noel Moreno Lemus: Msc. En Ciencias, Licenciado en Química Nuclear.
Correo electrónico: noel@uci.cu
Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

DEDICATORIA

A mi mamita por ser guía y ejemplo de superación todos estos años, por la ayuda, el cariño infinito que me ha dado, por ser la razón principal de mi empeño por superarme. A ti especialmente dedico mi esfuerzo en estos años.

A mi papá por su comprensión, cariño y apoyo en todo momento.

A mis cuatro abuelos, especialmente a mima y pipo que sé cuanto los enorgullece mi superación.

A mi tía Odalys y mi tío Pepito por estar siempre que los he necesitado.

A mi primito Ale…para que siga el ejemplo de su tata y estudie siempre.

A mi novio por su ayuda y sobre todo su paciencia.

A todos mis amigos, en especial a Irilys y Tita.

Mabel

Quiero dedicar este trabajo a la Yele y a la Mary que la vida no quiso que tuviésemos la misma sangre pero igual son mis hermanas.

A mis hermanos y mi novio que siempre estuvieron ahí cuando más los necesité dándome todo su amor.

Y sobre todo a mis padres que siempre me impulsaron a querer más de la vida y me guiaron con su ejemplo.

Y a Mabe por soportarme. Linda.

A todos gracias.

Tita

AGRADECIMIENTOS

Agradecemos a todos aquellos que de una forma u otra han hecho posible el desarrollo de este trabajo, en especial a nuestro tutor Noel por su asesoramiento científico, su paciencia, esmero y ayuda.

A Edel por su disposición incondicional y su infinita ayuda.

A nuestros familiares y amigos, por su constante apoyo, comprensión y soporte durante todos estos años.

A nuestro comandante Fidel Castro Ruz por habernos permitido realizar nuestros sueños en esta Universidad.

Resumen

El mayor reto al que se enfrentan los investigadores que trabajan en el campo de la Biología de Sistemas es al problema de tratar de entender el funcionamiento de los sistemas biológicos como un todo. La simulación computarizada de estos sistemas ofrece la posibilidad de comprimir el tiempo, esfuerzo y cantidad de recursos necesarios para realizar las investigaciones de los mismos.
Muchas herramientas han sido desarrolladas para este fin o incluyen funcionalidades para simular sistemas biológicos. Sin embargo la mayoría de estos sistemas son muy específicos, los mejores no se comercializan o no incluyen determinadas funcionalidades como: la realización de simulaciones muy complejas y el almacenamiento de los resultados de los estudios realizados.

En la versión 1.0 de del proyecto BioSyS se implementaron algoritmos que permitían realizar simulaciones distribuidas. Esta implementación, ha servido para realizar diversos estudios, pero tiene ciertas limitaciones como son:

1. La aplicación no se puede cerrar mientras se está simulando, incluso aunque la máquina donde está BioSyS no esté realizando ningún cálculo.
2. La aplicación no chequea el trabajo del procesador de las máquinas que manda a simular, sencillamente inicia los cálculos, esto puede entorpecer el trabajo de otros usuarios.
3. Se necesitan muchos datos para configurar las máquinas a utilizar.

En el presente trabajo de diploma se presenta el Módulo de Simulación de BioSyS que pone en manos de los investigadores una herramienta que permite la simulación computacional de sistemas biológicos descritos por Sistemas de ecuaciones diferenciales y utiliza la infraestructura de cálculo distribuido desplegada en la UCI para apoyar la realización de los cálculos y resolver aquellos problemas que son computacionalmente costosos.

TABLA DE CONTENIDO

Introducción

Uno de los mayores retos a los que se enfrentan los investigadores de la Biología de Sistemas es al de comprender el funcionamiento de los Sistemas Biológicos como un todo. Debido al elevado nivel de complejidad de los mismos se han utilizado herramientas de modelación gráficas y matemáticas que permiten representar el problema de una forma relativamente sencilla. En el caso de los modelos gráficos pueden enfocarse desde diferentes niveles de abstracción: moléculas, células y tejidos hasta organismos, poblaciones y ecosistemas.

Las herramientas matemáticas también pueden variar y el uso de una u otra depende del objetivo final que se persiga con las simulaciones posteriores. Algunas de estas herramientas son:

1. Autómatas celulares.
2. Ecuaciones en diferencias o mapas (ecuaciones recursivas).
3. Ecuaciones diferenciales en derivadas parciales.
4. Sistemas de Ecuaciones Diferenciales (SED).

Estos últimos han sido de los más utilizados en la modelación de sistemas biológicos debido a las facilidades de modelación y simulación que brindan. Con SED se pueden describir gran cantidad de procesos biológicos y los métodos para la resolución de los mismos son muy conocidos y están disponibles tanto en la literatura como en diversos sistemas software.

Dentro de los sistemas software existentes para resolver este tipo de sistemas matemáticos destacan MatLab, el Mathematica, el Octave, así como librerías implementadas en diferentes lenguajes como ODEtoJava, en Java, ODESolver, en C.

Resolver un SED requiere que se defina un juego de parámetros y de condiciones iniciales, problema de Cauchy. Cuando se quiere hacer un estudio intensivo de un sistema biológico que haya sido descrito mediante SED es necesario variar los parámetros y las condiciones iniciales y realizar las simulaciones correspondientes a cada combinación posible. Este trabajo no es posible hacerlo utilizando una sola computadora debido a la cantidad de cálculos que se requiere. Esto evidencia la necesidad de utilizar una infraestructura distribuida, pues en la misma medida en que aumente el número de máquinas a utilizar disminuirá el tiempo requerido para la realización de los cálculos.

Los sistemas software y las librerías antes mencionados solo permiten resolver un sistema cada vez. Existen sistemas software más específicos, dedicados a la simulación de SED, que tratan de resolver este problema mediante algoritmos de fuerza bruta, es decir realizar todas las simulaciones posibles, un ejemplo es el CellDesigner. Por lo general todos los sistemas software que se dedican a realizar

muchas simulaciones hacen uso de alguna infraestructura de cálculo distribuido para afrontar este problema, por lo general son infraestructuras propias de las compañías que desarrollan estas herramientas y no se tiene acceso a las mismas.

Para darle solución a este problema se implementaron en la versión 1.0 de BioSyS algoritmos que permitían realizar simulaciones distribuidas haciendo uso de las máquinas conectadas en una Red local, utilizando para la comunicación entre ellas el protocolo de comunicación SSH. Esta implementación, ha servido para realizar diversos estudios, pero tiene ciertas limitaciones como son:

1. La aplicación no se puede cerrar mientras se está simulando, incluso aunque la máquina donde está BioSyS no esté realizando ningún cálculo.
2. La aplicación no chequea el trabajo del procesador de las máquinas que manda a simular, sencillamente inicia los cálculos, esto puede entorpecer el trabajo de otros usuarios.
3. Se necesitan muchos datos para configurar las máquinas a utilizar.

Teniendo en cuenta esta problemática se plantea como **problema a resolver** en el presente trabajo de diploma, el siguiente:

¿Cómo automatizar la realización de múltiples simulaciones de Sistemas Biológicos descritos mediante SED haciendo uso del sistema de cómputo distribuido desplegado en la UCI?

Para resolver este problema nos hemos centrado en el **Objeto de Estudio**: La Simulación de Sistemas Biológicos y el **Campo de Acción:** La Simulación de Sistemas Biológicos descritos por SED.

Como **Objetivo General**: Implementar un módulo para realizar múltiples simulaciones a Sistemas Biológicos en ambientes distribuidos. **Los Objetivos Específicos** que se derivan son los siguientes:

1. Realizar el análisis del módulo de simulación.
2. Diseñar el módulo de simulación.
3. Implementar módulo de simulación.

Para resolver el problema planteado y lograr el cumplimiento de los objetivos se planificaron las siguientes **Tareas de la Investigación**:

1. Familiarización con los algoritmos de simulación implementados en la versión 1.0 de BioSyS.
2. Definición de los diferentes tipos de simulaciones a realizar sobre los sistemas de ecuaciones diferenciales.
3. Estudio del sistema de cómputo distribuido t-arenal y de las formas de implementación de algoritmos sobre este.
4. Levantamiento y descripción de los requisitos del módulo de simulación.

5. Diseño de las clases a implementar.
6. Definición de las interfaces a utilizar.
7. Implementación de las clases diseñadas.

El presente trabajo está estructurado en 4 capítulos cuyos contenidos son:

Capítulo 1: Fundamentación teórica. En este capítulo se explicarán los conceptos fundamentales de la simulación de Sistemas Biológicos, de los modelos matemáticos asociados a estos sistemas y de los métodos y herramientas matemáticas para resolverlos. Se demostrará la necesidad de utilizar una infraestructura distribuida para realizar múltiples simulaciones y en específico de la plataforma de cálculo distribuido desplegado en la UCI (t-arenal). Se tratará además las tendencias y tecnologías para desarrollar este módulo.

Capítulo 2: Características del sistema. En este capítulo se hará una descripción del sistema a automatizar y se mostrará un diagrama de casos de uso del sistema para un mejor entendimiento del mismo. Se definirán los actores y requerimientos funcionales del sistema que se implementará. Se plantearán los requerimientos mínimos que debe tener el sistema de cómputo donde se vaya a instalar la aplicación.

Capítulo 3: Análisis y diseño del sistema. En este capítulo se describen los patrones de diseño y de arquitectura que se tuvieron en cuenta durante el desarrollo de la aplicación y se evidencia su utilización. Se realiza el diagrama de clases del diseño para cada subsistema definido dando respuesta a la solución que se propone y se define el diagrama de despliegue con el objetivo de mostrar la distribución física de los nodos de cómputo que necesita la aplicación.

Capítulo 4: Implementación. En este capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, para esto se muestra el diagrama de componentes y el diagrama de despliegue detallado. Además se mostrará el código fuente de los principales métodos de la aplicación.

Capítulo 1: Fundamentación Teórica

Introducción

En este capítulo se explicarán los conceptos fundamentales de la simulación de Sistemas Biológicos, de los modelos matemáticos asociados a estos sistemas y de los métodos y herramientas matemáticas para resolverlos. Se demostrará la necesidad de utilizar una infraestructura distribuida para realizar múltiples simulaciones y en específico de la plataforma de cálculo distribuido desplegado en la UCI (t-arenal). Se tratará además las técnicas computacionales, las herramientas de desarrollo, metodología y lenguajes a utilizar, así como los roles desempeñados y los artefactos a realizar.

1.1 Simulación de Sistemas Biológicos

Simular un proceso mediante un ordenador nos proporciona una serie de ventajas, como poder examinar en unos segundos un proceso que dura millones de años (o, al revés, permitirnos ver durante algunos minutos una reacción producida en fotosegundos, pudiendo examinar los procesos allí desarrollados), nos permite someter a una población a unas situaciones difíciles de encontrar en la realidad (por ejemplo, provocando una contaminación en un bosque), conocer cómo pueden reaccionar unas células a un determinado tratamiento, etc. Desafortunadamente, nos encontramos también con una parte negativa: la realidad biológica no está exenta de complejidad, lo cual hace prácticamente imposible tener en cuenta todas las condiciones externas a la hora de realizar una simulación.

La simulación es una poderosa herramienta que es ampliamente utilizada por los científicos que estudian los sistemas complejos. Esta técnica incluye para estos estudios, tanto la construcción del modelo (gráfico o matemático), como su resolución y su uso analítico. [1]

1.1.1 Modelación de sistemas biológicos

Un buen modelo debe auxiliar al investigador proveyéndole pistas inesperadas y perspectivas novedosas, por medio de la unificación del conocimiento y la explicación de los fenómenos, y no debido sencillamente al mero recapitulamiento o descripción de los comportamientos biológicos ya conocidos. [2]

Los modelos son valiosos no tan sólo como integradores de los datos experimentales disponibles, sino también como herramientas que enfocan la atención del investigador sobre aquellos aspectos críticos de la investigación que se conocen imperfectamente, y que por lo tanto requieren seguimiento más profundo. [2]

El interés por desarrollar un modelo radica en la posibilidad de reproducir un fenómeno o predecir el funcionamiento de un sistema. Analizando las características de cada uno de los componentes, nos damos cuenta que modelar un sistema biológico resultará difícil y en algunos casos imposible de llevar a cabo. [3]

Como ya se explicó en la Introducción, los modelos de Sistemas Biológicos pueden ser agrupados en gráficos y matemáticos. Generalmente a partir del modelo gráfico realizado de un Sistema Biológico se puede generar un modelo matemático. El modelo matemático permite representar un problema médico o biológico de una manera objetiva definiendo una serie de relaciones matemáticas entre las mediciones cuantitativas (del problema) y sus propiedades.

1.1.2 Modelos matemáticos

Un modelo matemático es una descripción matemática de un fenómeno del mundo real, como puede ser el crecimiento de las poblaciones de animales, la concentración de un producto en una reacción química, el funcionamiento de las neuronas y la dinámica intracelular, por solo citar algunos ejemplos de su aplicación en biología. La finalidad de estos modelos radica en entender en profundidad el fenómeno y tal vez realizar alguna predicción sobre su comportamiento futuro. [4]

Algunas de las formas en que pueden ser descritos los modelos matemáticos son:

1. Autómatas celulares.
2. Ecuaciones en diferencias o mapas (ecuaciones recursivas).
3. Ecuaciones diferenciales en derivadas parciales.
4. Sistemas de Ecuaciones Diferenciales (SED).

Estos últimos han sido de los más utilizados en la modelación de sistemas biológicos debido a las facilidades de modelación y simulación que brindan. En el campo de la biología se ha reconocido durante mucho tiempo que la mayoría de los procesos biológicos se pueden describir en forma dinámica, es decir que suceden en asociación o en función de otros procesos. Tales relaciones dinámicas pueden expresarse convenientemente en forma matemática por medio de las ecuaciones diferenciales. Muchas situaciones involucran razones de cambio, por lo cual su modelo matemático estará dado por una ecuación diferencial o un conjunto de ellas.

El desarrollo de modelos matemáticos basados en sistemas de ecuaciones diferenciales ordinarias, constituye un enfoque particular del uso de modelos. Se basa en la resolución de estos sistemas de ecuaciones paso a paso a lo largo de un horizonte temporal mediante la aproximación de los valores que toman las variables de estado que definen la estructura del sistema a través del modelo, usando

métodos matemáticos que en forma iterativa aproximan la solución en cada punto para integrar la función en un subespacio de soluciones, donde tales soluciones tienen sentido biológico. [5]

La figura 1.0 ilustra el proceso de modelado matemático. Dado el problema que deseamos modelar, nuestra primera tarea consistirá en identificar las variables que intervienen y realizar suposiciones que simplifiquen el problema para poder abordarlo. El nivel de resolución del problema estará dado por el grado de simplificación que se realice, así una resolución baja significa que el problema ha sido muy simplificado y no es una representación ajustada a la realidad.

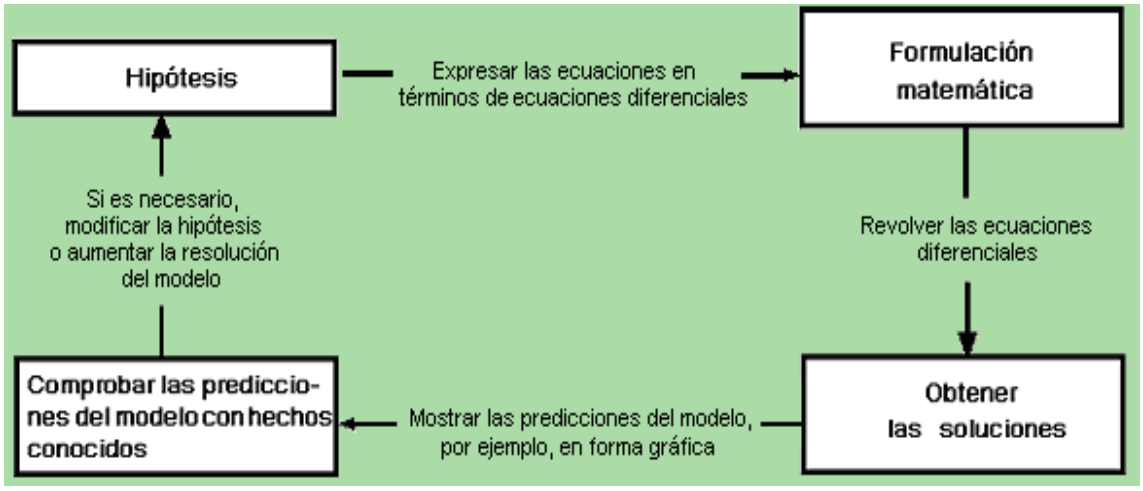


Figura 1.1 Proceso de modelado matemático.

1.1.3 Métodos de resolución de SED

Para comprender el comportamiento de los sistemas biológicos se deben analizar los resultados obtenidos del modelo matemático que lo describe. Existen distintos métodos para la resolución, de SED, algunos de ellos son:

1. Métodos de Runge-Kutta.
2. Métodos de Adams.

1.1.3.1 Métodos de Runge-Kutta

Basados en la Serie de Taylor, donde la cantidad de términos que se decidan tomar de la serie determinará la precisión del método. Estos métodos tienen un paso que puede ser fijo obteniéndose puntos equidistantes; y también puede variar haciendo aumentar el paso al doble en caso de detectar

errores pequeños o reducirlo a la mitad en presencia de errores grandes (paso autoajustable), esto complica el programa de cálculo, pero aumenta la eficiencia y en la mayoría de los casos son más rápidos.

El método de segundo orden requiere evaluar cada función 2 veces en cada paso de integración y el error local es del orden h3, de la misma forma el método de cuarto orden necesita 4 evaluaciones de cada función y posee un error local del orden h5; esto posibilita que se pueda integrar con un paso mayor para lograr un error similar al que obtendríamos si utilizásemos el de orden dos, mejorando así el tiempo de ejecución del algoritmo.

Los métodos de Runge-Kutta son métodos de paso simple, que sólo requieren de los resultados del paso anterior, esto les brinda la posibilidad de auto iniciarse ya que parten de las condiciones iniciales. Al igual que todos los métodos de paso simple no presentan inestabilidad numérica para paso h suficientemente pequeño, esto implica que pequeños cambios en las condiciones iniciales del sistema sólo originan cambios acotados en la solución. Una desventaja de los métodos Runge-Kutta es que requieren múltiples evaluaciones de la funciones en cada paso. [6]

1.3.1.2 Métodos de Adams

Utilizan procedimientos de integración numérica que hacen uso de un polinomio de interpolación para aproximar la función f(x, y(x)) de y' = f(x, y). Las diversas variantes de tomar este polinomio originan múltiples métodos de Adams. Estos métodos son de paso múltiple ya que necesitan de los valores iniciales en varios puntos equidistantes para su iniciación, esta información se puede obtener aplicando primero algún método de paso simple como los de Runge-Kutta, constituyendo una desventaja de los métodos de paso múltiple. Otra característica desventajosa de los métodos de Adams comparándolos con los de Runge-Kutta es que poseen un error local mayor en procedimientos del mismo orden, teniendo así menos precisión. Los métodos de Adams poseen estabilidad condicional. [6]

Debido a que el objetivo de este trabajo no es desarrollar nuevos métodos para la resolución de SED, solo se explican dos de ellos para ilustrar mejor el problema.

Al proceso de resolver un modelo matemático asociado a un sistema biológico mediante dichos métodos, es a lo que se le llama simulación de sistemas biológicos.

1.1.4 Herramientas Matemáticas para la resolución de SED

Dentro de la gran gama de asistentes matemáticos que se han venido desarrollando en la revolución de la informática, algunos han marcado pautas por su potencia de cálculo y sus prestaciones. Ejemplos de ellos son: el MatLab, el Mathematica y el Octave.

Estos han sido de los sistemas software más utilizados por científicos de diferentes ramas para resolver sus problemas, pues los mismos cuentan con un arsenal de herramientas ya programadas y de fácil utilización que le permiten a los usuarios centrarse en el problema concreto a resolver y olvidarse de los métodos numéricos y demás artefactos matemáticos que requiera la solución de dicho problema.

A continuación pasamos a detallar el uso de los mismos en la resolución de SED.

1.1.4.1 Mathematica

Mathematica es también un poderoso lenguaje de programación que emula múltiples paradigmas utilizando reescritura de términos. Utiliza bloques de código (librerías), para ampliar las capacidades y reorientar el cálculo.

El lenguaje de programación de Mathematica está basado en re-escritura de términos, y soporta el uso de programación funcional y de procedimientos (aunque en general, la programación funcional es más eficiente). Está implementado en una variante del Lenguaje de programación C orientado a objetos, pero el grueso del extenso código de librerías está en realidad escrito en el lenguaje Mathematica, que puede ser usado para extender el sistema algebraico añadiendo nuevo código en forma de paquetes.

El Mathematica implementa diferentes métodos de resolución de SED tanto numéricos como simbólicos, lo que lo convierte en una herramienta muy potente.

Usando estos métodos e implementando funciones propias es posible realizar estudios de sistemas dinámicos, pero, para ellos hay que tener un dominio profundo del Mathematica y de su lenguaje de programación, requisitos estos, muy alejados de los conocimientos que posee un investigador en ciencias biológicas.

A estas desventajas se le suman las exigencias de sus licencias, pues, las mismas se generan para la computadora específica en que se instale. [7]

1.1.4.2 Octave

Octave es un lenguaje de alto nivel para el cálculo numérico, siendo su sintaxis compatible con "MatLab", ampliamente utilizado por el entorno docente, pero desarrollado por la comunidad de software libre. Octave es un lenguaje especialmente orientado al mundo científico. Las características de Octave permiten que los algoritmos científicos se desarrollen en mucho menos tiempo que en otros lenguajes de programación. De este modo, Octave es el lenguaje ideal para el desarrollo de algoritmos de procesado digital de señales (voz, imagen, etc.), de sistemas de control, estadística u otros.

Aunque sea un lenguaje idóneo para desarrollar aplicaciones científicas, tiene algunas desventajas. Uno de los inconvenientes está relacionado con la velocidad de cómputo. Octave, al ser un lenguaje de programación interpretado, es más lento que un lenguaje compilado, ya que éste generaría instrucciones nativas para el procesador, que se ejecutarían más rápido.

Los algoritmos desarrollados y validados en Octave generalmente se tienen que decodificar en otro lenguaje para obtener aplicaciones finales de usuario. [7]

1.1.4.3 MatLab

MATLAB es la abreviatura de Matrix Laboratory (laboratorio de matrices). Es un programa de análisis numérico creado por The MathWorks en 1984. Está disponible para las plataformas Unix, Windows y Mac OS X.

Se pueden ampliar sus capacidades con Toolboxes, algunos de los cuales están destinados al procesado digital de señales, adquisición de datos, economía, inteligencia artificial, lógica difusa. También cuenta con otras herramientas como Simulink, que sirve para simular sistemas.

La primera versión surgió con la idea de emplear unos paquetes de subrutinas escritas en Fortran en los cursos de álgebra lineal y análisis numérico, sin necesidad de escribir programas en Fortran.
Usa un lenguaje de programación creado en 1970 para proporcionar un sencillo acceso al software de matrices LINPACK y EISPACK sin tener que usar Fortran. También tiene su propio compilador.

El MatLab cuenta con paquetes de funciones que permiten resolver múltiples problemas, además de que brinda la posibilidad de implementar funciones propias, haciendo uso del lenguaje de programación antes mencionado.

Para resolver SED existen en el MatLab ocho métodos numéricos los cuales se muestran en la figura 1.1.

Para poder utilizarlo en estudios de Biología de Sistemas se debe proceder de la misma forma que con el Mathematica, por lo que este asistente presenta exactamente los mismos inconvenientes que el anterior. [7]

Sin embargo brinda una gran cantidad de funcionalidades que determinaron nuestra decisión de utilizarlo como asistente matemático para realizar las simulaciones (resolución de SED) del módulo de simulaciones de BioSyS.

Las principales ventajas de MatLab pueden resumirse en:

1. Cálculos matemáticos interactivos.
2. Gráficas integradas.
3. Programación sencilla.
4. Entorno consistente con numerosas funciones propias.
5. Disponible para Windows, Unix (Linux, Solaris, BSD,...) y Mac
6. Sirve para cualquier área tecnológica (desde biología hasta Automoción).

El MatLab puede ser utilizado desde otros lenguajes de programación como C++, C# y Java por solo mencionar algunos. En el caso de Java que es el lenguaje que se ha utilizado en el desarrollo de BioSyS 1.0, permite una interacción sencilla con este asistente matemático pero solo en sistemas operativos GNU/Linux. Esto representa una desventaja si tenemos en cuenta que el sistema operativo más usado por los usuarios finales es Windows.

Por tal motivo, además del MatLab, se han buscado otras herramientas que permitan resolver SED sobre cualquier plataforma.
A continuación detallamos varias de ellas y explicamos la seleccionada.

1.1.4.4 ODESolver

Es una librería implementada en C++ que reúne un grupo de funciones que permiten al usuario resolver ecuaciones diferenciales utilizando lo métodos numéricos más conocidos (Euler, Runge-Kutta, Adams). Su alta modulación permite que pueda ser incluida, ser probada y debuggeada en otros códigos.

Es multiplataforma, puede ser utilizada en LINUX/UNIX, Windows NT-XP. A pesar de que con esta librería nuestra problemática se vería resuelta, la integración de la misma con el leguaje de programación escogido se convierte en un nuevo problema a resolver.

1.1.4.5 ODEToJava

El ODEToJava es un paquete de software, resolutor de Ecuaciones Diferenciales Ordinarias escrito en Java. Se puede utilizar para resolver problemas de valor inicial para Ecuaciones Diferenciales ordinarias rígidas y no rígidas.

Este paquete contiene diferentes variantes del conocido método de Runge-Kutta, que permiten resolver cualquier tipo de problema descrito mediante SED. El usuario puede influir en el proceso de solución en varios niveles probando variantes del método Runge-Kutta. ODEToJava provee un ambiente en el cual se puede estudiar métodos explícitos Runge-Kutta (para problemas no rígidos) y métodos explícitos -implícitos (para problemas rígidos). El uso de Java provee facilidades como la portabilidad, el diseño en capas orientado a objetos, que permite al desarrollador crear piezas que puede desechar o reutilizar fácilmente. Se integra fácilmente al software BioSyS 1.0, pues el mismo ha sido desarrollado íntegramente en Java, como se mencionaba anteriormente. Además, nos permite resolver SED en cualquier sistema operativo, pues la única limitante en cuanto a software es que en el sistema escogido este instalada la máquina virtual de Java.

Solucionador	Problema que resuelve	Método
Ode45	Ecuaciones Diferenciales Nonstiff	Runge-Kutta
Ode23	Ecuaciones Diferenciales Nonstiff	Runge-Kutta
Ode113	Ecuaciones Diferenciales Nonstiff	Runge-Kutta
Ode15s	Ecuaciones Diferenciales Stiff y DAEs	NDFs(BDFs)
Ode23s	Ecuaciones Diferenciales Stiff	Rosenbrock
Ode23t	Ecuaciones Diferenciales Stiff (moderado) y DAEs	Regla trapezoidal
Ode23tb	Ecuaciones Diferenciales Stiff	TR-BDF2
Ode15i	Ecuaciones diferenciales ímplícitas	BDFs

Tabla 1.1 Funciones matemáticas para resolver SED que utiliza MatLab.

Después de analizadas todas las variantes anteriores se ha decidido implementar en el módulo métodos de resolución de SED que hagan uso de MatLab y de la librería ODEtoJava. Será por tanto el usuario final el que, a la hora de simular elegirá tanto la herramienta a utilizar, como el método numérico.

Como ya se había mencionado, resolver un SED requiere que se defina un juego de parámetros y de condiciones iniciales, problema de Cauchy. Cuando se quiere hacer un estudio intensivo de un

sistema biológico que haya sido descrito mediante SED es necesario variar los parámetros y las condiciones iniciales y realizar las simulaciones correspondientes a cada combinación posible. Teniendo en cuenta que el alto grado de no linealidad de los modelos creados hace que las diferentes combinaciones puedan llevar a diferentes comportamientos.

Sin embargo estas herramientas matemáticas que se han ido mencionando, no están implementadas para realizar muchas simulaciones, solo permiten resolver un SED cada vez. Por lo tanto se hace necesario implementar funcionalidades que permitan resolver esta problemática.

Los detalles de esta implementación serán discutidos posteriormente pues son resultados del presente trabajo. No obstante creemos importante realizar un estudio de las posibles herramientas o técnicas computacionales a utilizar en la solución de este problema.

1.2 Técnicas Computacionales

El desarrollo acelerado de las redes de computadoras en los últimos años ha hecho reconsiderar la utilización de las costosas supercomputadoras para la ejecución de aplicaciones que demanden potencia de cómputo. Una simple computadora con memoria local y procesador de capacidades moderadas no es de mucha utilidad por sí sola, pero al ser conectada a otras máquinas a través de una red de interconexión suficientemente rápida, se exalta enormemente su utilidad ya que cientos o miles de máquinas podrían trabajar, como un equipo, realizando intensos cálculos para dar solución a un determinado problema.

Lo cierto es que un gran número de empresas, organizaciones y universidades de todo el mundo tienen agrupadas un conjunto de computadoras conectadas en red formando un conglomerado, comúnmente llamado clúster, para trabajar en la solución de problemas computacionalmente costosos. En Cuba, el Ministerio de Educación Superior aprobó la creación de clúster de computadoras en las Universidades de La Habana, Las Villas y Oriente para apoyar el cálculo masivo de datos biológicos. Así mismo existen clústeres en centros de investigación del país como por ejemplo en el CIGB, el CIM y Bioinfo.

1.2.1 Clúster

Un clúster está compuesto de hardware, sistemas operativos, middleware y el programa que da solución a un determinado problema haciendo uso del procesamiento paralelo. El Middleware es la capa de software que se sitúa entre la capa de aplicación y las capas inferiores (Figura 2.0) para brindar lo que se conoce como Single System Image (SSI). También debe ser responsable de realizar

un balance de carga para distribuir los cómputos y recuperarse ante las fallas que puedan tener algunos componentes de las capas inferiores. [8]

Figura 1.2 Arquitectura de los Clúster de Computadoras.

Clúster HPC: Son diseñados para solucionar “grandes” problemas en un tiempo relativamente reducido y son conocidos como High Performance Computing (HPC). [8]

Clúster HTC: High Throughput Computing (HTC) utilizan el tiempo ocioso que tienen las computadoras. Estos están diseñados más bien para explotar la mayor cantidad de recursos computacionales posibles y dar solución a un gran número de problemas un poco más “pequeños” para los cuales no se tiene tanta premura. Sin embargo, con el incremento de las prestaciones de hardware que están teniendo las computadoras secuenciales, son cada vez más los problemas que se están atacando con HTC. Como ejemplo se pueden mencionar los problemas de renderización de imágenes, simulaciones computacionales y procesamiento masivo de datos experimentales. [8]

1.2.2 Computación Grid

La Computación Grid o Grid Computing es una tecnología que ha emergido como el paradigma dominante para ofrecer, de manera transparente y segura, la funcionalidad necesaria que permita compartir y explotar los recursos computacionales (superordenadores, clúster, medios de almacenamiento, software, instrumentos, etc.) a través de Internet. Según la definición de uno de los

padres del Grid, Ian Foster, “un Grid es un sistema que coordina recursos, que no están sujetos a un control centralizado, usando interfaces y protocolos estándares, abiertos y de propósito general para proveer de servicios relevantes”. [9]

Es importante resaltar que la tecnología Grid no pretende sustituir a los clúster de computadoras ya que su ámbito de aplicación es diferente. El objetivo de la misma es unir de forma desacoplada los recursos de diferentes dominios de administración, respetando sus políticas de seguridad y herramientas de gestión internas, y no solo se refiere a capacidad de procesamiento y almacenamiento sino todo tipo de recursos disponibles en las redes de computadoras. [9]

La Computación Grid está creada con el fin de brindar una solución a determinadas cuestiones, como problemas que requieren de un gran número de ciclos de procesamiento o acceso a una gran cantidad de datos. Encontrar un hardware y un software que permitan brindar estas utilidades comúnmente proporciona inconvenientes de costos, seguridad y disponibilidad. En ese sentido se integran diferentes tipos de máquinas y de recursos, por lo tanto una red Grid nunca queda obsoleta, todos los recursos se aprovechan. Si se renuevan todas las PCs de una oficina, se pueden incorporar las antiguas y las nuevas.

Por otra parte, esta tecnología brinda a las empresas el beneficio de la velocidad, lo que supone una ventaja competitiva, con lo cual se provee una mejora de los tiempos para la producción de nuevos productos y servicios.

Facilita la posibilidad de compartir, acceder y gestionar información, mediante la colaboración y la flexibilidad operacional, aunando no sólo recursos tecnológicos dispares, sino también personas y aptitudes diversas. Otro de los aspectos al que se tiende es a incrementar la productividad otorgando a los usuarios finales acceso a los recursos de computación, datos y almacenamiento que necesiten, cuando los necesiten. [9]

Teniendo en cuenta las ventajas que proporciona esta técnica computacional en nuestro centro se ha desarrollado una plataforma de cálculo distribuido (t-arenal) basada en la Computación Grid.

1.3 t-arenal

Se trata de un sistema de cómputo distribuido programado en Java. Ha sido utilizado para dar solución a varios problemas de la Bioinformática y brinda un modelo de programación de alto nivel basado en el paradigma de la POO utilizando RMI para el paso de mensaje. El funcionamiento básico del sistema es el siguiente: un módulo servidor que radica en el nodo maestro del sistema espera solicitud de trabajo. Las solicitudes son realizadas por un módulo cliente que se encuentra en los nodos esclavos.

Una vez hecha la solicitud, el servidor genera una unidad de trabajo que entrega al cliente para que este la procese y de una respuesta parcial que se envía al servidor para que sea integrada y se obtenga la solución al problema original. A través de una interfaz gráfica se pueden enviar los cómputos a correr de forma distribuida y descargar los resultados. [10]

Características esenciales del sistema:

1. Transparencia.
2. Eficiencia.
3. Flexibilidad.
4. Escalabilidad.
5. Fiabilidad.
6. Grupos de usuarios con privilegios según el rol que desempeñen.
7. Autorización de elementos de cómputos.
8. Conexión desde otros sistemas software.

1.3.1 Ventajas

1. Es software libre, de fácil instalación, se dispone del código fuente y se le pueden realizar modificaciones y distribuir sin problemas.
2. El sistema es capaz de gestionar los recursos de cómputo independientemente del sistema operativo, el tipo de red y la arquitectura de hardware que tengan sus nodos sin necesidad de recopilar el código fuente.
3. El sistema es tolerante a fallas, es decir, es capaz de operar en un ambiente inseguro donde cada nodo de trabajo puede desconectarse de la red. En caso de que una máquina este trabajando en un cómputo y el usuario la desconecte, apague o reinicie, todo el cómputo no se pierde ya que el sistema detecta el error y se recupera enviando una copia de la unidad de trabajo perdida a otro nodo o a esa misma máquina si se reincorpora.
4. El módulo cliente que se ejecuta en los nodos de trabajo es capaz de detectar las nuevas versiones de software colocadas en el nodo maestro y actualizarse de forma automática.
5. Es posible reajustar la granularidad o tamaño de las subtareas en que se descompone el problema de forma dinámica en dependencia del comportamiento y el rendimiento que se esté obteniendo en el cómputo distribuido.
6. El sistema es de propósito general, es decir, que puede ser utilizado para dar solución a cualquier problema que pueda ser descompuesto en subtareas independientes. Además, se puede cambiar el algoritmo que utilizan los nodos esclavos para realizar los cálculos sin necesidad de reiniciar todo el sistema.

7. Se pueden correr varios cómputos distribuidos de forma simultánea y darle a prioridad a los problemas que demandan de mayor procesamiento.
8. Es posible administrar y gestionar todo el sistema de cómputo de forma remota mediante una aplicación de interfaz gráfica.
9. Los usuarios que interactúan con el sistema para hacer corridas distribuidas que den solución a un problema pueden ver el progreso de la ejecución en tiempo real y descargar los resultados una vez que se haya concluido.
10. Un usuario potencial que no tenga habilidades o conocimientos en el procesamiento paralelo puede hacer corridas distribuidas mediante la interfaz gráfica del sistema para dar solución a un determinado problema.
11. Existe un simple procedimiento que pueden seguir los programadores para diseñar e implementar nuevas aplicaciones que distribuyan los cómputos para dar solución a problemas de computación intensiva.

Para ejecutar un cómputo distribuido, el programador necesita solamente extender dos clases en Java: DataManager y Algorithm, que vienen programadas y predefinidas en el sistema. También se pueden usar bibliotecas adicionales de Java u otras clases definidas por el usuario. Así como entender los principales métodos de estas clases. [10]

1.4 Herramientas y metodologías utilizadas en la aplicación

1.4.1 Metodologías de desarrollo de software

Un proceso de desarrollo de software define *quién* está haciendo *qué*, *cuándo* y *cómo* alcanzar un determinado objetivo; en la ingeniería de software el objetivo es construir un producto de software o mejorar uno existente con calidad. [11]

En el desarrollo de un software los desarrolladores se enfrentan a un reto importante, desarrollar con calidad y en el menor tiempo posible. Con el transcurso del tiempo y a medida que se revoluciona la producción de software a nivel mundial, se han ido creando metodologías y procesos que aceleran, fortalecen y mejoran la calidad de la producción.

Todo proceso de desarrollo de software es riesgoso y difícil de controlar, de ahí la necesidad de tener una metodología que nos garantice cumplir con dos cosas fundamentales: Con los planes de producción del software y la satisfacción de nuestro cliente. El hecho de utilizar en nuestros proyectos una metodología en verdad no constituye una receta, pero no deja de ser algo de suma importancia si

queremos ser competentes en el mercado del software. [11]

A continuación describimos tres de las metodologías más utilizadas en este tipo de proceso.

1.4.1.1 Proceso Unificado de Desarrollo

RUP es un proceso de desarrollo de Software, o sea, es el conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema software. Sin embargo, el Proceso Unificado de desarrollo de software es más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organizaciones, diferentes niveles de aptitud y diferentes tamaños de proyecto.

Las tres características principales de RUP son: dirigido por casos de uso, centrado en la arquitectura, e iterativo e incremental. [11]

1.4.1.2 Metodología XP

La programación extrema o XP2 es una metodología reciente en el desarrollo de software. La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo.

XP fue inicialmente creada para el desarrollo de aplicaciones donde el cliente no sabe muy bien lo que quiere, lo que provoca un cambio constante en los requisitos que debe cumplir la aplicación. Por este motivo es necesaria una metodología ágil como XP que se adapta a las necesidades del cliente y donde la aplicación se va reevaluando en cortos períodos de tiempo. [12]

XP está diseñada para el desarrollo de aplicaciones que requieran un grupo de programadores pequeño, donde la comunicación sea más factible que en grupos de desarrollo grandes. La comunicación es un punto importante y debe realizarse entre los programadores, los jefes de proyecto y los clientes. Las características esenciales de esta metodología son las siguientes:

1. **Comunicación:** Los programadores están en constante comunicación con los clientes para satisfacer sus requisitos y responder rápidamente a los cambios de los mismos. Muchos problemas que surgen en los proyectos se deben a que después de concretar los requisitos que debe cumplir el programa no hay una revisión de los mismos, pudiendo dejar olvidados puntos importantes.

2. **Simplicidad:** Codificación y diseños simples y claros. Muchos diseños son tan complicados que cuando se quieren ampliar resulta imposible hacerlo y se tienen que desechar y partir de cero.
3. **Realimentación (Feedback):** Mediante la realimentación se ofrece al cliente la posibilidad de conseguir un sistema apto a sus necesidades ya que se le va mostrando el proyecto a tiempo para poder ser cambiado y poder retroceder a una fase anterior para rediseñarlo a su gusto.[13]

1.4.1.3 OpenUp

Es una versión más ágil de lo qué es el RUP, es un Proceso Unificado que aplica propuestas iterativas e incrementales dentro del ciclo de vida, tratando de ser manejable en relación con el RUP. Plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto. Y lo que principalmente plantea es que se debe utilizar solo los procesos que sean necesarios y en este caso se necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no es necesario para que el proyecto no tenga errores y con eso evitar entregar al usuario final un producto de mala calidad, aunque este tipo de método plantea además que no se deben utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo ser este modificado, mejorado y extendido.

OpenUp tiene dos ventajas importantes, la primera es que este tipo de método disminuye los riesgos y la otra ventaja es que puede utilizarse tanto en proyectos pequeños como en proyectos grandes y si se lo maneja con cuidado y con profesionalismo se puede desarrollar un software de gran calidad a pesar de que se lo diseñe en poco tiempo y con poca documentación.

Una de las finalidades que se busca con el desarrollo del presente trabajo es tener un software funcional en un lapso de tiempo corto, disminuyendo los riesgos que eso pueda implicar, por lo que se ha decidido que se utilizará esta metodología en el proceso de desarrollo de software.[14]

1.4.2 Lenguaje de Modelado

Como lenguaje de modelado se utilizará UML (Unified Modeling Language), que es lenguaje que utiliza OpenUp, es un lenguaje para especificar, construir, visualizar y documentar los artefactos (información que es utilizada o producida mediante un proceso de desarrollo de software) de un sistema de software orientado a objetos, que por su potencialidad se ha convertido en un estándar.

Uno de los objetivos principales de la creación de UML era posibilitar el intercambio de modelos entre

las distintas herramientas CASE orientadas a objetos del mercado. Para ello era necesario definir una notación y semántica común. UML también 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 para el desarrollo. [15]

1.4.3 Herramientas CASE

Las Herramientas CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por Ordenador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero.
Existe una larga lista de Herramientas CASE, Rational Rose y Visual Paradigm son algunas de ellas.

1.4.3.1 Rational Rose

Rational Rose es una herramienta software para el Modelado Visual mediante UML de sistemas software. Permite especificar, analizar y diseñar el sistema antes de codificarlo. Rational Rose es la herramienta CASE desarrollada por los creadores de UML (Booch, Rumbaugh y Jacobson), que cubre todo el ciclo de vida de un proyecto: concepción y formalización del modelo, construcción de los componentes, transición a los usuarios y certificación de las distintas fases.
Características Rational Rose:

1. “Mantiene la consistencia de los modelos del sistema software.
2. Chequeo de la sintaxis UML.
3. Generación documentación automáticamente.
4. Generación de código a partir de los modelos.
5. Ingeniería Inversa (crear modelo a partir código).” [16]

Resulta algo complicado de utilizar esta herramienta, debido a que propone el empleo del proceso de desarrollo unificado, el cual es altamente complejo e incorpora algunas funcionalidades a través de otras aplicaciones pero no las integra. Además está disponible solamente para sistema operativo Windows.

1.4.3.2 Visual Paradigm

Es una herramienta CASE que utiliza “UML”: como lenguaje de modelado. Se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, JBuilder, NetBeans IDE.

Visual Paradigm ofrece:

1. “Entorno de creación de diagramas para UML 2.0.
2. Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
3. Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
4. Capacidades de ingeniería directa (versión profesional) e inversa.
5. Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
6. Disponibilidad de múltiples versiones, para cada necesidad.
7. Disponibilidad de integrarse en los principales IDEs.
8. Disponibilidad en múltiples plataformas. ”[17]

Como el sistema operativo que se está utilizando es Ubuntu (distribución de GNU/Linux) se decidió utilizar el Visual Paradigm para visualizar y diseñar los elementos de software debido a que es multiplataforma y por las facilidades que brinda para el diseño de los diagramas necesarios y su documentación.

1.4.4 Lenguajes de programación

Durante las últimas dos décadas, C y C++ han sido los lenguajes más utilizados para desarrollar software. Estos lenguajes ofrecen una gran flexibilidad pero, en cambio, la productividad no es muy alta ya que requieren mucho tiempo de desarrollo.

Si bien un aspecto importante a tener en cuenta es el tiempo de desarrollo otro aspecto mucho más importante es la portabilidad, es decir usar la misma aplicación en distintas arquitecturas o sistemas operativos sin tener que recompilar.

Java y C Sharp (C#) son lenguajes de programación que cumplen con las características antes mencionadas. Cuando se refiere a tiempo de desarrollo podríamos decir que C# es superior a Java, pero en cuanto a la portabilidad no sucede lo mismo:
Existen algunos proyectos como mono y portable.NET para hacer de C# un lenguaje portable. Mono es una máquina virtual que emula al .NET y trata de competir con Java en cuanto a ser multiplataforma, .NET aporta funcionalidad en cuanto a la posibilidad de ejecutar e interpretar diversos fragmentos de código escritos en distintos lenguajes, esto significa que si se codifica en C#, VisualBasic o J# todos estos lenguajes serán convertidos al lenguaje intermediario MSIL(”Microsoft Intermediate Language”) a través de un compilador determinado, este lenguaje intermediario lo ejecuta una máquina virtual(.Net Framework, solo disponible para Windows) de esta manera su código escrito

en más de 20 lenguajes se podrá interpretar. Mono trata de implementar esta máquina virtual, pero el estado de Mono es incompleto sobre todo en el paquete Windows Forms que no está completamente portado y es inestable.

En cuanto a Java, también utiliza el concepto de máquina virtual. El código que se genera no es específico a una plataforma en particular. Un programa nativo: la máquina virtual (VM) se encarga de traducir este código para que la máquina pueda ejecutarlo. De esta manera un código generado en Java puede correr en cualquier plataforma, en donde se haya portado la VM. Además de que tiene una API mucho más desarrollada que la de C# (el jdk 1.5 tiene más de 3000 clases), con Java se puede hacer prácticamente todo, de una manera muy simple y además existe mucha documentación disponible.

Una de las finalidades que se busca con el desarrollo del presente software es que sea portable, por lo que se ha decidido implementar el mismo en Java. Este lenguaje tiene otras características importantes que determinaron que se seleccionara para la implementación de nuestro software como son:

1. "Orientado a Objetos: Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las características propias del paradigma orientado a objetos: abstracción, encapsulamiento, herencia y polimorfismo.
2. Simple: Posee una curva de aprendizaje muy rápida. Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos.
3. Robusto: Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos en Java ayuda a detectar errores lo antes posible en el ciclo de desarrollo. Java obliga a la declaración explícita de los tipos de los ítems de información, reduciendo las posibilidades de error. Maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de la misma.
4. Arquitectura neutral, portátil y robusta: Es neutral, al adoptar un sistema de código binario que es independiente de arquitecturas hardware, sistemas operativos y sistemas de ventanas. Es portátil, al definir de forma precisa los tipos y tamaños de los datos. Y es robusta, al poseer un chequeo del código tanto en tiempo de compilación como de ejecución. Y la mayor diferencia con C y C++, el modelo de memoria de Java elimina la posibilidad de sobrescribirla y la corrupción de los datos.
5. Independiente de la plataforma: Java se compila a un formato de código de byte que puede ser leído e interpretado por muchas plataformas, incluyendo Windows 95, Windows NT, Solaris 2.3, GNU\Linux, Mac OS, etc.

6. Seguro: El código de Java puede ser ejecutado en un entorno que prohíbe la introducción de virus, borrar y modificar ficheros o la ejecución de operaciones que provoquen la caída del ordenador y la pérdida de datos.
7. Multihilo: En Java, todo transcurre de forma paralela, con varias tareas de forma simultánea. Un único programa Java puede procesar diferentes cosas de forma independiente y continua." [18]

1.4.5 Entorno de Desarrollo Integrado

Dentro de los entornos de desarrollo integrado (IDE del inglés Integrated Development Environment) para el desarrollo de aplicaciones usando como lenguaje de programación Java se pueden encontrar NetBeans, JBuilder y Eclipse.

Eclipse: Es un IDE para todo tipo de aplicaciones, inicialmente desarrollado por IBM, y actualmente gestionado por la Fundación Eclipse. La característica clave de Eclipse es la extensibilidad. Eclipse es una gran estructura formada por un núcleo y muchos plugins que van conformando la funcionalidad final. La forma en que los plugins interactúan es mediante interfaces o puntos de extensión; así, los nuevos aportes se integran sin dificultad ni conflictos. [19]

JBuilder: Es un entorno de desarrollo integrado para el lenguaje de programación Java desarrollado por Borland. Posee varias ediciones, la Enterprise, para aplicaciones J2EE, Web Services y struts, la Developer para el desarrollo completo de aplicaciones Java, y la Foundation con capacidades básicas para iniciarse en Java. No fue seleccionado para el desarrollo de la herramienta JBuilder por no ser multiplataforma, solo puede ser ejecutado sobre el sistema operativo Windows.

NetBeans: Permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de Java escritas para interactuar con las APIs de NetBeans y un archivo especial que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software.

El NetBeans es una herramienta para programadores pensada para escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Es una base modular y extensible usada como una estructura de integración para crear aplicaciones de escritorio grandes. [20]

Existe además un número importante de módulos para extender el IDE NetBeans, es un producto libre y gratuito sin restricciones de uso. Por estas razones es que fue escogido para el desarrollo de nuestro software.

1.4.6 Herramienta de ORM (Mapeo de Objetos Relacional)

El mapeo objeto-relacional es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos. En la práctica esto crea una base de datos orientada a objetos virtual, por sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo). Hay paquetes comerciales y de uso libre disponibles que desarrollan el mapeo relacional de objetos.

Existen diversas herramienta ORM, Hibernate es una de ellas para la plataforma Java de libre distribución bajo los términos de la LGPL (Licencia Pública General Menor de GNU) que soporta la mayoría de los sistemas de bases de datos SQL. Su característica principal es el mapeo de clases en Java a tablas de una base de datos (y de tipos de datos de Java hacia tipos de datos de SQL). Genera las sentencias SQL, libera al desarrollador del manejo manual de los datos que resultan de la ejecución de dichas sentencias y para lograr la persistencia de los objetos (almacenarlos y recuperarlos) hace uso de la API de Java JDBC.

Hibernate servirá como puente entre nuestro sistema y la base de datos y sus funciones irían desde la ejecución de sentencias SQL a través de JDBC hasta la creación, modificación y eliminación de objetos persistentes además soporta la implementación de clases de acceso a datos DAO (Data Acces Object) con las operaciones que se pueden hacer con ellas (insertar, borrar, salvar, etc) o también conocidas como interface DAO que son de suma importancia para nuestro sistema permitiendo el acceso a los datos. [21]

1.5 Roles y Artefactos

Teniendo en cuenta los roles que define la metodología OpenUp, se decidió que los roles que se desempeñarán son: analista y desarrollador y los artefactos serán los que define la metodología para estos roles.

1.5.1 Rol: Analista

La persona en esta función representa al cliente y usuario final se refiere a la recopilación de los aportes de los interesados para entender el problema a resolver, y por la captura y el establecimiento

de prioridades de las necesidades. [22]

Realiza:

1. Crear casos de prueba.
2. Diseño de la Solución.
3. Aplicar prueba.
4. Evaluar Resultados.
5. Administrar Iteración.

Modifica:

1. Documento Visión.
2. Glosario.
3. Casos de Uso.
4. Modelo de Casos de Uso.

Este rol necesita de los siguientes conocimientos, destrezas y habilidades:

1. Experiencia en la identificación y entendimiento de los problemas y oportunidades.
2. Capacidad de articular las necesidades que están asociados con el problema clave que hay que resolver o la oportunidad de realizar.
3. Buenas habilidades de comunicación, oral y escrita.
4. Conocimiento de la empresa y la tecnología de dominios o la capacidad de absorber y comprender rápidamente esa información.

Artefactos de trabajo:

1. Levantamiento de requerimientos.
2. Casos de Uso.
3. Modelo de Casos de Uso.

1.5.1.1 Artefactos:

Levantamiento de Requerimientos:

Este artefacto define el ámbito del sistema, provee a los desarrolladores de un mejor entendimiento de los requisitos del sistema y una base para estimar costos y tiempo de desarrollo del sistema. Define una interfaz de usuario enfocada a las necesidades y metas del usuario.

Casos de Uso:

Este artefacto captura la secuencia de acciones que realiza un sistema que produce un resultado observable de valor para que los que interactúen con el sistema.

El principal objetivo del caso de uso es que es necesario para capturar el comportamiento del sistema desde la perspectiva del usuario final, para lograr uno o más objetivos. Los diferentes usuarios se benefician de diversas maneras.

Modelo de Casos de Uso:

Su artefacto capta un modelo de las funciones del sistema y su entorno, y sirve como un contrato entre el cliente y los desarrolladores.
Este artefacto se presenta un panorama de la intención de comportamiento del sistema. Es la base de un acuerdo entre las partes interesadas y el equipo del proyecto en cuanto a la intención de la funcionalidad para el sistema. También ayuda a orientar las diversas tareas en el ciclo de vida de desarrollo de software.

1.5.2 Rol: Desarrollador.

Esta función se encarga de la elaboración de una parte del sistema, incluyendo el diseño que encaja en la arquitectura, posiblemente de prototipos de la interfaz de usuario y, a continuación, la aplicación, la unidad de pruebas, y la integración de los componentes que forman parte de la solución. [22]

Realiza:

1. Buscar y esbozar requerimientos.
2. Detallar requerimientos.
3. Crear casos de prueba.
4. Aplicar prueba.
5. Evaluar resultados.

Modifica:

1. Diseño.
2. Aplicación.
3. Build.
4. Casos de Uso.
5. Casos de prueba.

Este rol necesita de los siguientes conocimientos, destrezas y habilidades:

1. Definir y crear soluciones técnicas en el proyecto de tecnología.
2. Comprender y ajustarse a la arquitectura.
3. Identificar y construir pruebas que requieren cubrir el comportamiento de los componentes técnicos.
4. Comunicar el diseño en una forma que otros miembros del equipo puedan entender.

Una persona que desempeñe este papel puede tener conocimientos especializados en una determinada área técnica, pero también debe tener una amplia comprensión de todas las tecnologías que intervienen para poder trabajar con otros miembros del equipo técnico.

Artefactos de trabajo:

1. Diseño.
2. Build.
3. Aplicación.
4. Casos de prueba.

1.5.2.1 Artefactos:

Diseño:

En este artefacto se describe la realización de la funcionalidad del sistema en términos de componentes y sirve como una abstracción del código fuente. Tiene como propósito describir los elementos del sistema para que puedan ser examinados y comprendidos en formas que no eran posibles por la lectura del código fuente.

Es importante que el desarrollador pueda analizar las decisiones fundamentales sobre la estructura y comportamiento del sistema y comunicar a los demás colaboradores. También es importante que estas decisiones puedan ser comunicadas a los diversos niveles de abstracción y granularidad. Algunos aspectos del diseño pueden ser representados por el código fuente, posiblemente con algunas anotaciones adicionales.

Build:

Una versión operativa de un sistema o de una parte de un sistema que demuestra un subconjunto de las capacidades que se presente en el producto final. Esta versión de trabajo del sistema es el resultado de poner la aplicación del sistema a través de un proceso de construcción (en general, un sistema automatizado de script) que crea una versión ejecutable del sistema. Esta versión ejecutable del sistema suele tener un número de ficheros de soporte que también se consideran parte de este artefacto compuesto.

En un ciclo de vida iterativo, cada construcción debe pasar de la iteración anterior de la construcción, añadiendo más funcionalidad y mejorar la calidad. El propósito de la primera se basa en que minimizar o eliminar el riesgo o verificar decisiones es lograr una construcción consistentemente estable en iteraciones posteriores.

Este artefacto es casi siempre un producto compuesto formado por numerosas piezas necesarias para el sistema ejecutable. Por lo tanto, un build es algo más que archivos ejecutables, sino que, además, incluye cosas tales como archivos de configuración, archivos de ayuda, y los repositorios de datos que se pondrán juntos en el producto resultante. Las características específicas de las partes pueden variar según la tecnología en uso.

Aplicación:

Software compuesto por los archivos de código, los archivos de datos, y el respaldo de los archivos, como los archivos de ayuda en línea que representan partes de un sistema que se puede construir.

Tiene como propósito representar las partes físicas que componen el sistema de construcción, organizada de una manera que sea comprensible y manejable.

Este artefacto es la colección de uno o más de estos elementos:

1. El código fuente de archivos.
2. Los archivos de datos.
3. Creación de guiones.
4. El resto de los archivos que se transforman en el sistema ejecutable.

Casos de pruebas:

Contiene las instrucciones que validan los distintos componentes de software a realizar.

Este artefacto cubre todos los pasos que son necesarios para validar un componente de software. Especifica entradas de prueba, las condiciones de ejecución, y los resultados esperados. Estos datos son identificados en el marco de la evaluación de un aspecto particular de un escenario.

Los ensayos deberán documentar de una forma clara al término de la prueba si el componente se ha ejecutado correctamente.

1.6 Patrones de arquitectura y patrones de diseño

1.6.1 Concepto de patrón

Según Christopher Alexander:

"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo siquiera dos veces de la misma forma".[23]

1.6.2 Patrones de arquitectura

Los patrones de arquitectura describen un problema particular y recurrente de diseño, que aparece en

contextos de diseño específico, y presenta un esquema genérico demostrado con éxito para su solución.

Algunos patrones de arquitectura son:

1. Capas
2. Tubería-filtros
3. Pizarra
4. Bróker
5. Modelo-Vista-Controlador
6. Presentación-Abstracción-Control
7. Reflexión
8. Microkernel

De acuerdo con la definición y la función que realizan estos patrones se ha decido utilizar en el diseño del módulo el siguiente:

Modelo-Vista-Controlador (MVC): En ocasiones se le define más bien como un patrón de diseño o como práctica recurrente, y en estos términos es referido en el marco de la estrategia arquitectónica de Microsoft.

El patrón MVC separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres clases diferentes:

- Modelo: El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).
- Vista: Maneja la visualización de la información.
- Controlador: Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado. [23]

1.6.3 Patrones de Diseño

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular (Buschmann, 1996).

Son menores en escala que los patrones arquitectónicos, y tienden a ser independientes de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero sí sobre la de un subsistema, debido a que especifica a un mayor nivel de detalle,

sin llegar a la implementación, el comportamiento de los componentes del subsistema. [24]

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

Podemos clasificar a estos patrones según su propósito: [25]

1. **Patrones de creación:** para creación de instancias.
2. **Patrones estructurales:** relaciones entre clases, combinación y formación de estructuras mayores.
3. **Patrones de comportamiento:** interacción y cooperación entre clases.

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

Tabla 1.2 Clasificación Patrones de Diseño.

De acuerdo con la definición y la función que realizan estos patrones hemos decido utilizar en el diseño del módulo los siguientes:

1. **Fachada:** Proporciona una interfaz unificada para un conjunto de interfaces de un subsistema. Define una interfaz de alto nivel que hace que el subsistema sea más fácil de usar.
2. **Estratégico:** Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.
3. **Patrón Observador:** Para el mecanismo de publicación y suscripción que permite la notificación de los cambios en el modelo a las vistas. Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambia de estado se notifica y actualizan automáticamente todos los objetos.

1.6.3.2 Patrones de diseño GRASP

Lo patrones de GRASP, no compiten con los patrones de diseño, los patrones de GRASP, son una guía para ayudarnos a encontrar los patrones de diseño (que son más concretos). Los patrones GRASP son parejas de problema solución con un nombre, que codifican buenos principios y sugerencias relacionados frecuentemente con la asignación de responsabilidades. Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos,

expresados en forma de patrones.

Se pueden destacar cinco patrones principales que debido a la importancia que tienen hemos decido utilizar: [26]

1. **Experto:** Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele ser útil en el diseño orientado a objetos.
   - El cumplimiento de una responsabilidad requiere a menudo información distribuida en varias clases de objetos. El patrón Experto asigna responsabilidades a las clases que tienen la información necesaria para cumplir con la responsabilidad.
2. **Creador:** Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. Lo que define este patrón es que una instancia de un objeto la tiene que crear el objeto que tiene la información para ello.
3. **Alta cohesión:** Mantiene la complejidad dentro de limites manejables, es decir asigna una responsabilidad de modo que la cohesión siga siendo alta.
   La cohesión es una medida de cuan relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.
4. **Controlador:** es un patrón que sirve como intermediario entre una determinada interfaz y el algoritmo que la implementa, de tal forma que es la que recibe los datos del usuario y la que los envía a las distintas clases según el método llamado. Asigna las responsabilidades de capturar los eventos del sistema a las clases.

1.7 Conclusiones del capítulo

Para el desarrollo del sistema se utiliza la metodología OpenUp debido a que es más factible en este caso utilizar solo los procesos más importantes para el sistema, no utilizar demasiados artefactos y sobre todo que el proyecto se acople a las necesidades del usuario pudiendo ser este modificado, mejorado y extendido. Como roles que define esta metodología desempeñaremos rol de analista y desarrollador.

Como lenguaje de modelado se utiliza UML y como herramienta Case, Visual Paradigm por las ventajas que proporciona esta herramienta, fundamentalmente porque es multiplataforma.

El lenguaje utilizado es Java por ser multiplataforma y la herramienta que se consideró más adecuada para la implementación es NetBeans por la rapidez de compilado y su fácil trabajo con ventanas y como herramienta de mapeo relacional de objetos, Hibernate.

Por el nivel de cálculo que se requiere se utiliza computación Grid y cálculo distribuido mediante t-arenal, para agilizar el proceso de cálculo y reducir el tiempo de simulación.

Para la resolución del modelo matemático se hace uso del asistente matemático MatLab y de la librería ODEtoJava debido a los métodos de resolución de SED que tienen implementados.

Se utilizó como patrón de arquitectura Modelo-Vista-Controlador, así como patrones de diseño GoF (Decorador, Fachada, Puente, Adaptador, Estratégico) y GRASP (Alta cohesión, Bajo Acoplamiento, Creador, Experto y Controlador).

Capítulo 2: Características del Sistema

Introducción

En este capítulo se hará una descripción del sistema a automatizar y se mostrará un diagrama de casos de uso del sistema para un mejor entendimiento del mismo. Se definirán los actores del sistema y los requerimientos funcionales del sistema que se implementó. Se plantearán los requerimientos mínimos que debe tener el sistema de cómputo donde se vaya a instalar la aplicación.

2.1 Actores del Sistema

Los actores de un sistema pueden ser otros sistemas o hardware externo que se relacionan o interactúan con dicho sistema, no necesariamente tiene que ser una persona. Cada actor juega un rol determinado al interactuar con el sistema y diferentes usuarios pueden asumir el mismo rol de un actor. Luego de definir el concepto de actores del sistema se puede establecer el o los actores del sistema en cuestión. [27]

Nombre del Actor:	**Descripción:**
Investigador	El investigador es aquella persona que va a interactuar con el sistema para realizar las simulaciones.

Tabla 2.1 Actores del Sistema

2.2 Requisitos no Funcionales

Restricciones que afectan a los servicios o funciones del sistema, tales como restricciones de tiempo, sobre el proceso de desarrollo y estándares. Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Estas propiedades son las que hacen al producto atractivo, usable, rápido o confiable. [27]

Requisitos no funcionales:

1. Apariencia o interfaz externa
    - La aplicación deberá tener una interfaz externa amigable, que sea sencilla y fácil de

entender por el usuario para así evitar que el usuario se pierda dentro de la aplicación.

2. Software:
    - No hay restricciones en cuanto al sistema operativo a instalar en la PC que esté instalada la aplicación, puesto que la misma será multiplataforma.
    - La máquina virtual de Java 1.5 tiene que estar instalada.
    - En cada una de las máquinas clientes que se utilicen para realizar las simulaciones distribuidas deberán tener instalado cualquier distribución de Linux o Windows como sistema operativo.
    - El asistente matemático MatLab tiene que estar instalado en cada una de las máquinas que se vaya a utilizar como clientes de la plataforma distribuida.
    - El cliente de la Grid deberá estar instalado en cada una de las PCs que se utilizarán para la simulación distribuida.
3. Hardware:
    - Se debe contar con 256 MB de memoria RAM como mínimo, aunque lo ideal sería 512 MB.
    - Procesadores Pentium IV.
    - Disco duro de 20 GB como mínimo (depende de la cantidad de información a almacenar).
4. Restricciones del diseño y la implementación:
    - Lenguaje de programación Java.
    - NetBeans como IDE de desarrollo.
    - Visual Paradigm como herramienta CASE.
    - PostgreSQL como gestor de bases de datos.
5. Seguridad
    - Confidencialidad: Se requiere de usuario y contraseña para poder acceder a la información.
    - Se requiere de usuario y contraseña para poder acceder al servidor de la Grid.
    - Disponibilidad: En caso de tener el usuario y la contraseña se le garantiza poder acceder a la información almacenada en todo momento.
6. Usabilidad:
    - El sistema le ofrecerá al investigador la posibilidad de realizar simulaciones distribuidas o locales, en este sentido se centra el diseño de la aplicación y en específico de las interfaces que harán posible el intercambio de datos de manera que le resulte al usuario

de fácil entendimiento.

7. Rendimiento
    - Para hacer más rápida la obtención de los resultados de las simulaciones se implementará una aplicación que haga uso del cálculo distribuido de las mismas para agilizar el proceso de obtención de dichos resultados.
8. Soporte:
    - El sistema debe permitir la interacción con los demás módulos que componen la plataforma.
    - Se realizarán distintas pruebas al software una vez concluido para comprobar su funcionalidad.
    - Terminado el software se prestarán los servicios de instalación y configuración de la aplicación.
    - Se prestarán servicios de mantenimiento del software.
9. Portabilidad:
    - El sistema deberá funcionar en cualquier sistema operativo sobre el cual se haya instalado la máquina virtual de Java 1.5 o superior.
10. Confiabilidad:
    - El sistema no debe permitir que existan fallos, pero de ocurrir se debe garantizar que la pérdida de información sea mínima.
    - De ocurrir algún fallo en las conexiones para realizar los cálculos distribuidos, el sistema está diseñado para detectar este error y enviar la parte del trabajo que no se concluyó a otra máquina que esté disponible para que esta la continúe.

2.3 Requisitos Funcionales

Declaración de los servicios que el sistema debe proporcionar. Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir. [27]

Requisitos Funcionales:

R1. Editar Preferencias.

R2. Cargar el modelo matemático.

R3. Definir valores necesarios para realizar simulaciones.

R4. Gestionar Simulación.

R4.1. Realizar la Simulación.

R4.2. Chequear el estado de la simulación.

R4.3. Detener o parar los procesos de simulación.

2.4 Diagrama de Casos de Uso del Sistema

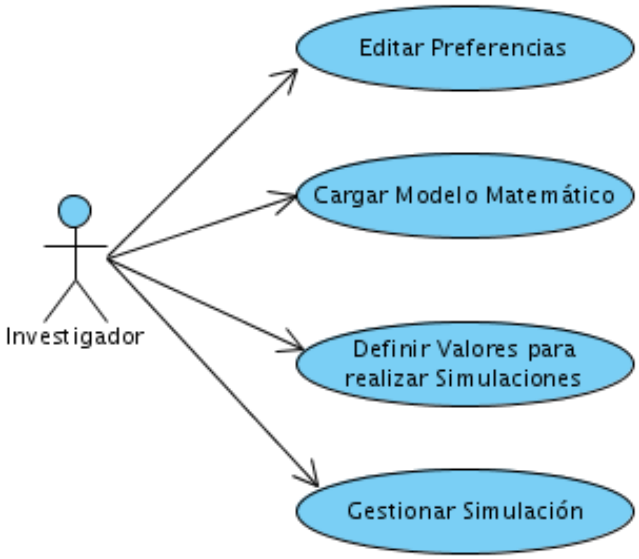


Figura 2.1 Diagrama de CUS

2.5 Casos de Uso del Sistema

1. Editar Preferencias
2. Cargar el modelo matemático.
3. Definir valores necesarios para realizar simulaciones.
4. Gestionar Simulación.

2.6 Descripción de los casos de uso del sistema

Descripción del Caso de Uso Editar Preferencias.

En este caso de uso el investigador puede editar si lo desea las preferencias del sistema como: Seleccionar la herramienta matemática (MatLab u ODEtoJava), si es MatLab se debe seleccionar comando para ejecutarlo y el puerto socket a utilizar, si el investigador no desea introducir alguno de estos datos el sistema puede generarlos automáticamente. El investigador también puede editar las preferencias de conexión al servidor Grid (puerto RMI, puerto Socket, IP) y de los clientes que se usarán (IP, comando para ejecutar MatLab, puerto Socket).

<table>
<tr><td>Nombre del Caso de Uso</td><td colspan="2">Editar Preferencias</td></tr>
<tr><td>Actores</td><td colspan="2">Investigador</td></tr>
<tr><td>Propósito</td><td colspan="2">Definir opciones que brinda el sistema para realizar la simulación.</td></tr>
<tr><td>Resumen</td><td colspan="2">El Investigador debe decidir los términos en que se realizará la simulación: herramienta matemática a utilizar, máquinas que participarán en el proceso de simulado, configuración local de la conexión y servidor Grid a utilizar.</td></tr>
<tr><td>Referencias</td><td colspan="2">R1</td></tr>
<tr><td>Precondiciones</td><td colspan="2">El investigador tiene que estar autenticado en el sistema.</td></tr>
<tr><td>Poscondiciones</td><td colspan="2">Preferencias guardadas.</td></tr>
<tr><td colspan="3">Curso normal de los eventos</td></tr>
<tr><td colspan="2">Acción del actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1. El Investigador selecciona la opción Preferencias.

3. El investigador puede:
Editar Preferencias.
● Seleccionar herramienta matemática:
○ MatLab:
▪ Seleccionar comando para ejecutar MatLab.
▪ Seleccionar el puerto socket a utilizar.
○ Java.
● Seleccionar el fichero donde se guardarán las preferencias.
Editar Grid.</td><td>2. El sistema muestra la interfaz correspondiente.

4. El sistema guarda los datos.</td></tr>
</table>

● Introducir el IP del servidor Grid. ● Introducir las PC's cliente a utilizar. ▪ Ip Cliente. ▪ Comando MatLab. ▪ Puerto Socket. ● Introducir el puerto RMI a utilizar. ● Introducir el puerto socket a utilizar.	
Prioridad	Opcional

Descripción del Caso de Uso Cargar el Modelo Matemático.

Este caso de uso permite al investigador cargar y mostrar el modelo matemático con el que desea trabajar.

Nombre del Caso de Uso	**Cargar el Modelo Matemático**
Actores	Investigador
Propósito	Seleccionar el modelo matemático a resolver para obtener resultados en las simulaciones.
Resumen	El investigador selecciona el modelo matemático sobre el cual se realizaran las simulaciones posteriores.
Referencias	R2
Precondiciones	Debe existir modelo matemático creado con anterioridad y el investigador tiene que estar autenticada en el sistema.
Poscondiciones	Sistema listo para realizar simulaciones.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**

2. El Investigador selecciona modelo matemático a utilizar.	1. El sistema muestra los modelos matemáticos existentes. 3. El sistema muestra los datos del modelo matemático seleccionado.
Prioridad	Crítico

Descripción del Caso de Uso Definir Valores Necesarios para Realizar Simulaciones.

El investigador define los valores de las condiciones iniciales (variables, parámetros, tiempos de integración, tolerancia relativa y absoluta), igualmente se define la forma en que se realizará la simulación (local o distribuida) y el método numérico.

Nombre del Caso de Uso	**Definir valores necesarios para realizar simulaciones.**
Actores	Investigador
Propósito	Definir valores necesarios para realizar simulaciones.
Resumen	El investigador define los valores de: condiciones iniciales, parámetros, tiempo de integración, si la simulación será local o distribuida.
Referencias	R3
Precondiciones	Debe haberse cargado un modelo matemático con anterioridad.
Poscondiciones	Sistema listo para comenzar a simular.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**
1. El Investigador define los siguientes valores: • Variables. • Parámetros. • Tiempos. de integración. • Tolerancia Relativa. • Tolerancia Absoluta. • Método Numérico.	

2. El investigador define si la simulación será local o distribuida.	3. El sistema verifica la validez de los datos. 4. El sistema guarda los valores definidos.
Curso alterno de los eventos	
Acciones del actor	**Acciones del sistema**
	3. El sistema verifica la validez de los datos. Los datos no son válidos. 4. El sistema muestra un mensaje “Introduzca valores válidos para las variables.”
Prioridad	Crítico

Descripción del Caso de Uso Gestionar Simulación.

Este caso de uso permite al investigador gestionar las simulaciones (comenzar, chequear, detener).

Nombre del Caso de Uso	**Gestionar Simulación.**
Actores	Investigador
Propósito	Resolver un modelo matemático que describa a un sistema biológico.
Resumen	El investigador manda a simular el sistema biológico, chequea el estado de la simulación, decide si detenerla o proseguir con la misma y por ultimo almacena los resultados obtenidos.
Referencias	R5, R5.1, R5.2,R5.3
Precondiciones	El investigador se debe haber autenticado en el sistema, introducido los valores de las condiciones iniciales y cargado el modelo matemático.
Poscondiciones	Se almacena en la Base de Datos los resultados de la simulación.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**

1. El investigador accede a la opción “Simulación”. El investigador podrá: • Realizar simulación. • Chequear simulación. • Detener simulación.	2. El sistema realizará las acciones necesarias para: Si "Comenzar simulación” ir a Sección Realizar simulación. Si "Chequear simulación” ir a Sección Chequear simulación. Si "Detener simulación” ir a Sección Detener simulación.
Sección “Realizar Simulación”.	
1. El investigador selecciona la opción Ejecutar Simulación.	2. El sistema verifica si la simulación se realizará de forma local o distribuida. Se realiza localmente. 3. El sistema comienza a simular el modelo matemático cargado. 4. El sistema almacena los resultados de las simulaciones en la Base de Datos.
Escenario “Chequear Simulación”.	
1. El Investigador selecciona la opción Ejecuciones.	2. El sistema verifica si se esta corriendo en ese momento alguna simulación. Existe una simulación ejecutándose. 3. El sistema muestra la barra de progreso que representa el estado de la simulación en ese momento. 4. El sistema muestra un reporte con las simulaciones realizadas, en espera y las que fallaron.
Escenario “Detener Simulación”.	
1. El Investigador selecciona la opción detener simulación.	2. El sistema verifica si se esta corriendo en ese momento alguna simulación. Existe una simulación

	ejecutándose. 3. El sistema detiene todos los procesos de simulación que se encuentren en ejecución.
Curso Alterno de Eventos	
Sección "Realizar Simulación".	
Acción del actor	**Respuesta del Sistema**
	2. El sistema verifica si la simulación se realizará de forma local o distribuida. Se realiza de forma distribuida. 3. El sistema busca respuesta del servidor. El servidor responde. 4. El sistema se autentica dentro del servidor Grid con el usuario y la contraseña definidos en las preferencias. Autenticación exitosa. 5. El sistema comienza a simular el modelo matemático cargado. 6. El sistema almacena los resultados de las simulaciones en la Base de Datos.
Acción del actor	**Respuesta del Sistema**
5. El investigador va a la Sección "Grid" del caso de uso Editar Preferencias, e introduce un nuevo IP para el servidor Grid.	3. El sistema busca respuesta del servidor. El servidor no responde. 4. El sistema muestra un mensaje:"El servidor especificado no responde."
Acción del actor	**Respuesta del Sistema**
	4. El sistema se autentica dentro del servidor Grid con el usuario y la contraseña definidos en las preferencias.

6. El investigador va a la Sección “Grid” del caso de uso Editar Preferencias, e introduce un nuevo IP para el servidor Grid.	Autenticación fallida. 5. El sistema muestra un mensaje:”Usuario o contraseña incorrecto”.
Escenario “Chequear Simulación”.	
Acción del actor	**Respuesta del Sistema**
	2. El sistema verifica si se esta corriendo en ese momento alguna simulación. No existe una simulación ejecutándose. 3. El sistema muestra un mensaje:”No hay ninguna simulación en ejecución”.
Escenario “Detener Simulación”.	
Acción del actor	**Respuesta del Sistema**
	2. El sistema verifica si se esta corriendo en ese momento alguna simulación. No existe una simulación ejecutándose. 3. El sistema muestra un mensaje:”No hay ninguna simulación en ejecución”.
Prioridad	Crítico

2.7 Conclusiones del capítulo

Se evidenció la necesidad de implementar una herramienta que facilite la ejecución del proceso de simulación para lo cual se obtuvieron las actividades a automatizar y el flujo de información en cada una de las actividades descritas. Se definió el actor que interactúa con el sistema. Se tuvieron en cuenta además, las funcionalidades que debe tener el software y las cualidades del mismo planteadas como requisitos funcionales y requisitos no funcionales respectivamente. Se hizo una descripción de alto nivel de cada uno de los casos de uso contemplados en el diagrama de casos de uso del sistema.

Capítulo 3: Análisis y Diseño

Introducción

En este capítulo se describen los patrones de diseño y de arquitectura que fueron utilizados durante el desarrollo de la aplicación y se evidencia su utilización. Se realiza el diagrama de clases del diseño para cada subsistema o paquete definido dando respuesta a la solución que se propone y se define el diagrama de despliegue con el objetivo de mostrar la distribución física de los nodos de cómputo que necesita la aplicación.

3.1 Descripción de patrones arquitectónicos y de diseño

3.1.1 Patrones Arquitectónicos

El patrón arquitectónico utilizado es:

MVC: La aplicación separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres tipos de clases diferentes:

Vista: Las clases de esta capa se encuentra agrupadas en el paquete del diseño Presentación, el cual cuenta con las clases: BioSyS, VisualSimulacion, VisualPreferencia, VisualClientes, VisualProgreso, encargadas de visualizar la información y de la interacción con los diferentes tipos de usuarios. Estos aspectos incluyen el manejo y aspecto de las ventanas, el formato de los reportes y menues.

Controlador: Consta de la clases SimulacionControl, PreferenciasControl, BioSySControl, encargadas de gestionar el procesamiento de peticiones de las clases interfaces y BSSimulacion encargada de gestionar el procesamiento de las simulaciones y la muestra de respuestas del sistema.

Modelo: Agrupa a las clases BSMatLab, BSEngine, BSCondicion, BSParametro, BSVariable, BSClassLoader, BSPreferencias, BSPC encargadas de gestionar la información necesaria para el sistema y almacenar los datos persistentes.

3.1.2 Patrones de Diseño

Patrones Gof.

Fachada: Proporciona una interfaz unificada para un conjunto de interfaces en este caso BioSyS se encarga de facilitar el acceso a otras interfaces como son VisualSimulacion y VisualPreferencia.

Estratégico: Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución en este caso tenemos que la clase BSMatLab cuenta con cuatro constructores en los cuales varían los parámetros y se debe escoger en tiempo de ejecución cual de ellos utilizar, ya que los mismos están implementados para generar la información que no haya sido especificada por

el cliente, de igual forma tenemos la clase BSSimulacion que tiene tres métodos Simular con parámetros distintos.

Patrón Observador: Aplicamos este patrón para mostrar el estado del progreso de la simulación. Cuando cambia el estado del progreso de la simulación se actualiza automáticamente en la interfaz VisualProgreso.

Patrones Grasp:

Experto: Al encontrarse la información encapsulada en clases, cada una es responsable de manejar el acceso a esa información.

Creador: Para acceder a la información encapsulada en cada clase es necesario crear objetos de las mismas, y la encargada de crear dichos objetos es la propia clase.

Alta cohesión: La información ha sido distribuida de forma que cada clase maneje un volumen información aceptable.

Controlador: Siguiendo este patrón se implementaron las clases SimulacionControl. PreferenciasControl y BioSySControl que se encargan de la gestión de información entre las clases interfaces y las clases que las implementan.

3.2 Diagramas de Clases del diseño

Para modelar el diagrama de clases del diseño se definieron dos paquetes y un subsistema de diseño así como la comunicación que existe entre ellos. Es importante conocer que un paquete es un mecanismo de organización de elementos que subdividen el modelo de diseño en otros más pequeños que colaboran entre sí y un subsistema es un sistema cuya operación es independiente de los servicios provistos por otros subsistemas. A continuación presentamos los paquetes y el subsistema definidos:

Paquete Presentacion: Este paquete reúne las clases encargadas de la configuración necesaria para comenzar las simulaciones. En el mismo se configuran los datos del servidor y de los clientes, los datos referentes a las simulaciones y se carga el modelo matemático de la Base de Datos.

Paquete Simulacion: Este paquete se encarga de la gestión de simulaciones, dado un modelo matemático lo resuelve devolviendo una lista de soluciones. Es el encargado de gestionar los valores de las variables y los parámetros necesarios para simular, permite realizar las operaciones de forma local o distribuida, así como chequear el progreso de los cálculos.

Subsistema t-arenal: Subsistema encargado de gestionar las peticiones solicitadas por el paquete

Simulacion, para ella utiliza una clase *BSDataManager* que es la encargada de fragmentar el trabajo y enviárselo junto a la clase *BSAlgorithm* que lo procesa en las máquinas clientes de la Grid.

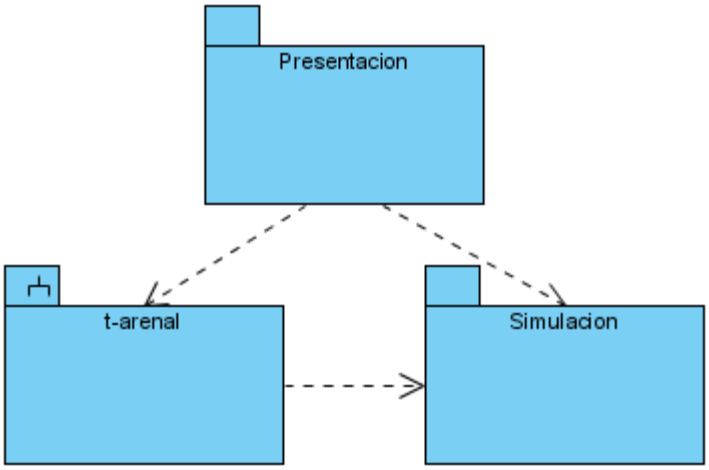


Figura 3.1 Diagrama de Paquetes.

3.2.1 Paquete de Presentación

A partir de la interfaz principal *BioSyS* el usuario puede acceder a *VisualPreferencia* antes de comenzar a simular para configurar las preferencias de la simulación, estas preferencias serán almacenadas mediante la clase *BSPreferencia* en un fichero que contiene la herramienta matemática que va a utilizar, el método numérico, los datos para configurar el servidor y además una lista de datos de los clientes. De no configurar las preferencias el sistema utilizará las preferencias por defecto.

El investigador puede acceder también a la interfaz *VisualSimulacion* donde define los valores de las variables y los parámetros, el tiempo de integración inicial y final y decide si la simulación se realizará de forma local o distribuida. De realizarse de forma local la clase *SimulacionControl* utiliza el paquete *Simulacion* para resolver el modelo matemático. Si se realiza de forma distribuida la clase *SimulacionControl* utiliza el *subsistema t-arenal* para fragmentar y repartir el trabajo entre las máquinas clientes quien luego simula utilizando el paquete *Simulacion*.

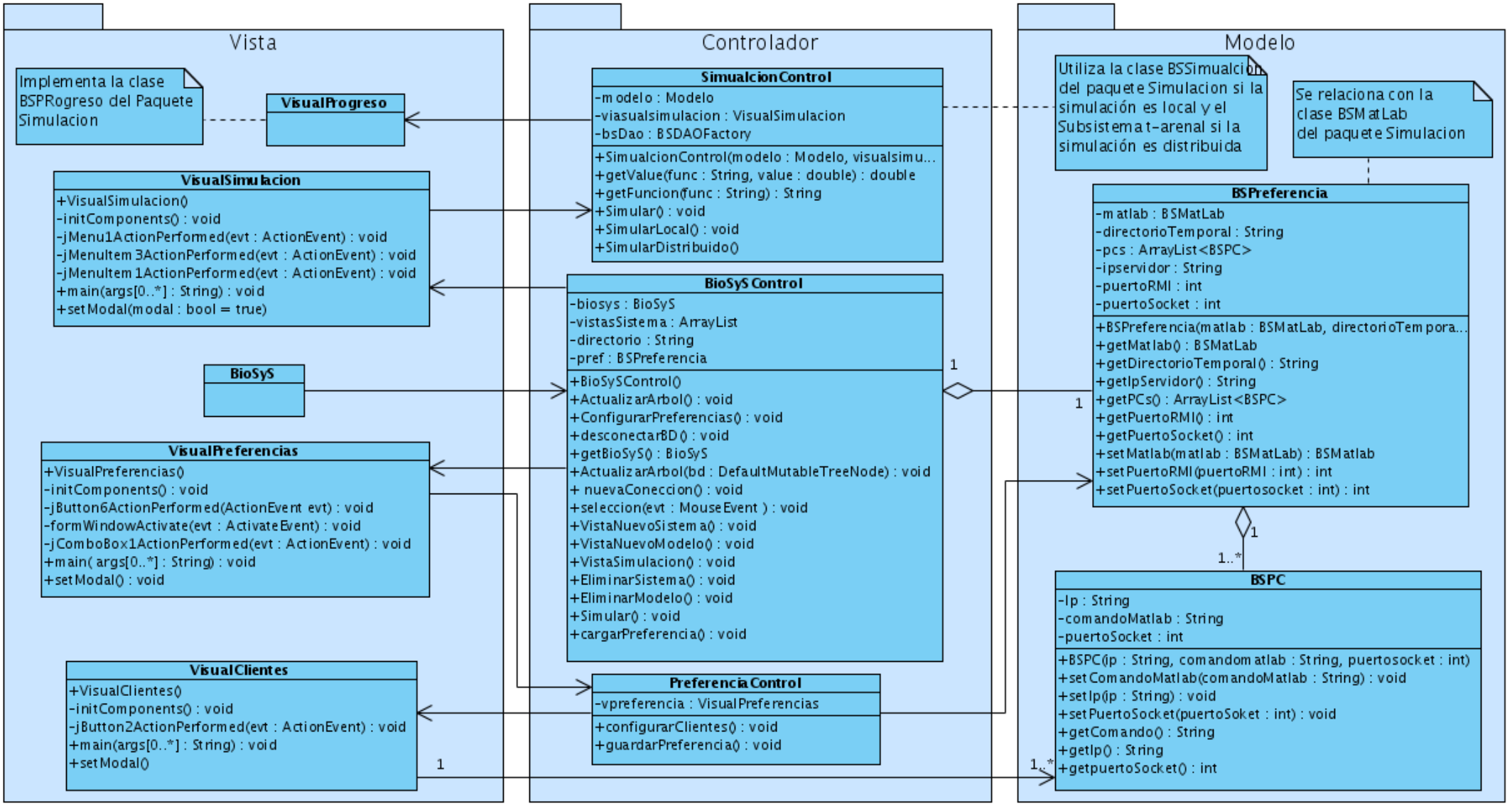


Figura 3.2 Diagrama de clases del diseño. *Paquete Presentacion*.

3.2.2 Paquete de Simulación

La clase principal de este paquete es *BSSimulacion*, esta clase contiene tres métodos para simular, el primero simula con MatLab y necesita un objeto de esta clase que contenga el comando para poder ejecutarlo, también tiene como parámetro el modelo matemático que se va a resolver, una lista de condiciones que contiene los valores de las variables y los parámetros que utiliza para simular, los valores de los tiempos de integración y las tolerancias relativa y absoluta. Este método realiza simulaciones locales y distribuidas, devuelve una lista de resultados que luego se almacenan en la base de datos.

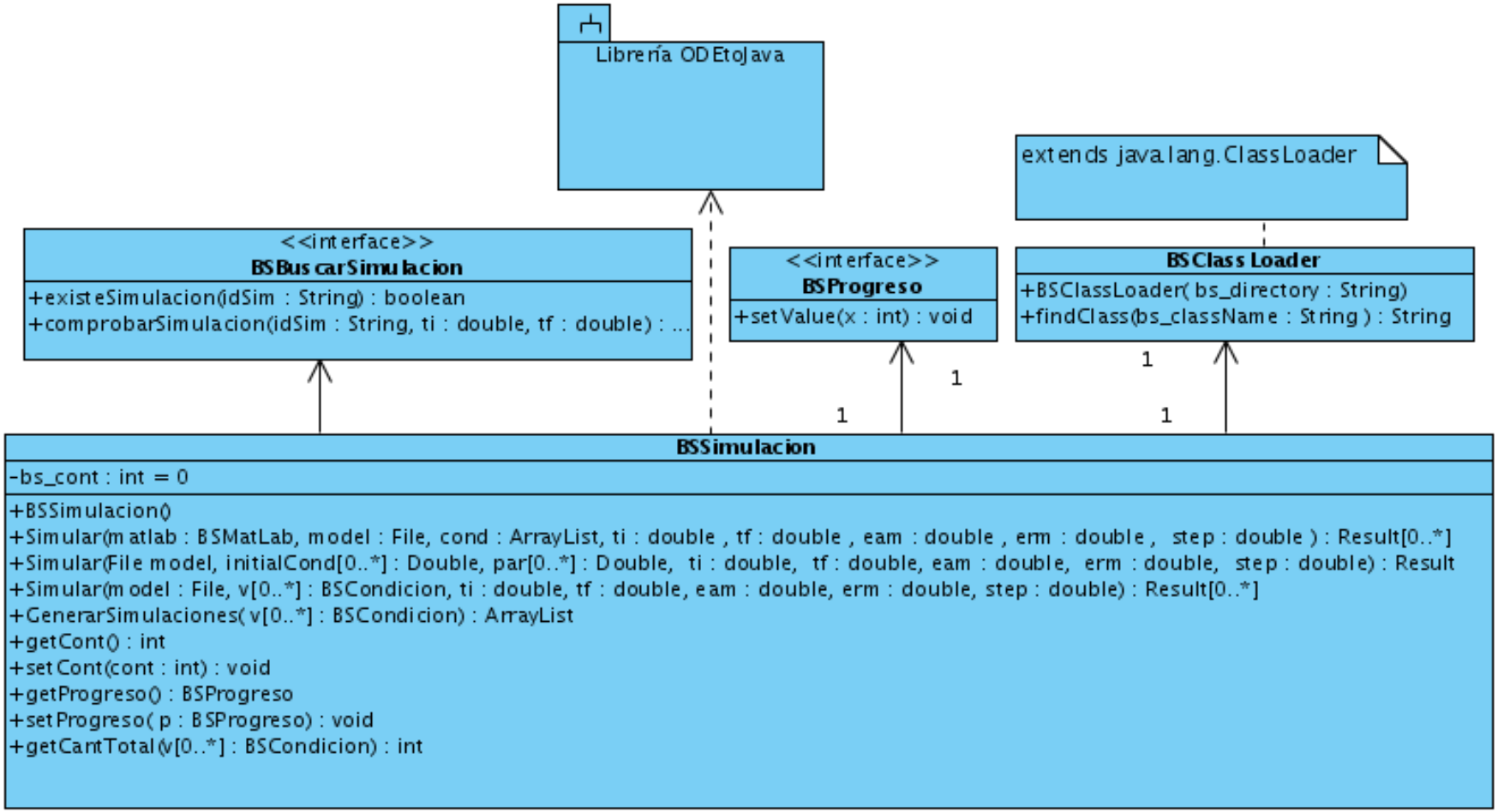
El segundo método que simula tiene como parámetro el modelo matemático, la lista de condiciones y los tiempos de integración pero utiliza la librería *ODEtoJava* para simular y solo realiza una simulación por lo tanto devuelve un solo resultado.

El tercer método es para simular con la librería *ODEtoJava* y tiene los mismos parámetros que el anterior pero realiza múltiples simulaciones utilizando el método anterior por lo que devuelve una lista de resultados.

Esta clase utiliza la librería *ODEtoJava* para resolver los modelos matemáticos cuando sea seleccionada en las preferencias, tanto para resolver simulaciones locales como distribuidas.

La clase *BSCondicion* contiene los datos de las variables y los parámetros que serán utilizadas para simular, *BSMatLab* contiene el comando para ejecutar MatLab, el método numérico y el puerto socket, la clase *BSClassLoader* se utiliza para cargar el modelo matemático de forma dinámica cuando se simula con la librería *ODEtoJava* y la clase *BSResultado* se utiliza cuando se terminan las simulaciones ya que contiene la información referente a los resultados de las mismas.

Para un mejor entendimiento del diagrama fue separado en las dos figuras siguientes (las clases representadas en estas figuras no se relacionan):


Librería ODEtoJava
extends java.lang.ClassLoader
<<interface>>
BSBuscarSimulacion
+existeSimulacion(idSim : String) : boolean
+comprobarSimulacion(idSim : String, ti : double, tf : double) : ...
<<interface>>
BSProgreso
+setValue(x : int) : void
BSClass Loader
+BSClassLoader(bs_directory : String)
+findClass(bs_className : String) : String
1
1
1
1
BSSimulacion
-bs_cont : int = 0
+BSSimulacion()
+Simular(matlab : BSMatLab, model : File, cond : ArrayList, ti : double , tf : double , eam : double , erm : double , step : double) : Result[0..*]
+Simular(File model, initialCond[0..*] : Double, par[0..*] : Double, ti : double, tf : double, eam : double, erm : double, step : double) : Result
+Simular(model : File, v[0..*] : BSCondicion, ti : double, tf : double, eam : double, erm : double, step : double) : Result[0..*]
+GenerarSimulaciones(v[0..*] : BSCondicion) : ArrayList
+getCont() : int
+setCont(cont : int) : void
+getProgreso() : BSProgreso
+setProgreso(p : BSProgreso) : void
+getCantTotal(v[0..*] : BSCondicion) : int

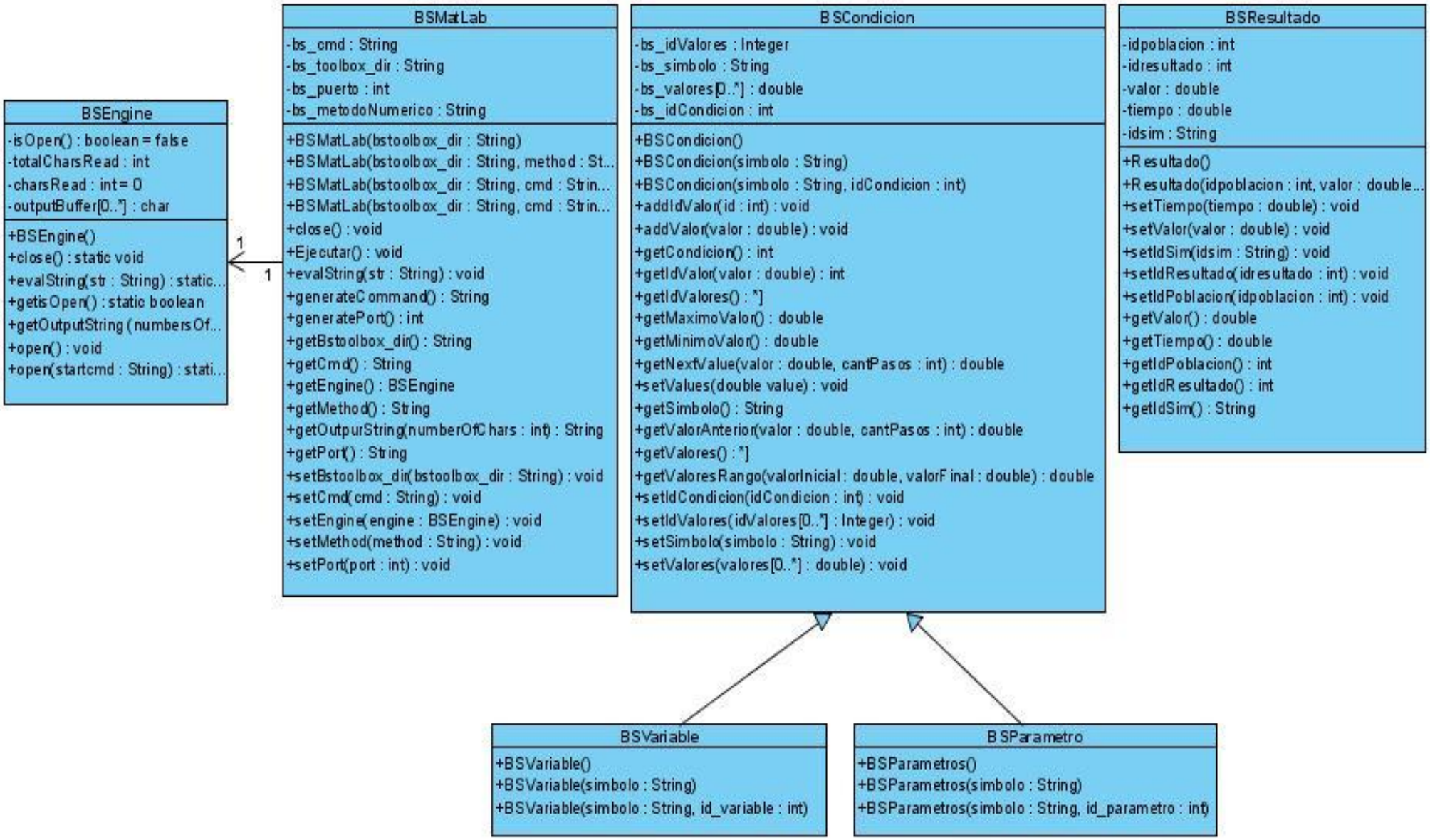


Figura 3.3. Diagrama de clases del diseño. *Paquete Simulacion.*

3.2.3 Subsistema t-arenal

La clase *BSDataManager* recibe el trabajo y se encarga de fragmentarlo, la clase *BSAlgorithm* se encarga de repartir el trabajo fragmentado en las máquinas clientes y de mandar a simular utilizando el paquete *Simulacion*. Las clases *BSDataManager* y *BSAlgorithm* son especificaciones de las clases *DataManager* y *Algorithm* de la Grid. Uno de los métodos más importantes de estas clases es *generateWorkUnits()* de *BSDataManager* este método recibe la información (número ip y sistema operativo) de los clientes que se usarán para simular y es el encargado de repartir el trabajo entre estos clientes atendiendo a las peticiones que realicen. El otro método fundamental es *proccesUnit()* de la clase *BSAlgorithm* que se encarga de procesar las unidades de trabajo en cada cliente utilizando el paquete *Simulacion* y la herramienta matemática seleccionada en cada caso. Debido a la importancia de estos dos métodos serán explicados detalladamente en el siguiente capítulo.

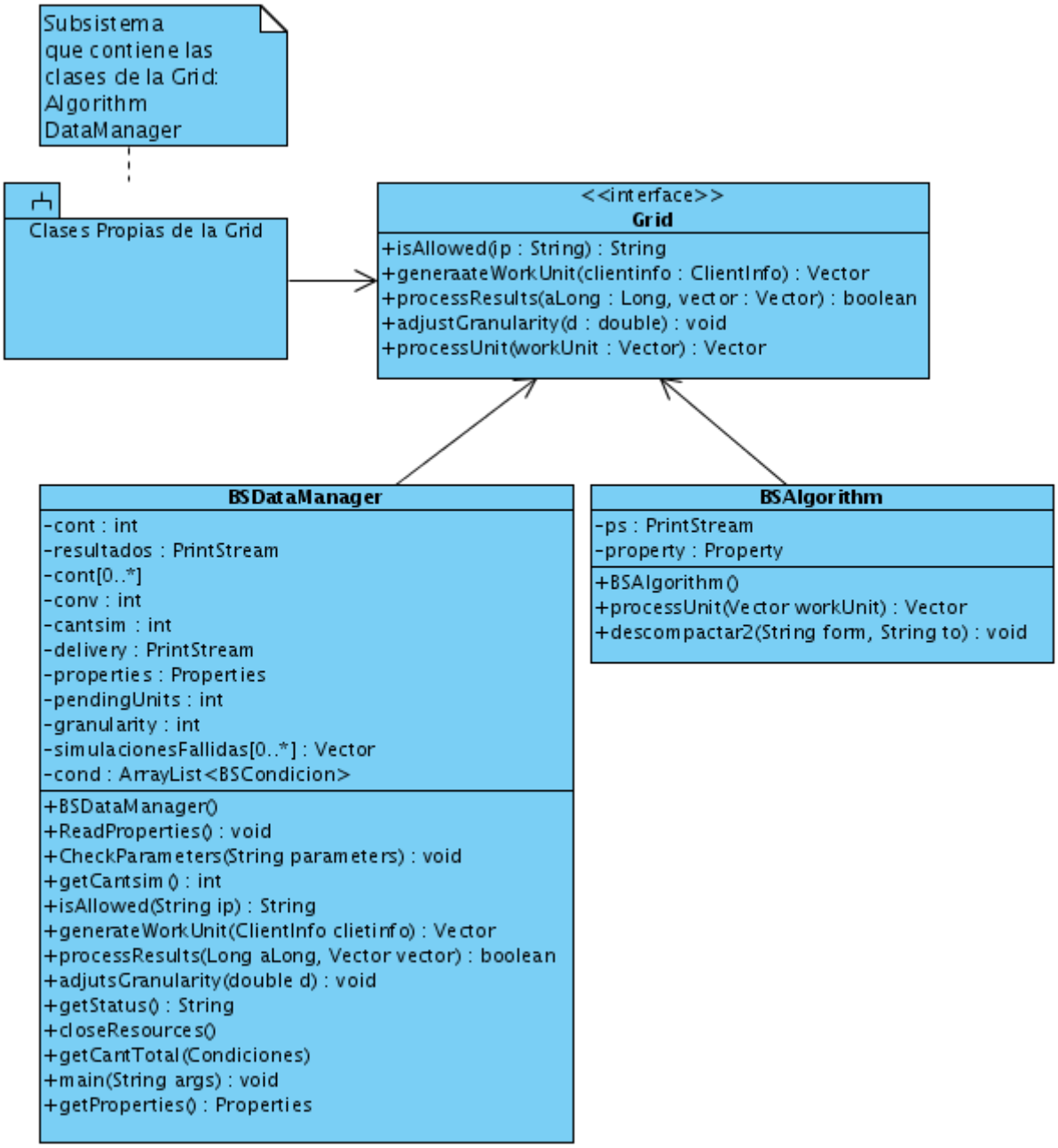


Figura 3.4 Diagrama de clases del diseño. *Subsistema t-arenal.*

3.3 Diagramas de Interacción

Guardar Preferencias: La clase *BioSySControl* muestra la interfaz *VisualPreferencias* a través del método *setModal()*. Si se desea cambiar la configuración preestablecida de las preferencias, el usuario modifica los datos en la interfaz y se guardan en un fichero mediante los métodos *guardarPreferencias()* de *PreferenciaControl* y el constructor de *BSPreferencia*. Igualmente si lo desea

puede configurar las preferencias de las PC clientes que va utilizar en las simulaciones distribuidas, se utiliza la interfaz *VisualCliente*, estos datos se guardan en el mismo fichero mediante el método *configurarClientes()* de *PreferenciaControl* y el constructor de *BSPC().*

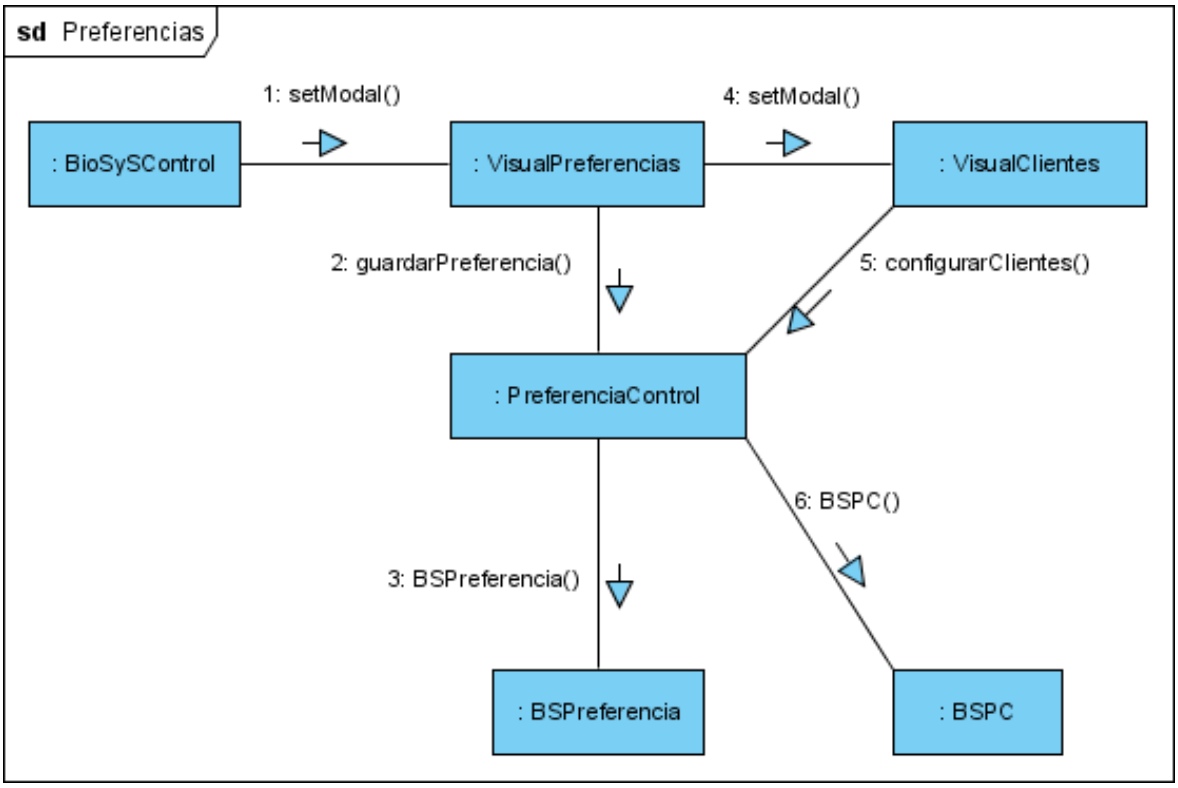


Figura 3.5 Diagrama de colaboración. Escenario Guardar Preferencias.

Realizar simulación local con ODEtoJava: La clase *BioSySControl* muestra la interfaz *VisualSimulacion* a través del método *setModal(),* esta clase manda a ejecutar el método *simularLocal()* de *SimulacionControl*, en este método se crea la lista de condiciones(variables y parámetros) y se manda a simular con ODEtoJava a la clase *BSSimulacion*, pasándole al método *Simular()* la lista de condiciones, los valores de tiempo y tolerancia y un fichero.class que contiene el modelo matemático, para cargar el modelo del fichero se utiliza la clase *BSClassLoader*. En la clase *SimulacionControl* se crea un objeto de la clase *BSResultado* a la que se le asigna el valor que devuelve *Simular()* con el resultado de la simulación. El método *SimularLocal()* es el encargado de guardar los resultados en la Base de Datos (BD) cuando se termine la simulación.

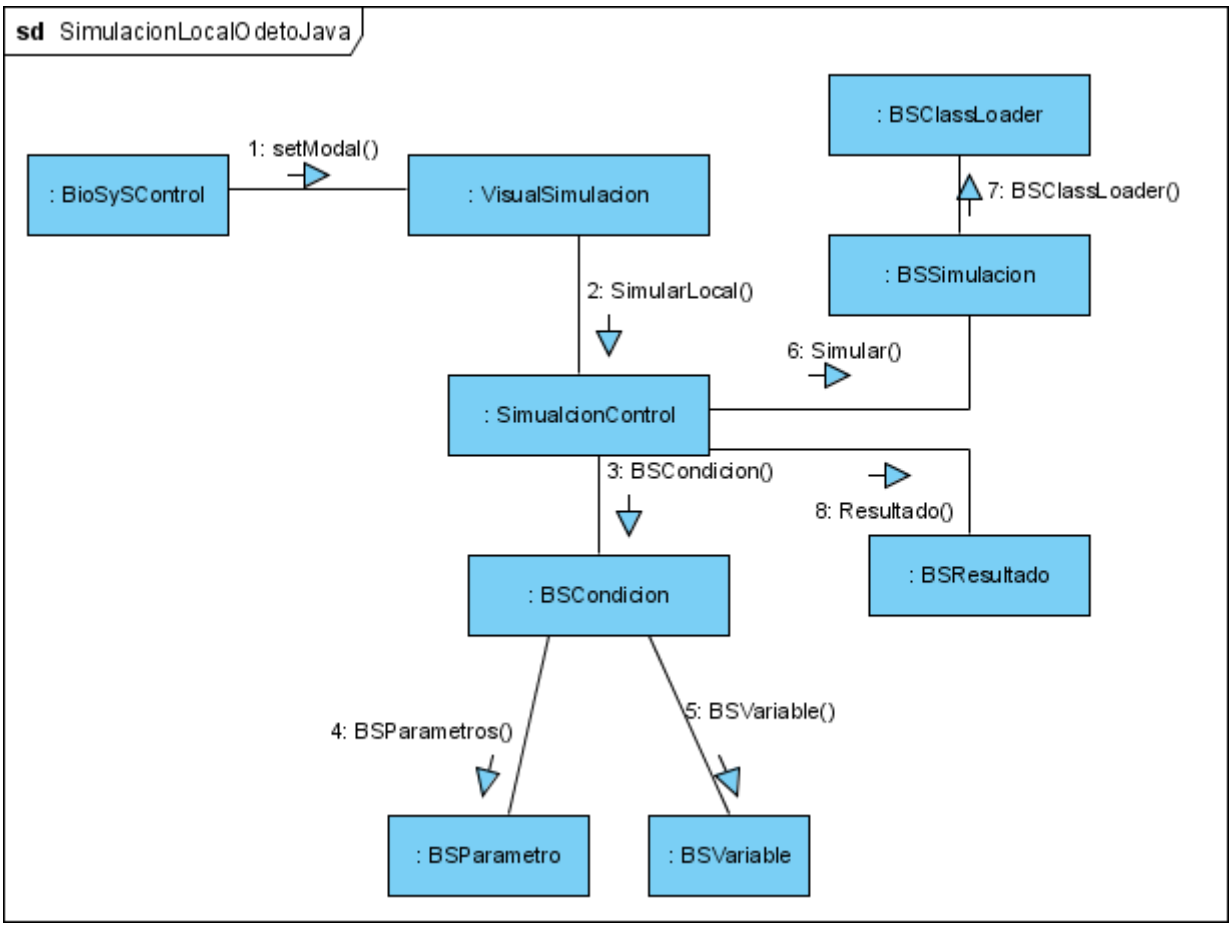


Figura 3.6 Diagrama de colaboración. Escenario Realizar simulación local con ODEtoJava.

Realizar simulación local con MatLab: La clase *BioSySControl* muestra la interfaz *VisualSimulacion* a través del método *setModal(),* esta clase manda a ejecutar el método *SimularLocal()* de *SimulacionControl*, en este método se crea la lista de condiciones(variables y parámetros), crea un objeto MatLab y lo ejecuta utilizando la clase *BSEngine* (clase que se utiliza para ejecutar MatLab desde aplicaciones en Java) luego llama al método *Simular()* de la clase *BSSimulacion* pasándole como parámetros: el objeto MatLab, el modelo matemático, las condiciones y los valores de los tiempos y tolerancias. En la clase *SimulacionControl* se crea un objeto de la clase *BSResultado* a la que se le asigna el valor que devuelve *Simular()* con el resultado de la simulación. El método *SimularLocal()* es el encargado de guardar los resultados en la Base de Datos (BD) cuando se termine la simulación.

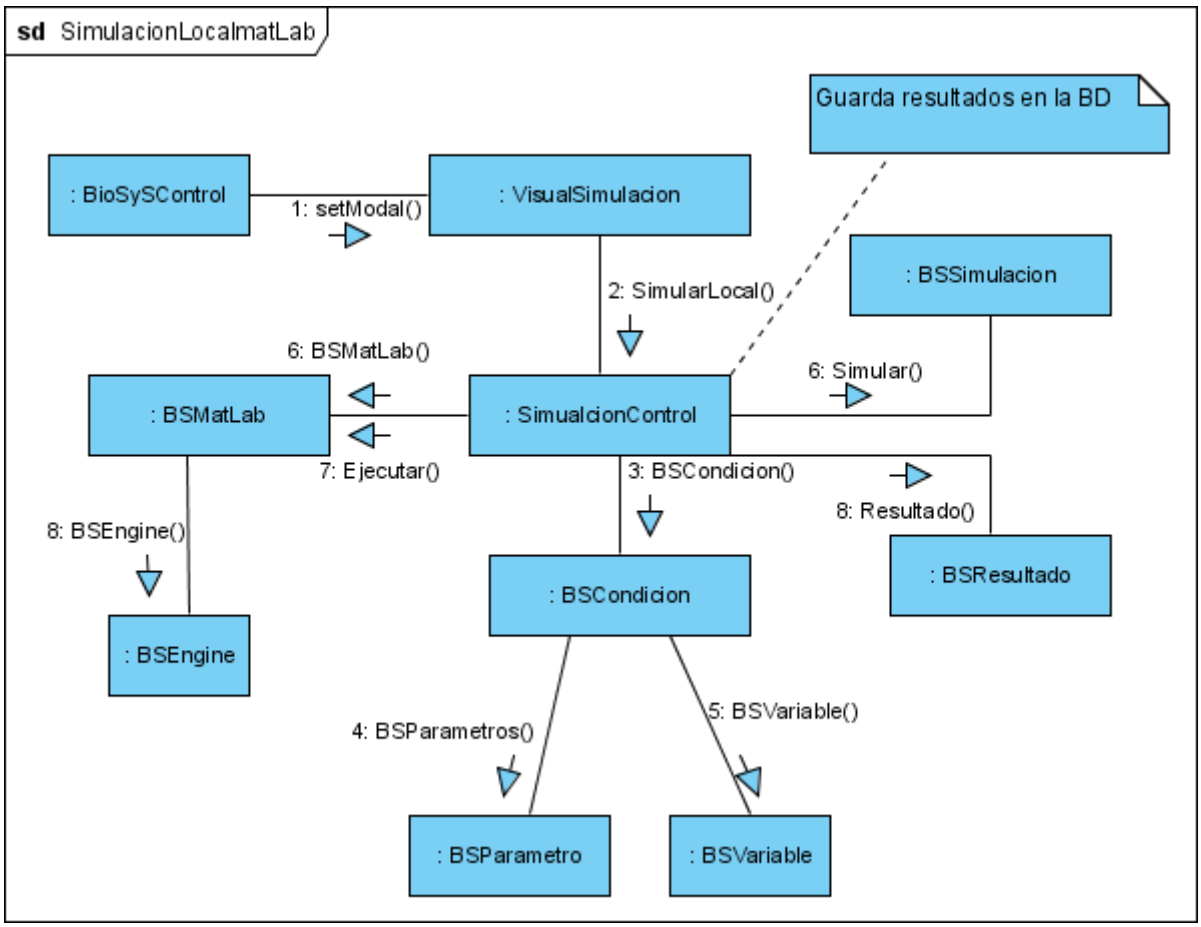


Figura 3.7 Diagrama de colaboración. Escenario Realizar simulación local con MatLab.

Realizar simulaciones distribuidas con ODEtoJava: La clase *BioSySControl* muestra la interfaz *VisualSimulacion* a través del método *setModal(),* esta clase manda a ejecutar el método *SimularDistribuido()* de *SimulacionControl* y este al método *generateWorkUnit()* de *BSDataManager* que crea la lista de condiciones(variables y parámetros), la divide y devuelve un vector con el número ip de la PC cliente, los valores de los tiempos y la tolerancia y dos lista con los valores de las variables y los parámetros, este vector lo utiliza *BSAlgorithm* en el método *processUnit(),* donde crea un objeto de *BSSimulacion* y llama al método *Simular()* pasándole todos los datos contenidos en el vector, (este método utiliza *BSClassLoader* para cargar el fichero del modelo matemático), se simula haciendo uso de la librería ODEtoJava contenida en *Simulacion* y se envían los resultados a *BSAlgorithm* que los guarda en la BD.

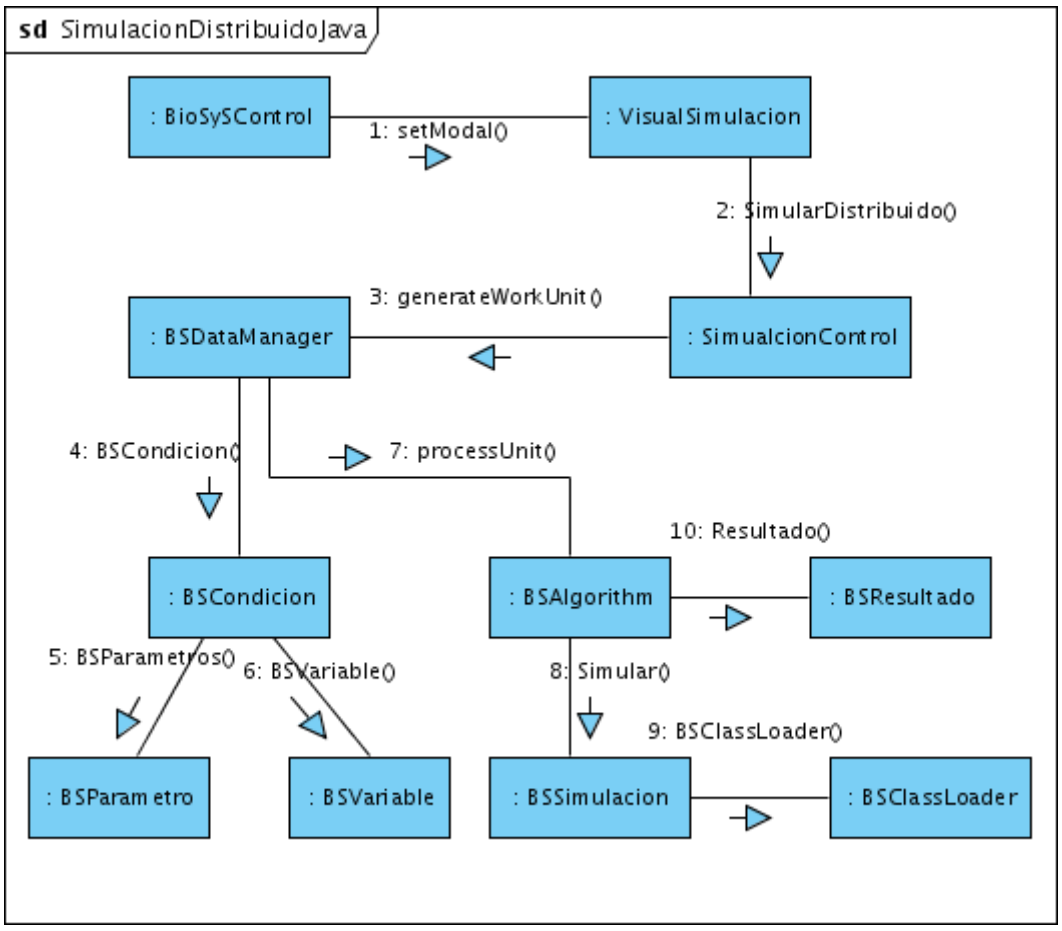


Figura 3.8 Diagrama de colaboración. Escenario Realizar simulaciones distribuidas con ODEtoJava.

Realizar simulaciones distribuidas con MatLab: La clase *BioSySControl* muestra la interfaz *VisualSimulacion* a través del método *setModal(),* esta clase manda a ejecutar el método *SimularDistribuido()* de *SimulacionControl* y este al método *generateWorkUnit()* de *BSDataManager* que crea la lista de condiciones(variables y parámetros), la divide y devuelve un vector con el número ip de la PC cliente, los valores de los tiempos y la tolerancia y dos lista con los valores de la variables y los parámetros, este vector lo utiliza *BSAlgorithm* en el método *processUnit(),* donde crea un objeto MatLab y un objeto *BSSimulacion*, llama al método *Simular()* de esta clase pasándole todos los datos contenidos en el vector y el objeto MatLab cuando se termina de simular se envían los resultados a *BSAlgorithm* que los guarda en la BD.

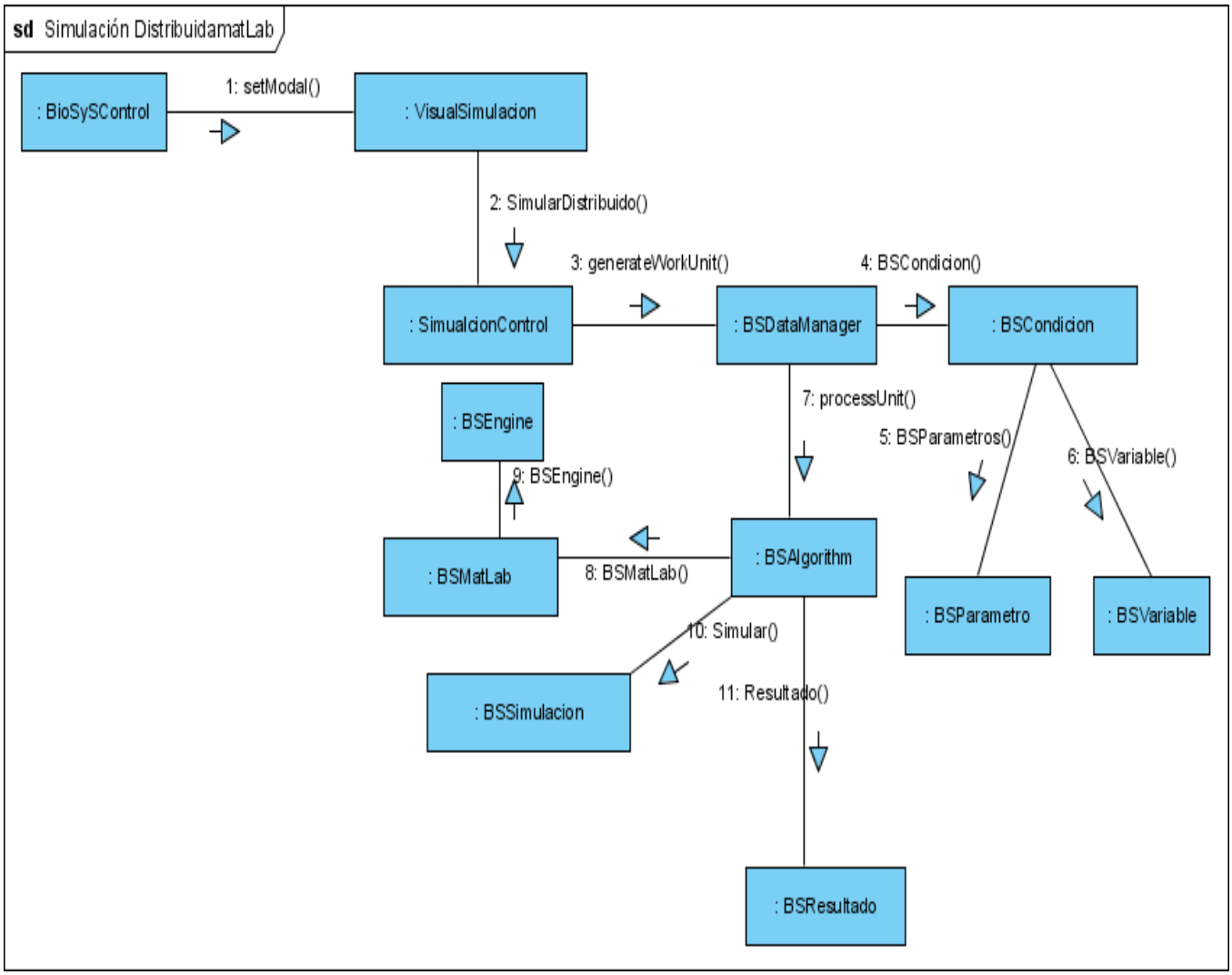


Figura 3.9 Diagrama de colaboración. Escenario Realizar simulaciones distribuidas con MatLab.

3.4 Modelo de Despliegue

El diagrama de despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad del mismo entre los nodos de cómputo que utiliza. No es más que la representación física de todos los recursos que utiliza el sistema, por ejemplo, nodos con capacidad de procesamiento, módems e impresoras. [28]

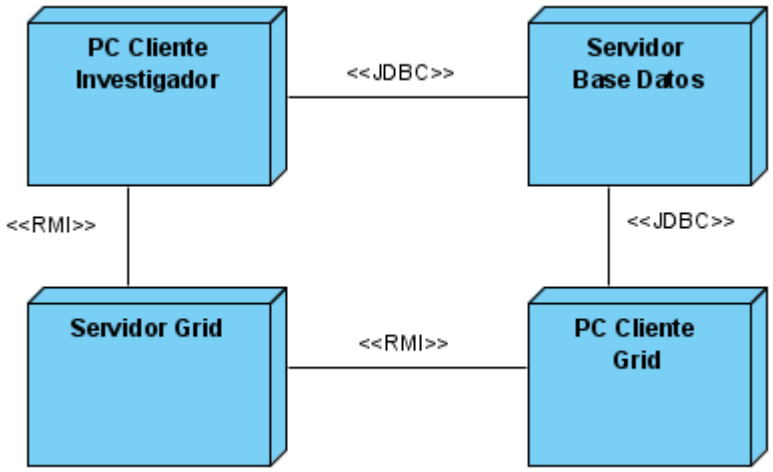


Figura 3.10 Diagrama de Despliegue.

3.5 Conclusiones del capítulo

Como resultado de este capítulo se conoce cómo fue diseñado el software, partiendo de las clases del diseño definidas. Se conoció cómo se llevan a cabo las funcionalidades que requieren el sistema y la distribución física de los nodos de procesamiento que se necesitan para su correcto funcionamiento y mejor rendimiento.

Capítulo 4: Implementación

Introducción

En este capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, para esto se muestra el diagrama de componentes y el diagrama de despliegue detallado. Además se mostrará el código fuente de los principales métodos de la aplicación.

4.1 Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software, sean estos componentes de código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización y las restricciones impuestas por los lenguajes de programación y las herramientas utilizadas en el desarrollo. Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. [29]

A continuación una breve descripción de los principales componentes que integran esta aplicación.

Simulacion: Este componente reúne todas las clases con las funcionalidades necesarias para realizar las simulaciones, guardar los datos de las variables y parámetros, ejecutar el comando de MatLab y realizar las simulaciones ya sea con ODEtoJava o con MatLab.

Presentacion: Reúne todas las clases interfaz que interactúan con el usuario, encargadas de recoger toda la información necesaria para la configuración de las simulaciones: configuración del servidor que se va a utilizar, definición de la herramienta matemática y el método numérico, de los valores de las variables, los parámetros, los tiempos de integración y los clientes que se utilizaran para simular.

BSDAO: Capa de acceso a datos que se utiliza para guardar en la BD.

jdsGUILibrary: Librería para conectarse al servidor Grid.

Subsistema t-arenal: Se encuentran los componentes *BSDataManager*, *BSAlgorithm* y *Grid* encargados de fragmentar y repartir las simulaciones entre las máquinas clientes disponibles y almacenar los resultados en la Base de Datos. También se encuentra en este subsistema el componente *Simulacion*.

MatLab: Este componente representa la aplicación de MatLab que se ejecutará en cada PC cliente para resolver los cálculos de las simulaciones.

Diagrama de Componentes para cuando se realizan simulaciones locales: En este caso el componte Presentacion utiliza la BD para cargar el modelo matemático, si se va a utilizar como herramienta matemática MatLab la clase *SimulacionControl* del componente *Presentacion* manda a ejecutar MatLab y utiliza el componente *Simulacion* para enviar toda la información del modelo matemático y el método numérico a la función de MatLab que lo resuelve, cuando este termina envía los resultados al componente *Simulacion* y este a la clase *SimulacionControl* del componente *Presentacion* que guarda los resultados en la BD mediante un objeto DAO. Si se simula con la librería ODEtoJava incluida en *Simulacion*, se devuelven los datos al concluir los cálculos al componente *Presentacion* y se guardan en la BD de la misma forma.

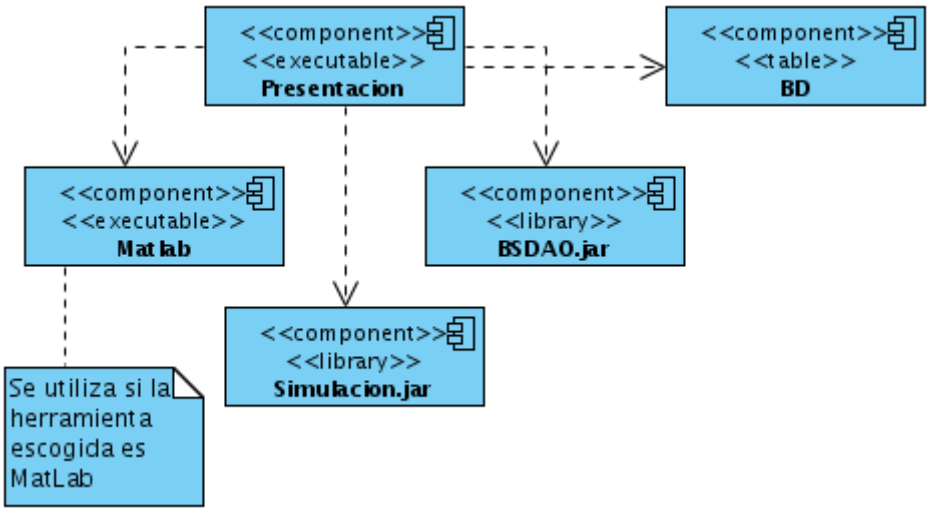


Figura 4.1 Diagrama de Componentes. Simulaciones Locales.

Diagrama de Componentes para cuando se realizan simulaciones distribuidas: El componente *Presentacion* utiliza la base de datos para cargar los modelos matemáticos que se encuentran almacenados y luego se conecta al servidor Grid, *BSDataManager* es el encargado de generar las unidades de trabajo que serán enviadas a los clientes junto con los componentes *BSAlgorithm*, *Simulacion* y *BSDAO*. En el cliente *BSAlgorithm* ejecuta MatLab si se decidió utilizar esta herramienta y luego manda a simular con *Simulacion*, cuando termina la simulación se devuelven los resultados a *BSAlgorithm* que es el responsable de guardarlos en la BD utilizando un objeto *DAO*.

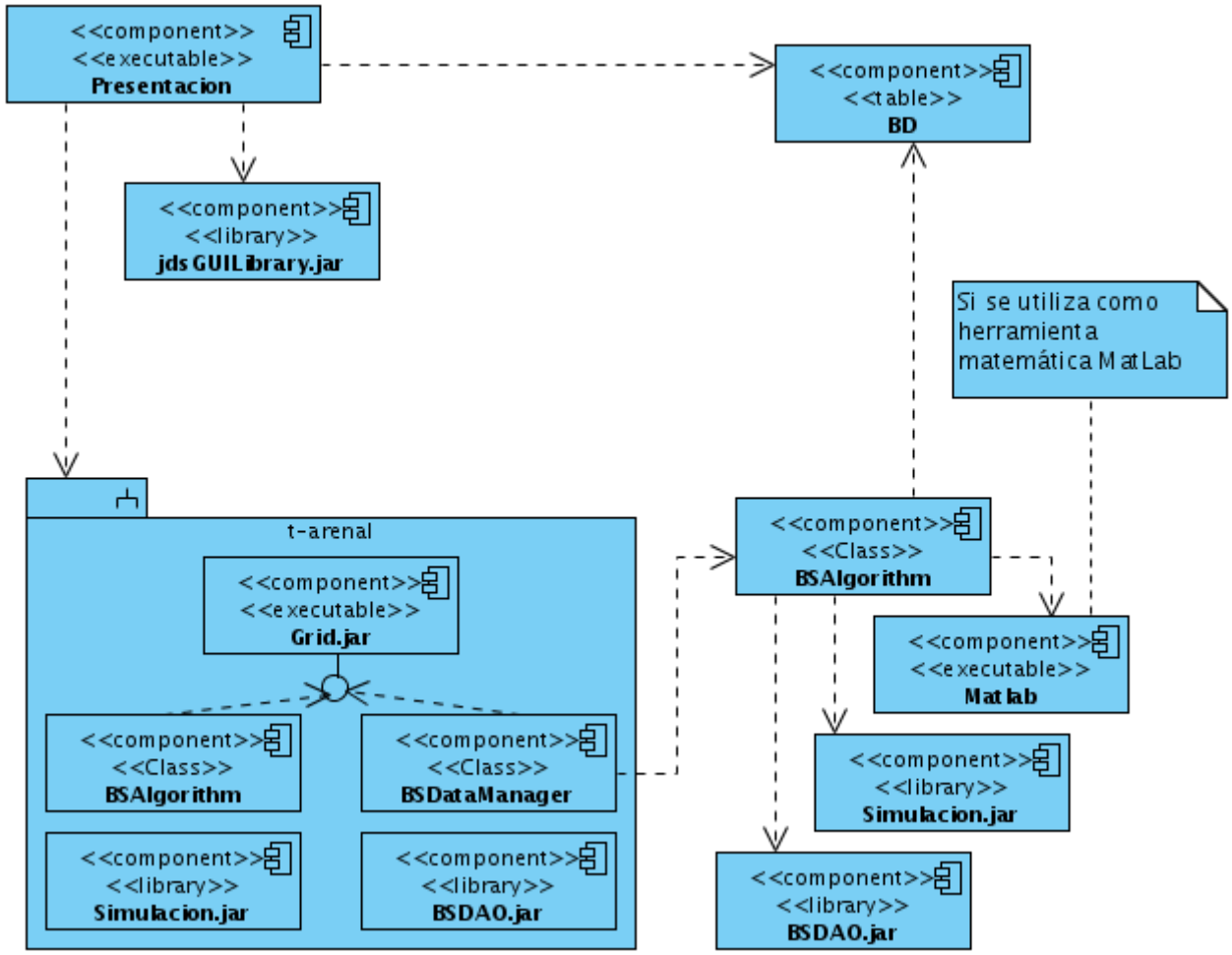


Figura 4.2 Diagrama de Despliegue. Simulaciones Distribuidas.

4.2 Diagrama de despliegue detallado

Nodo PC Cliente: Representa la máquina cliente asignada al investigador, que en este caso es el cliente del sistema, permite lograr que las peticiones del cliente sean atendidas por el servidor t-arenal, esto se logra haciendo uso de objetos distribuidos con tecnología RMI teniendo en cuenta que es la tecnología asociada al lenguaje Java.

Nodo t-arenal: Representa un servidor de objetos que contiene objetos que pueden accederse a distancia, los cuales son tratados por el programador como si estuvieran en su computadora local. Por ser una plataforma de cálculo distribuido permite una mayor agilidad en el proceso de cálculo.

Nodos PC Cliente: Representan las máquinas que están a disposición del servidor de la Grid (servidor de objetos) son las encargadas de realizar el trabajo enviado por él y para ello pueden tener instalada la herramienta de cálculo MatLab.

Nodo Servidor de BD: Representa un servidor de Base de Datos utilizado para el almacenamiento de los datos de la aplicación y para lograr la conexión del sistema con la base de datos se utiliza JDBC como protocolo de comunicación.

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

RMI (Java Remote Method Invocation): Es un mecanismo ofrecido en Java para invocar un método remotamente. 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.

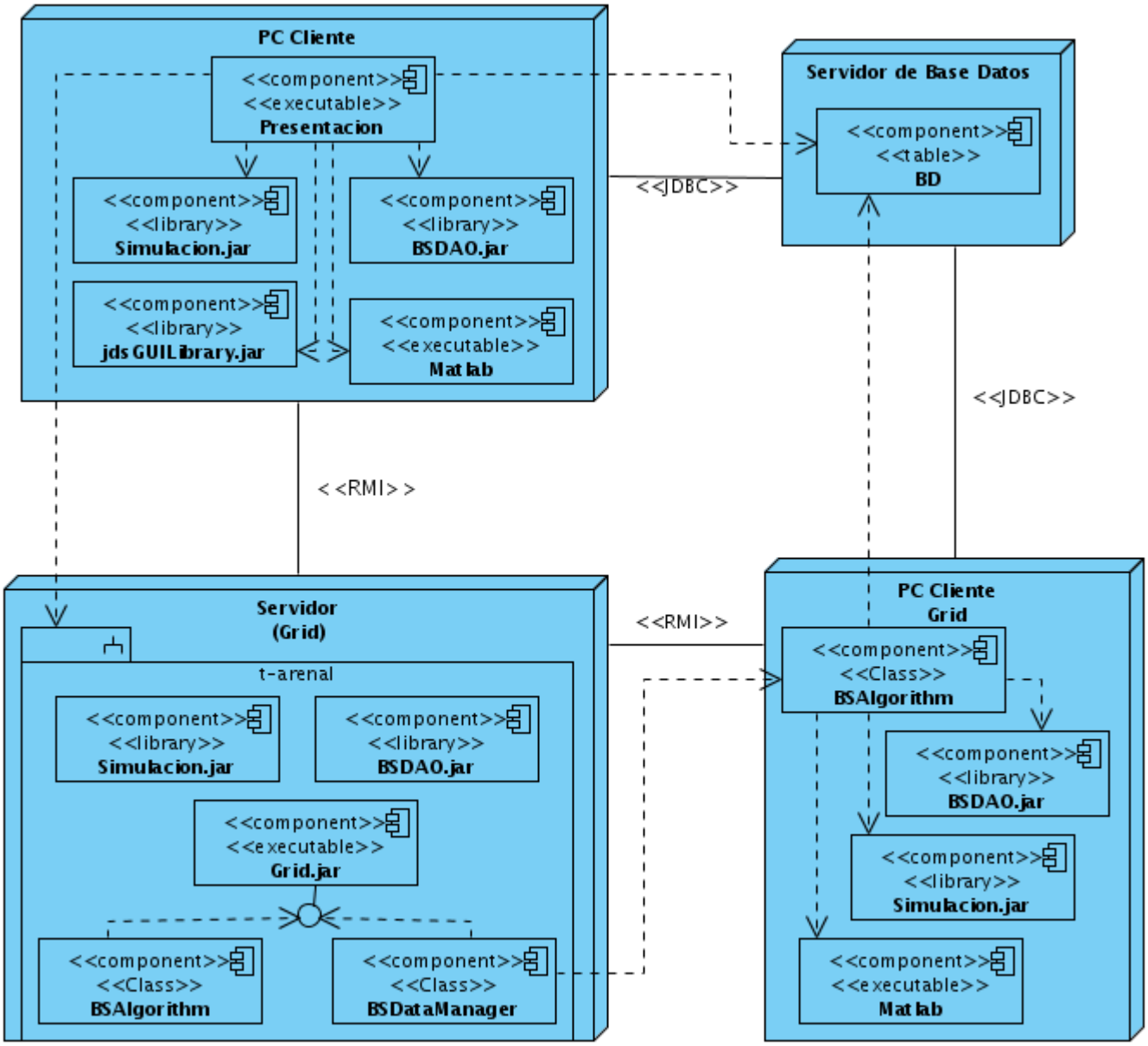


Figura 4.3 Diagrama de Despliegue Detallado.

En el nodo PC Cliente Investigador se encuentran los componentes necesarios para realizar las simulaciones locales, en caso de que se desee simular con MatLab es indispensable que esté instalado en esta PC. Desde este nodo se puede guardar los resultados de las simulaciones en la BD cuando se realicen completamente. Si desea realizar simulaciones distribuidas se utiliza la plataforma de cálculo distribuido t-arenal donde se encuentran los componentes propios del servidor y los componentes *Simulacion* y *BSDAO*. (Estos componentes no se utilizan en el servidor pero están ubicados ahí para enviarlos a cada uno de los clientes)

El servidor y los clientes tienen definido un puerto socket mediante el cual se realizan las peticiones de los clientes y el envío de datos del servidor hacia ellos, por esta vía el servidor envía a cada PC cliente el problema que tiene que resolver y los componentes *BSAlgorithm*, *Simulacion* y *BSDAO*. Cuando el cliente termina de simular se envían los resultados a la BD a través de *BSAlgorithm*.

4.3 Código fuente de principales clases

El paquete de clases más importante de nuestro sistema es el paquete *Simulacion* que contiene los tres métodos fundamentales para gestionar las simulaciones, estos son los tres métodos *Simular()* de la clase *BSSimulacion* a continuación la descripción de cada uno de ellos:

Simular con MatLab:

Este método ejecuta la simulación utilizando MatLab, recibe como parámetros una instancia de la clase *BSMatLab*, el modelo matemático a simular, una matriz de condiciones iniciales para las n simulaciones a realizar (cond), el tiempo inicial (ti) y final (tf), la tolerancia absoluta (eam)y relativa(erm), el paso a utilizar (_ step_ ,en caso de ser autoajustable será 0). Inicialmente se verifica si existe alguna interface de progreso y se inicializa en 0, luego levanta un servidor socket por el puerto especificado, se crea un hilo que ejecuta la función de MatLab que resuelve el modelo matemático con los parámetros correspondientes. Cuando MatLab pide la lista de condiciones que necesita para simular se crea el objeto con los datos necesarios y se envía por el puerto socket, espera que MatLab avise por el puerto socket que ha realizado una simulación e incrementa el valor de la interface de progreso si existe. Verifica si se realizaron todas las simulaciones, recibe el objeto con las mismas y cierra la conexión del servidor socket. Devuelve una lista (ArrayList<Result>) con el resultado de la n simulaciones realizadas.

```
public ArrayList<Result> Simular(BSMatlab matlab, File model, ArrayList
cond, double ti,
            double tf, double eam, double erm, double step)
            throws IOException, ClassNotFoundException,
InterruptedException{

        if(getProgreso()!=null)
            getProgreso().setValue(0);
        ServerSocket server = new ServerSocket(matlab.getPort());
        BSHiloMatlab m = new BSHiloMatlab(matlab, "BSSimulaciones
('"+model.getName().substring(0,model.getName().indexOf(".m"))
                        + "', '"+model.getParent()+"', ["+ti+" "+tf+"],
'"+matlab.getMethod()
```

```
                        +"', "+step+", "+eam+", "+ erm+",
"+matlab.getPort()+")");
        m.start();
        Socket client = server.accept();
        ObjectOutputStream out = new
ObjectOutputStream(client.getOutputStream());
        out.writeObject(cond);
        setCont(0);
        do{
            server.accept();
            setCont(getCont() + 1);
            if(getProgreso()!=null)
                getProgreso().setValue(getCont());

        } while(getCont()!=((ArrayList)cond.get(0)).size());

        ObjectInputStream data = new
ObjectInputStream(client.getInputStream());
        server.close();

        return (ArrayList)data.readObject();
    }
```

Tabla 4.1 Método Simular con MatLab.

Simular con ODEtoJava (Resuelve una sola simulación):

Este método ejecuta las simulaciones utilizando la librería ODEToJava. Recibe como parámetros: un objeto de tipo ODE (iface), un vector con los valores de las condiciones iniciales (initialCond), un vector con los valores de los parámetros (par), el Tiempo Inicial (ti) y Tiempo Final (tf), el valor de la Tolerancia Absoluta (eam) y la Tolerancia Relativa (erm), el Paso a utilizar (step, de ser autoajustable tomará valor 0). Inicialmente verifica el valor del paso, si es autoajustable se crea un objeto que contiene el rango de tiempo inicial y tiempo final, de no ser autoajustable el paso crea un objeto con el rango de tiempo inicial, tiempo final y el paso. Se crean dos arreglos uno para el error absoluto y otro para el error relativo máximo para cada una de las ecuaciones del sistema, además se crean los arreglos de condiciones iniciales y parámetros respectivamente. Finalmente resuelve el modelo matemático con los parámetros y condiciones especificados. Devuelve el resultado de la simulación realizada (Result).

```
 public  Result Simular(ODE iface, ArrayList<Double> initialCond,
ArrayList<Double> par, double ti, double tf,
            double eam, double erm, double step)
            throws ClassNotFoundException, InstantiationException,
IllegalAccessException, IOException {
```

```
        Span span;
        if (step == 0) {

            span = new Span(ti, tf);
        } else {

            span = new Span(ti, tf, step);
        }

        double[] atol1 = new double[initialCond.size()];

        double[] rtol1 = new double[initialCond.size()];
        for (int i = 0; i < initialCond.size(); i++) {
            atol1[i] = eam;
            rtol1[i] = erm;
        }

        double[] c = new double[initialCond.size()];
        for (int j = 0; j < c.length; j++) {
            c[j] = initialCond.get(j);
        }

        double[] m = new double[par.size()];
        for (int j = 0; j < m.length; j++) {
            m[j] = par.get(j);
        }

        return ErkTriple.erk_triple(iface, span, c, m, -1.0,
                                    new Btableau("dopr54"), 5.0, atol1,
rtol1,
                                    "StiffDetect_Off", "EventLoc_Off",
"Stats_Off");



      }
```

Tabla 4.2 Método Simular con ODEtoJava. Realiza una simulación.

Simular con ODEToJava (Resuelve varias simulaciones):

Este método ejecuta las simulaciones utilizando la librería ODEToJava. Recibe como parámetros: el Modelo Matemático a simular (model), una lista de *BSCondicion* que incluye las Variable y los Parámetros con sus respectivos valores , el Tiempo Inicial (ti) y Tiempo Final (tf), el valor de la Tolerancia Absoluta (eam) y la Tolerancia Relativa (erm), el Paso a utilizar (step, de ser autoajustable tomará valor 0). Inicializa dos arreglos de contadores, el primero para las variables y parámetros, y el segundo para banderas de las variables y parámetros que verifica si fue incrementado o no, inicializa además un contador para la cantidad de simulaciones (todos los contadores son inicializados en 0 y

las banderas en false). Declara una instancia de la clase *BSClassLoader*, se carga la clase (modelo matemático) y se le asigna al objeto cl (Objeto cl: objeto de tipo Class), luego castea ese objeto a ODE y se lo asigna a iface (iface: objeto de tipo ODE). Inicializa una variable con la cantidad total de combinaciones posibles de simulaciones a realizar. Declara tres listas, una de resultados donde se guardaran los resultados de las simulaciones, la segunda contiene una combinación de los valores de las variables y la tercera una combinación de los valores de los parámetros.
Genera una combinación de variables y parámetros. Incrementan los contadores en forma de reloj y se actualizan las banderas. Se llama al método *Simular()* que realiza una sola simulación e incrementa el contador de simulaciones y la interface progreso si existe. Devuelve una lista de resultados de las n simulaciones realizadas.

```
public ArrayList GenerarSimulaciones(ArrayList<BSCondicion> v) throws
IOException, ClassNotFoundException{

        ArrayList result = new ArrayList();

        int[] con = new int[v.size()];
        boolean[] conv = new boolean[v.size()];
        setCont(0);
        for (int i = 0; i < con.length; i++) {
            con[i] = 0;
            conv[i] = false;
        }
        int cantsim = getCantTotal(v);
        ArrayList vpop = new ArrayList();
        ArrayList vpar = new ArrayList();
        do {
            ArrayList vpopsim = new ArrayList();
            ArrayList vparsim = new ArrayList();
            for (int i = 0; i < v.size(); i++) {
                if (v.get(i) instanceof BSVariable) {
                    vpopsim.add(v.get(i).getValores().get(con[i]));
                } else if (v.get(i) instanceof BSParametro) {
                    vparsim.add(v.get(i).getValores().get(con[i]));
                }
            }
            vpop.add(vpopsim);
            vpar.add(vparsim);

            if (con[con.length - 1] != v.get(v.size() -
1).getValores().size() - 1) {
                con[con.length - 1]++;
                conv[con.length - 1] = false;
            } else {
                con[con.length - 1] = 0;
```

```
                conv[con.length - 1] = true;
            }
            if (con[con.length - 1] == 0) {
                for (int i = con.length - 1; i > 1; i--) {
                    if (con[i] == 0 && conv[i]) {
                        if (con[i - 1] != v.get(i - 1).getValores().size()
- 1) {

                            con[i - 1]++;
                            conv[i - 1] = false;
                        } else {
                            con[i - 1] = 0;
                            conv[i - 1] = true;
                        }

                    }
                }
            }
            if (conv[1] && con[1] == 0) {
                if (con[0] != v.get(0).getValores().size() - 1) {
                    con[0]++;
                    conv[0] = false;
                } else {
                    con[0] = 0;
                    conv[0] = true;
                }
            }
            for (int i = 0; i < conv.length; i++) {
                conv[i] = false;
            }

            setCont(getCont() + 1);

            if(getProgreso()!=null)
                getProgreso().setValue(getCont());
        } while (getCont() != cantsim);

        result.add(vpop);
        result.add(vpar);
        return result;
    }
```

Tabla 4.3 Método Simular con ODEtoJava. Realiza múltiples simulaciones.

Otras clases importantes para la gestión de las simulaciones de forma distribuida son *BSDataManager* y *BSAlgorithm* encargadas de generar y procesar las unidades de trabajo respectivamente. A continuación los principales métodos de estas clases:

La clase *BSDataManager*, es la encargada de generar las unidades de trabajo que son procesadas por *BSAlgorithm*, para generar las unidades de trabajo utiliza el método *generateWorkUnit().*

Generar unidad de trabajo:

Este método recibe la información del cliente (*clientInfo* contiene ip, sistema operativo, etc) cuando este hace una petición al servidor. Verifica que el sistema operativo sea Linux ya que se va a simular con MatLab, luego verifica si el ip está en la lista que contiene los ip donde ya se han realizado simulaciones anteriormente, mediante el método *isAllowed(ip)* y si existen simulaciones fallidas, de existir alguna se devuelve en el vector output. Si no existen simulaciones fallidas, se fragmenta la lista de condiciones, de cada fragmento se asignan los valores de las variables y los parámetros a dos listas distintas vpop y vpar. Verifica que cantidad de trabajo se le puede enviar al cliente utilizando la variable granularity. Devuelve un vector output con el ip, las listas de condiciones (vpop y vpar) y las properties (nombre del modelo, nombre del método numérico, tiempos de integración, tolerancias relativa y absoluta y el paso con el que se va a integrar) y un vector v que contiene el comando para ejecutar MatLab y el puerto socket.

```
public Vector generateWorkUnit(ClientInfo clientInfo) throws Throwable {
        String ip = clientInfo.getIP();
        Vector output = new Vector();
        if(properties.getProperty("sistema").equalsIgnoreCase("matlab")){
        Vector v = null;
        if (clientInfo.getOS().toLowerCase().contains("linux")) {
            v = isAllowed(ip);
        }

        if (v != null && !isFailet(ip)) {
            if (simulacionesFallidas.size() != 0) {
                output = simulacionesFallidas.get(0);
                simulacionesFallidas.remove(0);
                output.set(0, ip);
                delivery.println(((ArrayList) output.get(3)).size() +
"\t" + ip);
                pendingUnits++;
                //  output.set(1, client);
                return output;
            }
            if (cont == cantsim) {
                return null;
            }

            ArrayList vpop = new ArrayList();
            ArrayList vpar = new ArrayList();
            ArrayList idsim = new ArrayList();
            ArrayList tiempos = new ArrayList();
            for (int i = 0; i < granularity&&cont!=cantsim; i++) {
                vpop.add(cond.get(0).get(cont));
```

```
                vpar.add(cond.get(1).get(cont));
                idsim.add(cond.get(2).get(cont));
                tiempos.add(cond.get(3).get(cont));
                cont++;
            }
            delivery.println(granularity + "\t" + ip);
            pendingUnits++;
            output.add(ip);
            output.add(properties);
            output.add(v);
            output.add(vpop);
            output.add(vpar);
            output.add(idsim);
            output.add(tiempos);
            return output;
        } else {
            delivery.println("No enviado a: \t" + ip);
            return null;

        }
        }else{

        if (!isFailet(ip)) {
            if (simulacionesFallidas.size() != 0) {
                output = simulacionesFallidas.get(0);
                simulacionesFallidas.remove(0);
                output.set(0, ip);
                delivery.println(((ArrayList) output.get(3)).size() +
"\t" + ip);
                pendingUnits++;
                //  output.set(1, client);
                return output;
            }
            if (cont == cantsim) {
                return null;
            }

            ArrayList vpop = new ArrayList();
            ArrayList vpar = new ArrayList();
            ArrayList idsim = new ArrayList();
            ArrayList tiempos = new ArrayList();
            for (int i = 0; i < granularity&&cont!=cantsim; i++) {
                vpop.add(cond.get(0).get(cont));
                vpar.add(cond.get(1).get(cont));
                idsim.add(cond.get(2).get(cont));
                tiempos.add(cond.get(3).get(cont));
                cont++;
            }
            delivery.println(granularity + "\t" + ip);
            pendingUnits++;
            output.add(ip);
```

```
            output.add(properties);
            output.add(vpop);
            output.add(vpar);
            output.add(idsim);
            output.add(tiempos);
            return output;
        } else {
            delivery.println("No enviado a: \t" + ip);
            return null;
        }
        }
    }
```

Tabla 4.4 Método Generar Unidades de Trabajo.

La clase BSAlgorithm utiliza el método processUnit() que recibe como parámetro la unidad de trabajo generada por BSDataManager.

Procesar unidad de trabajo:

Este método recibe el vector que devuelve *BSDataManager*. Castea la información de la posición 1 del vector *workUnit()* como Properties y se lo asigna a un objeto. Verifica si es MatLab la herramienta matemática y crea una lista con los valores de las variables y los parámetros, el id de la simulación y los tiempos contenidos en las posiciones 3, 4, 5 y 6 del vector *workUni().* Se descarga el fichero del modelo matemático y el directorio de las funciones de MatLab (*BSToolbox.zip*) que luego se descompacta. Se verifica si el vector v (de la clase *BSDataManager,* ocupa la posición 2 del vector que se recibe como parámetro) contiene el comando y el puerto, en caso de no contener alguno de los dos o ninguno, los genera y crea un objeto MatLab específico en cada caso. Ejecuta el método *Simular()* pasándole todos los parámetros y se lo asigna a un objeto *BSResulado.* Si la herramienta matemática es ODEtoJava se crea una lista con los valores de las variables y los parámetros, el id de la simulación y los tiempos contenidos en las posiciones 3, 4, 5 y 6 del vector *workUni().* Declara una instancia de la clase *BSClassLoader*, se carga la clase (modelo matemático) y se le asigna al objeto cl (Objeto cl: objeto de tipo Class), luego castea ese objeto a ODE y se lo asigna a iface (iface: objeto de tipo ODE). Se llama al método *Simular()* pasándole todos los parámetros y se lo asigna a un objeto *BSResulado*. Al finalizar guarda los resultados en la BD.

```
public Vector processUnit(Vector workUnit) throws Throwable {
        properties = (Properties) workUnit.get(1);
        ArrayList<Set<Resultado>> result = null;
        Vector v = new Vector();

        ps = new PrintStream(new FileOutputStream(new
```

```
File(PROBLEMDIRECTORY,
                "Progreso.txt")), true);

        ArrayList r = new ArrayList();
        if(properties.getProperty("sistema").equalsIgnoreCase("matlab")){
        r.add(workUnit.get(3));
        r.add(workUnit.get(4));
        r.add(workUnit.get(5));
        r.add(workUnit.get(6));
        downloadFile(properties.getProperty("model.name") + ".m");
         downloadFile("BSToolbox.zip");
        File compactado = getFile("BSToolbox.zip");
        File RESOURCE_FOLDER = new
File(PROBLEMDIRECTORY.getParentFile().getParentFile(), "resources");

        boolean flag;
        if (!RESOURCE_FOLDER.exists()) {
            flag = RESOURCE_FOLDER.mkdirs();
            if (flag == false) {

                throw new Exception("Unable to create resource folder");
            }
        }
       File BSTOOLBOX_FOLDER = new File(RESOURCE_FOLDER, "BSToolbox");

        if (!BSTOOLBOX_FOLDER.exists()) {
            flag = BSTOOLBOX_FOLDER.mkdirs();
            if (flag == false) {

                throw new Exception("Unable to create BSToolbox folder");
            }
        }
        descompactar2(compactado.getAbsolutePath(),
BSTOOLBOX_FOLDER.getAbsolutePath());
         if (((Vector) workUnit.get(2)).size() == 2) {
                ps.println(1);
                m = new BSMatlab(BSTOOLBOX_FOLDER.getAbsolutePath(),
(String) ((Vector) workUnit.get(2)).get(0), Integer.parseInt((String)
((Vector) workUnit.get(2)).get(1)));
            } else if (((Vector) workUnit.get(2)).size() == 1) {
                ps.println(2);
                m = new BSMatlab(BSTOOLBOX_FOLDER.getAbsolutePath(),
(String) ((Vector) workUnit.get(2)).get(0));
            } else {
                ps.println(3);
                m = new BSMatlab(BSTOOLBOX_FOLDER.getAbsolutePath());
            }
            ps.println(4);
            m.setMethod(properties.getProperty("metodo.name"));
            m.setPort(m.generatePort());
            m.Ejecutar();
```

```
            ps.println(5);

             try {
            BSSimulacion s = new BSSimulacion();
              result = s.Simular(m,
                    new File(PROBLEMDIRECTORY,
properties.getProperty("model.name") + ".m"),
                    r,
Double.parseDouble(properties.getProperty("time.initial")),

Double.parseDouble(properties.getProperty("time.final")),
                    Double.parseDouble(properties.getProperty("eam")),
                    Double.parseDouble(properties.getProperty("erm")),
                    Double.parseDouble(properties.getProperty("step")));
            ps.println(result.size());
        } catch (Exception ex) {
            ps.println(ex.getMessage());
            error = ex.getMessage();
        }
        if (m != null) {
            ps.println(m.getOutputString(1000));
            m.close();
        }
        }else{
        r.add(workUnit.get(2));
        r.add(workUnit.get(3));
        r.add(workUnit.get(4));
        r.add(workUnit.get(5));
        downloadFile(properties.getProperty("model.name") + ".class");
         try {
            BSSimulacion s = new BSSimulacion();
            File f = new File(PROBLEMDIRECTORY,
properties.getProperty("model.name") + ".class");
        ClassLoader loader = new BSClassLoader(f.getParent());
        Class cl = loader.loadClass(f.getName().substring(0,
f.getName().indexOf(".class")));
        ODE iface = (ODE) cl.newInstance();
        result = new ArrayList<Set<Resultado>>();
             for (int i = 0; i < ((ArrayList)r.get(0)).size(); i++) {

               result.add(
                       s.Simular(
                       iface,
                       ((ArrayList<ArrayList<Double>>)r.get(0)).get(i),
                       ((ArrayList<ArrayList<Double>>)r.get(1)).get(i),
                       ((ArrayList<String>)r.get(2)).get(i),

((ArrayList<ArrayList<Double>>)r.get(3)).get(i).get(0),

((ArrayList<ArrayList<Double>>)r.get(3)).get(i).get(1),
                       Double.parseDouble(properties.getProperty("eam")),
```

```
                    Double.parseDouble(properties.getProperty("erm")),
                    Double.parseDouble(properties.getProperty("step"))
                    )
                    );
             }
             ps.println(result.size());
        } catch (Exception ex) {
            ps.println(ex.getMessage());
            error = ex.getMessage();
        }
        if (m != null) {
            ps.println(m.getOutputString(1000));
            m.close();
        }
        }

        if (result != null && result.size() != 0) {
            BSDAOFactory bsDao = BSDAOFactory.getDAOFactory(
                    Integer.parseInt(properties.getProperty("gestorbd")),
                    properties.getProperty("servidorbd"),
                    properties.getProperty("puertobd"),
                    properties.getProperty("nombrebd"),
                    properties.getProperty("usuariobd"),
                    properties.getProperty("passbd"));

            for (int i = 0; i < result.size(); i++) {
                Object[] resultado = result.get(i).toArray();
                if (bsDao.getSimulacionDAO().getSimulacion(((Resultado)
resultado[0]).getIdsim()) == null) {
                    Simulacion sim = new Simulacion();

sim.setIdm(Integer.parseInt(properties.getProperty("idmodelo")));
                    sim.setIdsim(((Resultado) resultado[0]).getIdsim());
                    sim.setResultados(result.get(i));
                    bsDao.getSimulacionDAO().insertarSimulacion(sim);
                } else {
                    Simulacion sim =
bsDao.getSimulacionDAO().getSimulacion(((Resultado)
resultado[0]).getIdsim());
                    for (int j = 1; j < resultado.length; j++) {
                        sim.getResultados().add(resultado[j]);
                    }
                    bsDao.getSimulacionDAO().updateSimulacion(sim);
                }
            }
            ps.println("Devolver");
            v.add(1);
            v.add(workUnit.get(0));
            v.add(m.getCmd());
        } else {
```

```
            ps.println("error");
            if (error != null && (error.equalsIgnoreCase("Matlab") ||
error.equalsIgnoreCase("Puertos"))) {
                v.add(-1);
                v.add(workUnit.get(0));
                v.add(error);
                v.add(workUnit);
            } else {
                v.add(0);
                v.add(workUnit.get(0));
                v.add(error);
                v.add(workUnit);
            }
        }
        ps.close();
        return v;
    }
```

Tabla 4.5 Método Procesar Unidades de Trabajo.

4.4 Pruebas de rendimiento

Se realizaron pruebas para demostrar la superioridad del proceso de simulado cuando se realiza de forma distribuida sobre el proceso que se realiza de forma local. Para llevar a cabo estas pruebas se realizó la simulación del mismo modelo matemático, utilizando los mismos valores de condiciones iniciales, la librería ODEtoJava y el método numérico Runge-Kutta de cuarto orden tanto de forma local como distribuida. Se utilizó la librería ODEtoJava que realiza los cálculos de forma más ágil que la herramienta MatLab, además no se cuenta actualmente con un número suficiente de máquinas clientes con SO Linux para arrojar resultados significativos en cuanto a la superioridad de utilizar la infraestructura t-arenal sobre la realización de simulaciones locales.

El modelo matemático utilizado describe el comportamiento del sistema biológico compuesto por las células defectoras, reguladoras y el compuesto químico interleuquina. EL CIM (Centro de Inmunología Molecular) está trabajando con este sistema para el desarrollo de tratamientos contra el cáncer, por lo que los valores de las condiciones iniciales, los tiempos y las tolerancias usados son valores reales para este modelo matemático. A continuación el modelo matemático expresado en sus ecuaciones algebraicas y diferenciales:

	Ecuaciones algebraicas
	tb=(1/(2*K))*((K*(s+r+e)+1)-sqrt((K*(s+e+r)+1)^2-4*K^2*(r+e)*s))
	eb=(e/(e+r))*tb
	rb=(r/(e+r))*tb

Modelo Matemático	ef=e-eb rf=r-rb SigE=(Ke*il+i?)/(1+Ke*il+i?) SigR=(Kr*il)/(1+Kr*il) il1=il/(1+Ke*eb+Kr*rb)
	Ecuaciones diferenciales de = fe+pe*SigE*((1-rb/s)^(s-1))*eb-(1-?e)*pe*(1-SigE)*eb-ef dr = fr+pr*SigR*rb-(1-?r)*pr*(1-SigR)*rb-rf dil = fi+kpi*pe*((1-rb/s)^(s-1))*eb-kdi*il-ki*il1*(Ke*eb+Kr*rb)

Tabla 4.6 Modelo Matemático.

Nombre	Valor Inicial	Valor Final	No Puntos Evaluar
Variables			
e	10		
r	10		
il	10^{-6}		
Parámetros			
k	0.1		
s	6		
ke	0.01		
kr	1		
iα	100		
fe	10^{-5}		
pe	21		
αe	1		
fr	10^{-5}		
pr	14		
αr	1		
fi	0		
kpi	100	1000	100
kdi	10		
ki	500	5000	100

Tabla 4.7 Valores de Condiciones Iniciales.

Tiempo Inicial (ti)	Tiempo Final (tf)	Tolerancia Absoluta (ta)	Tolerancia Relativa (tr)
0	10	10^{-6}	0.0010

Tabla 4.8 Valores de Tiempo y Tolerancia.

Para estos valores de parámetros y variables se generan 10 201 simulaciones, cada una de estas simulaciones debe integrarse en el intervalo de tiempo especificado y devolver un resultado por cada una de las variables (un resultado para la célula defectora “e”, un resultado para la célula reguladora “r” y otro resultado para la interleuquina “il”), lo que indica que para una simulación deben devolverse 3216 resultados, 32 806 416 resultados en total.

Resultados de las pruebas:

Tipo de Simulación	No. Máquinas Clientes	Cantidad de simulaciones realizadas	Tiempo de Cálculo (días:horas:min)
Local	1	10 201	2:15:7
Distribuida	475	10 201	0:2:24

Tabla 4.9 Resultados

Para las simulaciones distribuidas la plataforma contaba con 475 máquinas clientes, pero debe señalarse que siempre existen fallos de estas, ya que no están dedicadas únicamente al cálculo, por lo que en ocasiones no concluyen las simulaciones enviadas o no hacen peticiones al servidor. Por tanto no se puede afirmar que se hayan utilizado todas las máquinas clientes en el proceso de pruebas.
Con los resultados arrojados se evidencia la superioridad de utilizar la plataforma de cálculo distribuido t-arenal para resolver este tipo de problemas, ya que mejoró el tiempo de cálculo para resolver el mismo modelo matemático utilizando aproximadamente 2 días y 13 horas menos que de forma local.

4.5 Conclusiones del capítulo

Se describieron cómo los elementos del modelo de diseño fueron implementados en términos de componentes, se realizó el diagrama de componentes, el diagrama de despliegue detallado, fueron explicados los principales métodos de la aplicación y se compararon los resultados de simulaciones locales y simulaciones distribuidas.

Conclusiones

Con el desarrollo de este trabajo se alcanzó satisfactoriamente el objetivo propuesto, pues se desarrolló una aplicación que permite realizar simulaciones distribuidas de sistemas biológicos que son modelados mediante sistemas de ecuaciones diferenciales utilizando la plataforma de cálculo distribuido desplegado en la UCI: t-arenal.

- Se realizó el análisis y diseño de todas las clases lo cual facilitó una correcta implementación.
- Se realizó la implementación del módulo de Simulación del proyecto BioSyS, la cual satisface los requerimientos especificados por el cliente.

Recomendaciones

Al culminar este trabajo nuestro equipo recomienda:

- La realización de pruebas de caja negra al módulo para comprobar las funcionalidades del sistema.
- Estudiar formas de integración de la herramienta matemática Octave con el lenguaje de programación Java, ya que dicha herramienta fue desarrollada por la comunidad de software libre a diferencia de MatLab.
- Implementar la Ayuda de la aplicación con el objetivo de informarle al usuario cómo trabajar con ella y de mostrar información importante referente a los sistemas biológicos.

Referencias Bibliográficas

1. **Costa Portela, Carlos**. *Informática y Simulación Biológica*. [En línea] http://www.biologia.org

2. Univesidad de Veracruz. *La simulación computarizada.* [En línea] http://www.public.iastate.edu.

3. **Cortés, Josefina Barrera.** *Modelado de procesos biológicos mediante técnicas de inteligencia Artificial.*

4. Bioinformáticos. *Biomatemática.* [En línea] http://www.bioinformaticos.com.ar.

5. **Rabatte Suárez, Ivonne, Blázquez Morales, Sobeida Leticia y Contreras Costeño, Diego.** Universidad Central de Venezuela. *Ecuaciones diferenciales aplicadas al área de Ciencias de la Salud.* [En línea] http://www.ciens.ucv.ve.

6. **González Mulet, Yunet y Alonso Delgado, Yanet.** *Software para la Simulación de Sistemas Biológicos: Módulo de Simulación y Análisis.* Ciudad de la Habana : UCI, 2007.

7. **Moreno Lemus, Noel.** *BioSyS: Software para la simulación y análisis de sistemas biológicos.* Ciudad de la Habana : UCI, 2007.

8. **Longendri Aguilera Mendoza.** *Sistema de Computo Distribuido aplicados a la bioinformatica*. Ciudad de la Habana: UCI, 2008.

9. **Santinelli, Mariano y Andre, Gustavo.** *¿Qué es la computación Grid?* : Universidad Nacional de Luján.

10. **Keane, T.** *Java Distributed System:Developer Manual*: Department of Computer Science, National University of Ireland, 2003.

11. **Grady, Booch, Ivar, Jacobson y James, Rumbaugh.** *El Proceso Unificado de Desarrollo de Software.* 2000.

12. **Newkirk, J.** *"La programación extrema en la práctica".* Madrid: Pearson Education, 2002.

13. **Stephens, M., Rosenberg,D.** *"Extreme programming refactored: the case against X.P".* California : Berkeley, 2003.

14. Sitio de Metodología de Desarrollo. Ayuda extendida de OpenUp. [En línea] http://10.34.20.5:5800/OpenUP.

15. **Ferré Grau, Xavier y Sánchez Segura, María Isabel.** Tutorial UML, Desarrollo orientado a objetos con UML. [En línea] 2004. http://www.clikear.com.

16. **Jacobson, Ivar y Booch, Grady.** *El Proceso Unificado de Desarrollo de Software.* 1999. volumen 1..

17. Visual Paradigm. [En línea] http://www.visual-paradigm.com/product/vpuml.

18. **Moreno Lemus, Noel.** *BioSyS: Software para la simulación y análisis de sistemas biológicos.* s.l. : UCI, 2007.

19. **Bermejo, Laura y Gómez, Enrique.** *Eclipse como IDE.*

20. NetBeans. [En línea] http://www.netbeans.org.

21. *Persistencia relacional para Java y .Net*. [En línea] http://www.hibernate.org

22. Sitio de Metodología de Desarrollo. Ayuda extendida de OpenUp. [En línea] 01 de 2008. http://10.34.20.5:5800/OpenUP.

23. **Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* 2004.

24. **Cardeso, Fabio, Núñez, Gabriel y Camacho, Erica.** *Arquitecturas de Software, Guías de Estudio.* 2004.

25. **Manuel Lagos Torres.** Introducción al diseño con patrones. [En línea] http://www.elrincondelprogramador.com.

26. **Visconti, Marcello y Astudillo, Hernán.** *Fundamentos de Ingeniería de Software, Patrones de Diseño,.*

27. **Sitio de la Asignatura Ingeniería de Software 1**. *Conferencia 3_ FT Requerimientos.* [En línea] http://teleformacion.uci.cu.

28. **Sitio de la Asignatura Ingeniería de Software 2**. *Material de Apoyo Conferencia Diseño.* [En línea] http://teleformacion.uci.cu.

29. **Sitio de la Asignatura Ingeniería de Software 2.** *Conferencia4. FT Implementación.* [En línea] http://teleformacion.uci.cu.

Bibliografía

1. Costa Portela, Carlos. Informática y Simulación Biológica. [En línea] http://www.biologia.org

2. Univesidad de Veracruz. La simulación computarizada. [En línea] http://www.public.iastate.edu.

3. Cortés, Josefina Barrera. Modelado de procesos biológicos mediante técnicas de inteligencia Artificial.

4. Bioinformáticos. Biomatemática. [En línea] http://www.bioinformaticos.com.ar.

5. Rabatte Suárez, Ivonne, Blázquez Morales, Sobeida Leticia y Contreras Costeño, Diego. Universidad Central de Venezuela. Ecuaciones diferenciales aplicadas al área de Ciencias de la Salud. [En línea] http://www.ciens.ucv.ve.

6. González Mulet, Yunet y Alonso Delgado, Yanet. Software para la Simulación de Sistemas Biológicos: Módulo de Simulación y Análisis. Ciudad de la Habana : UCI, 2007.

7. Moreno Lemus, Noel. BioSyS: Software para la simulación y análisis de sistemas biológicos. Ciudad de la Habana : UCI, 2007.

8. Longendri Aguilera Mendoza. Sistema de Computo Distribuido aplicados a la bioinformatica. Ciudad de la Habana: UCI, 2008.

9. Santinelli, Mariano y Andre, Gustavo. ¿Qué es la computación Grid? : Universidad Nacional de Luján.

10. Keane, T. Java Distributed System:Developer Manual: Department of Computer Science, National University of Ireland, 2003.

11. Grady, Booch, Ivar, Jacobson y James, Rumbaugh. El Proceso Unificado de Desarrollo de Software. 2000.

12. Newkirk, J. “La programación extrema en la práctica”. Madrid: Pearson Education, 2002.

13. Stephens, M., Rosenberg,D. “Extreme programming refactored: the case against X.P”. California : Berkeley, 2003.

14. Sitio de Metodología de Desarrollo. Ayuda extendida de OpenUp. [En línea] http://10.34.20.5:5800/OpenUP.

15. Ferré Grau, Xavier y Sánchez Segura, María Isabel. Tutorial UML, Desarrollo orientado a objetos con UML. [En línea] 2004. http://www.clikear.com.

16. Jacobson, Ivar y Booch, Grady. El Proceso Unificado de Desarrollo de Software. 1999. volumen 1..

17. Visual Paradigm. [En línea] http://www.visual-paradigm.com/product/vpuml.

18. Moreno Lemus, Noel. BioSyS: Software para la simulación y análisis de sistemas biológicos. s.l. : UCI, 2007.

19. Bermejo, Laura y Gómez, Enrique. Eclipse como IDE.

20. NetBeans. [En línea] http://www.netbeans.org.

21. Persistencia relacional para Java y .Net. [En línea] http://www.hibernate.org

22. Sitio de Metodología de Desarrollo. Ayuda extendida de OpenUp. [En línea] 01 de 2008. http://10.34.20.5:5800/OpenUP.

23. Reynoso, Carlos y Kiccillof, Nicolás. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. 2004.

24. Cardeso, Fabio, Núñez, Gabriel y Camacho, Erica. Arquitecturas de Software, Guías de Estudio. 2004.

25. Manuel Lagos Torres. Introducción al diseño con patrones. [En línea] http://www.elrincondelprogramador.com.

26. Visconti, Marcello y Astudillo, Hernán. Fundamentos de Ingeniería de Software, Patrones de Diseño,.

27. Sitio de la Asignatura Ingeniería de Software 1. Conferencia 3_ FT Requerimientos. [En línea] http://teleformacion.uci.cu.

28. Sitio de la Asignatura Ingeniería de Software 2. Material de Apoyo Conferencia Diseño. [En línea] http://teleformacion.uci.cu.

29. Sitio de la Asignatura Ingeniería de Software 2. Conferencia4. FT Implementación. [En línea] http://teleformacion.uci.cu.

30. Pressman, Roges S, 2005. Ingeniería de Software un Enfoque Práctico. 2005, 5ta edición, 589 páginas, [En línea] http://biblioteca.uci.cu/bives/titdigitales.

31. Manuel Álvarez Blanco; Arnaldo Gómez Montenegro. Matemática numérica. 2003.

Anexos

Anexo 1 Descripción de Patrones de Diseño

A continuación se describen los patrones de diseño que fueron utilizados durante el desarrollo de la aplicación y se evidencia su utilización.

1.1 Patrón Fachada

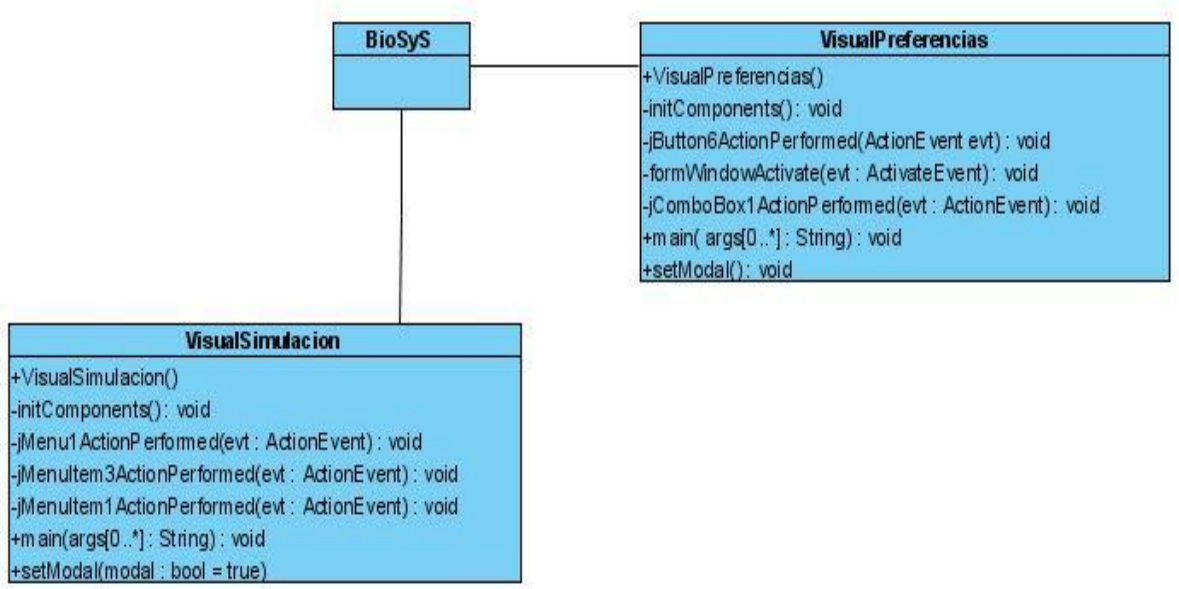
Patrón:	Fachada
Descripción:	Proporciona una interfaz unificada para un conjunto de interfaces en este caso *BioSyS* se encarga de facilitar el acceso a otras interfaces como son *VisualSimulacion* y *VisualPreferencia.*
Imagen Ejemplo	
	

Tabla Anexo 1. Patrón Fachada.

1.2 Patrón Estratégico

Patrón:	Estratégico
Descripción :	Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución en este caso tenemos que la clase *BSMatLab* cuenta con cuatro constructores en los cuales varían los

	parámetros y se debe escoger en tiempo de ejecución cual de ellos utilizar, ya que los mismos están implementados para generar la información que no haya sido especificada por el cliente, de igual forma tenemos la clase BSSimulacion que tiene tres métodos *Simular()* con parámetros distintos.
Imagen Ejemplo	

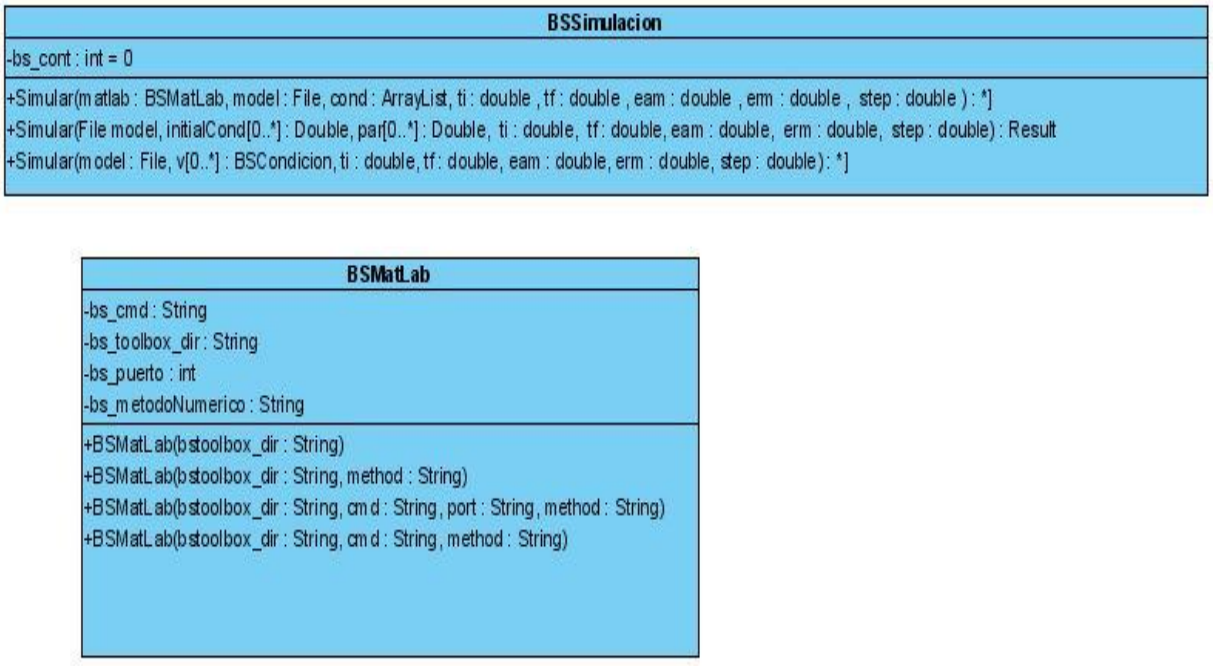


Tabla Anexo 1. Patrón Estratégico.

1.3 Patrón Observador

Patrón:	Observador
Descripción :	Aplicamos este patrón para mostrar el estado del progreso de la simulación. Cuando cambia el estado del progreso de la simulación se actualiza automáticamente en la interfaz *VisualProgreso.*
Imagen Ejemplo	

Tabla Anexo 1. Patrón Observador.

1.4 Patrón Experto

Patrón:	Experto
Descripción:	Al encontrarse la información encapsulada en clases, cada una es responsable de manejar el acceso a esa información. Ejemplo las clases *BSCondicion* que maneja toda la información referente a las condiciones iniciales y la clase *BSResultado* que contiene la información de los resultados de las simulaciones.
Imagen Ejemplo	

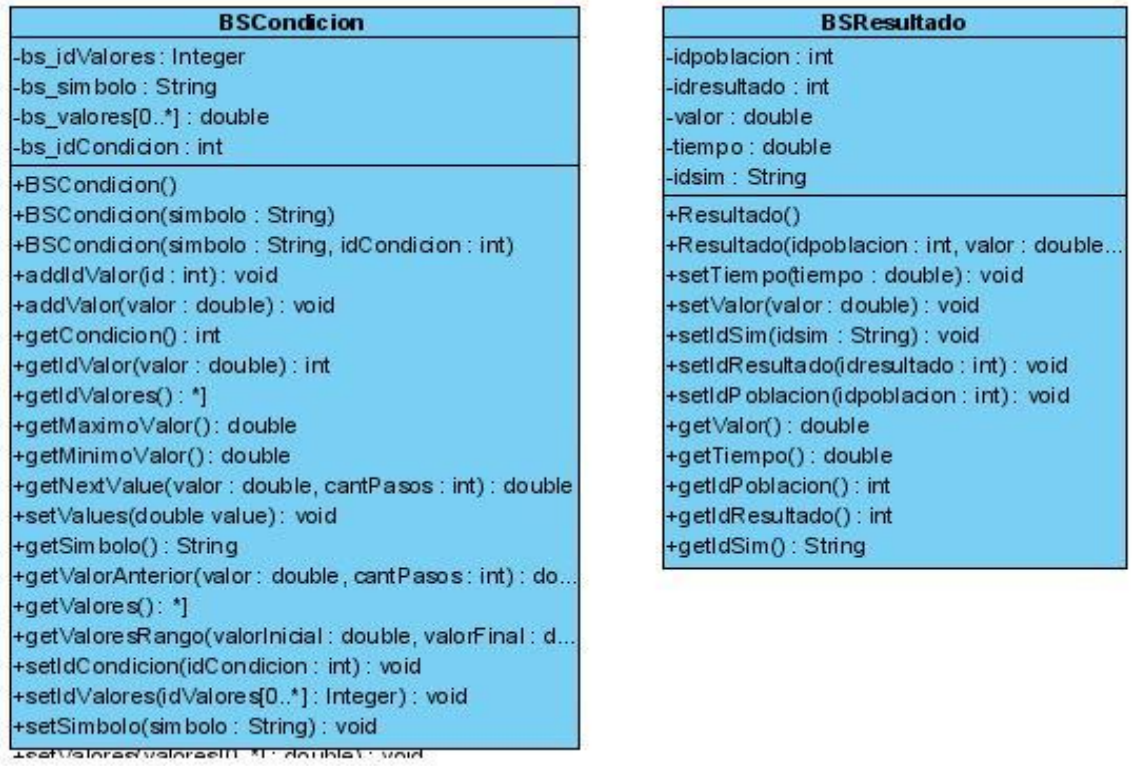


Tabla Anexo 1. Patrón Experto.

1.5 Patrón Creador

Patrón:	Creador
Descripción :	Para acceder a la información encapsulada en cada clase es necesario crear objetos de las mismas, y la encargada de crear dichos objetos es la propia clase. Ejemplo la clase *BSPreferencia* necesita crear un objeto de la clase *BSMatLab* para acceder a la información de esta.
Imagen Ejemplo	

Tabla Anexo 1. Patrón Creador.

1.6 Patrón Creador

Patrón:	Controlador
Descripción:	Siguiendo este patrón se implementaron las clases *SimulacionControl. PreferenciasControl* y *BioSySControl* que se encargan de la gestión de información entre las clases interfaces y las clases que las implementan.
Imagen Ejemplo	

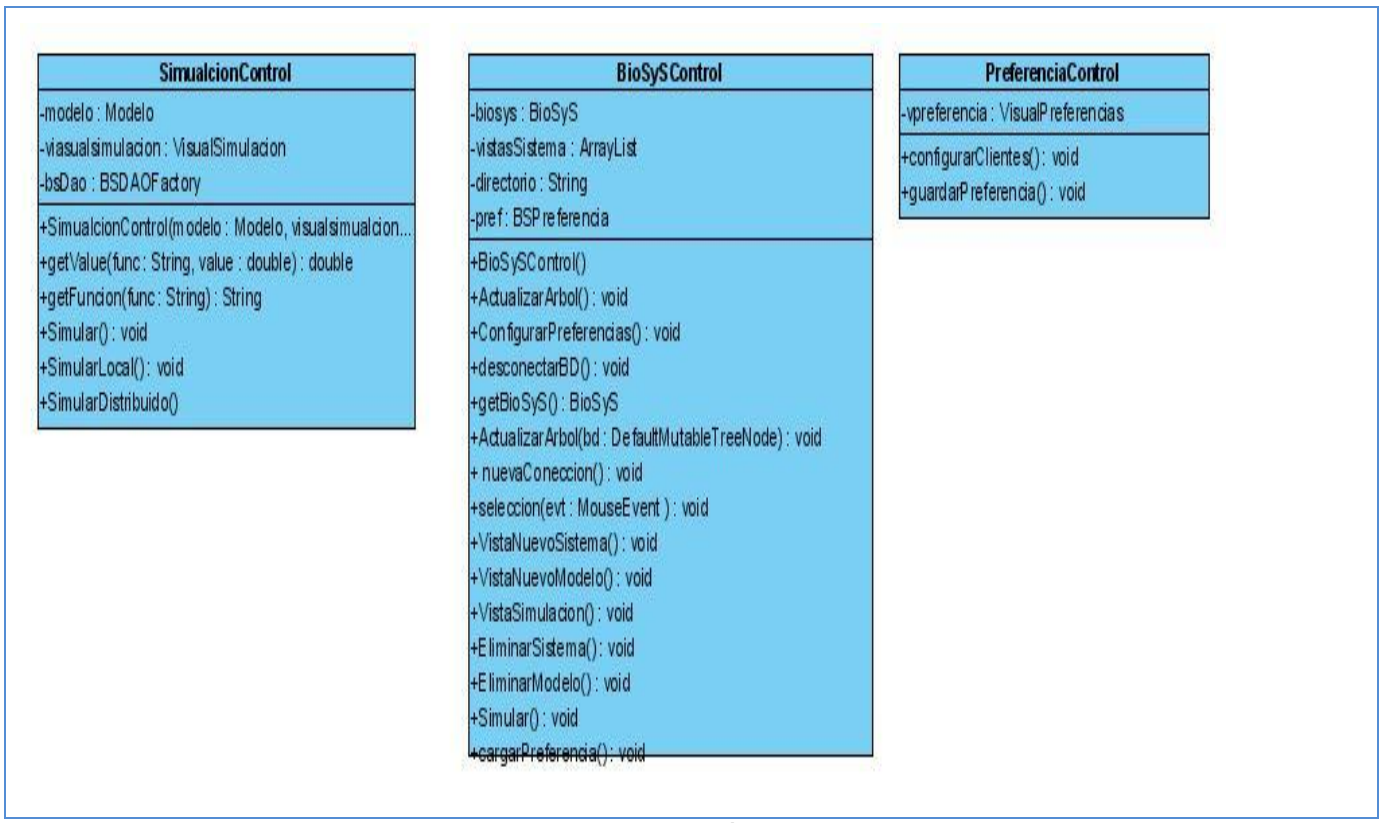
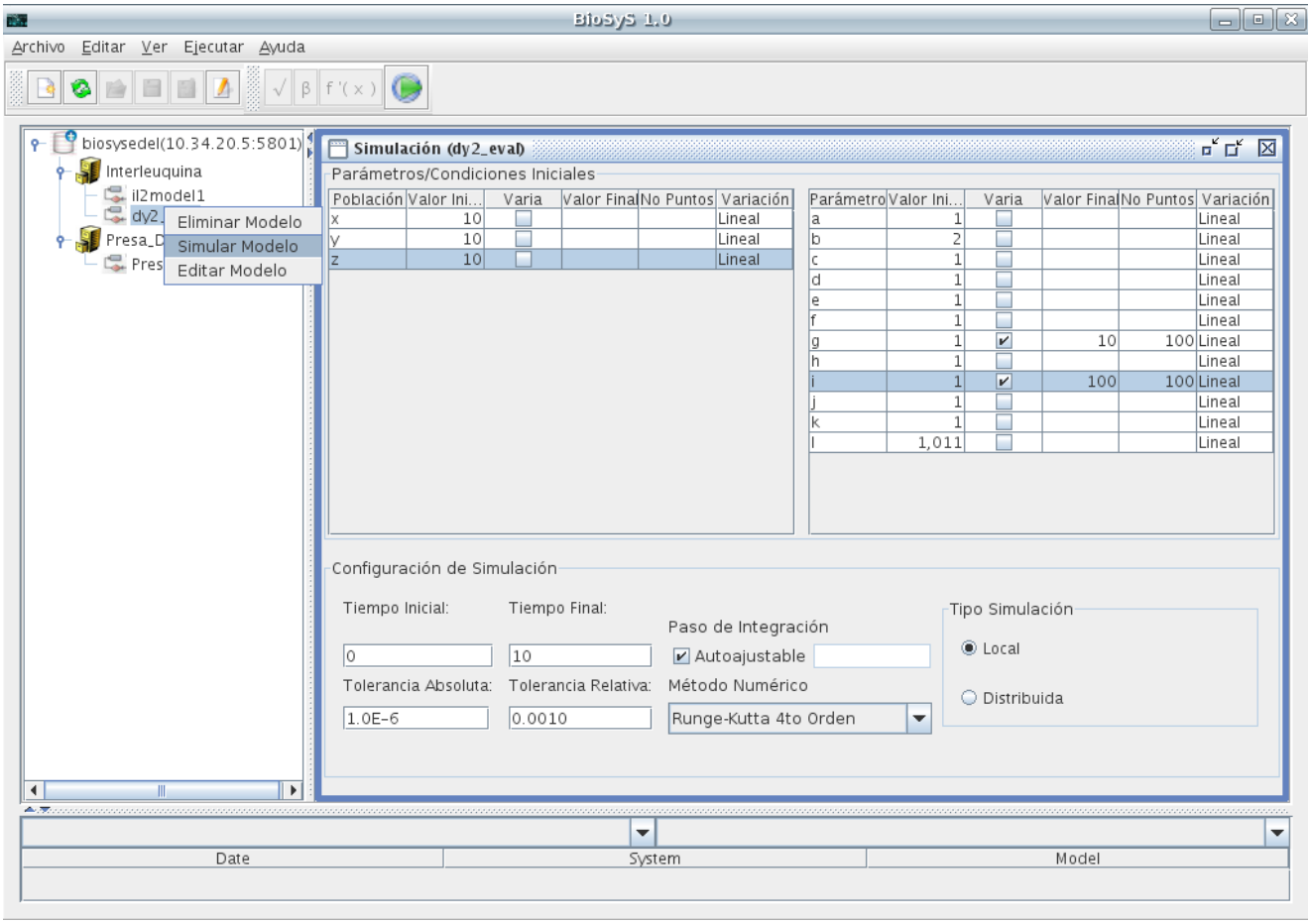


Tabla Anexo 1. Patrón Controlador.

Anexo 2 Interfaz Visual

Interfaz Principal donde se definen los valores de las condiciones iniciales para las simulaciones a partir de la opción Simular Modelo.



Interfaz Preferencias donde se definen las preferencias del sistema y la configuración de la conexión al servidor t-arenal.


Preferencias
Preferencias
Grid
Herramienta Matemática:
OdeToJava
Matlab
OdeToJava
Ejecutable de Matlab:
...
Puerto Socket:
1024
Directorio Temporal:
del/NetBeansProjects/BioSyS 1.0/temp
...
Aceptar
Cancelar


Preferencias
Preferencias
Grid
Servidor:
jdcs4.uci.cu
Puerto Socket:
5801
Puerto RMI:
5800
Clientes
Aceptar
Cancelar

Glosario de Términos

Middleware: Es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas.

Problema de Cauchy: También llamado problema de valor inicial o PVI consiste en resolver una ecuación diferencial sujeta a unas ciertas condiciones iniciales sobre la solución cuando una de las variables que la definen (usualmente, la variable temporal), toma un determinado valor (generalmente, t=0, para modelar las condiciones del sistema en el instante inicial)

Plugins: Es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver (controlador) en una aplicación, para hacer así funcionar un dispositivo en otro programa.

RMI: Es un mecanismo ofrecido en Java para invocar un método remotamente.

Single System Image: En computación distribuida, un grupo de SSI es un sistema que en múltiples redes, bases de datos distribuidas o servidores aparecerá al usuario como un único sistema.

SSH: (Secure SHell) es el nombre de un protocolo y del programa que lo implementa, y sirve para acceder a máquinas remotas a través de una red. Permite manejar por completo la computadora mediante un intérprete de comandos.

Strut: Es un framework de la capa de presentación que implementa el patrón MVC en Java.

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