UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

Título:"BIOSYS: Adición de nuevos métodos de simulación y utilización e integración del asistente matemático Octave"

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

Autores:

Jorge Díaz del Toro

Aleida Perera Alba

Tutores:

Msc. Noel Moreno Lemus.

Lic. Félix Argelio Martínez Nariño.

Junio, 2009

"Es increíble que la matemática, habiendo sido creada por la mente humana, logre describir la naturaleza con tanta precisión"

Albert Einstein

Declaración de Autoría

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

Para que así conste firmo la presente a los __ días del mes __ del año 2009.

Aleida Perera Alba

_____

Firma del Autor

Jorge Díaz del Toro

_____

Firma del Autor

Noel Moreno Lemus

_____

Firma del Tutor

Félix Argelio Martínez Nariño

_____

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.

Félix Argelio Martínez Nariño: Licenciado en Lenguas Extranjeras.

Correo electrónico: famartinez@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Aleida Perera Alba: Estudiante 5to Curso Ing. Informática

Correo electrónico: aperera@estudiantes.uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Jorge Díaz del Toro: Estudiante 5to Curso Ing. Informática

Correo electrónico: jddeltoro@estudiantes.uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Agradecimientos

Un especial agradecimiento a todos nuestros compañeros de proyecto, tanto a los de BioSyS como a los de Graphtool, por sus sabios consejos y opiniones acerca de nuestro trabajo, sobre todo cuando afrentábamos momentos difíciles.

A nuestro tutor Noel por sus ideas y representar una guía a la hora de tomar decisiones.

A todos los profesores que de una forma u otra contribuyeron en nuestra formación profesional, durante estos cinco años de duros estudios.

Y a todos aquellos que se interesaron en conocer como avanzaba nuestro trabajo y nos brindaron incondicionalmente su ayuda.

Dedicatoria

A mi mamá, por ser capaz de comprenderme con solo una mirada, y bridarme su apoyo en todo momento, por aliviar mis penas y ayudarme a solucionar mis mayores problemas cuando los enfrento, por darme la vida y amarme como soy, a ti vieja dedico este trabajo.

A mi papá por ser el principal impulsor de mis sueños, por inspirarme ese espíritu creativo y de constancia, por ser un ejemplo a seguir como persona, como padre y como hombre, por mostrarme que cada día podemos ser mejores y que todos los obstáculos son superables.

A mi familia por su constante preocupación, por ser mi soporte y brindarme tantas alegrías, por confiar siempre en mí y por no defraudarme en nada.

A Aleida que es mi fuente de energía diaria, por darme su ayuda ilimitada, por su responsabilidad, y por soportar mis momentos más difíciles.

A todos mis amigos de la universidad que más que amigos ya son familia, y que nunca podre olvidar, a ellos que también son merecedores de estos resultados.

Jorge

Dedicatoria

A mi mamá por ser símbolo de amor, dedicación, comprensión, preocupación por todo cuanto me rodeaba y ayuda infinita en todos los momentos.

A mi papá por ser no solo un padre incondicional sino la guía intelectual, el apoyo en los estudios, el ejemplo a seguir y por ser el máximo contribuyente a que hoy sea la persona que soy: una ingeniera.

A mis abuelos por darme una crianza tan llena de amor y comprensión.

A mi hermano, mis tíos y primas, especialmente a mi tía Marlene por haberme dado tanto apoyo y cariño en estos cinco años de la carrera.

A mi novio Jorge por haber compartido la experiencia de realizar un trabajo de diploma juntos y brindarme tanto amor y ayuda en el transcurso de estos años de formación.

A mis amigos de la universidad por los momentos alegres que pasamos y a los que nunca olvidaré, especialmente a Yanet, Veylys, Garnache y Denlis.

Aleida

Resumen

En el mundo de la biotecnología y la biología de sistemas, existen grandes inconvenientes a la hora de desarrollar y trabajar con entes biológicos, comprender como funcionan y como se desarrollan bajo ciertas condiciones, es una tarea que exige mucho conocimiento, materiales económicos y tiempo.

De hecho las investigaciones toman años y en el mejor de los casos los resultados no son altamente confiables por desarrollarse bajo condiciones variables en el transcurso de mucho tiempo. La creación de software que simule este comportamiento ha acompañado a la informática, específicamente a las ciencias de la computación, desde el momento mismo en que se comenzó a contar con grandes capacidades de cálculo y ordenadores de gran procesamiento.

Cuba ha sido líder durante cierto tiempo en el mercado internacional de productos farmacéuticos, derivados de estos estudios, debido al alto desarrollo de la medicina y la biotecnología que se ha alcanzado. Sin embargo, a pesar del avance logrado en esta rama de la economía y la ciencia, los científicos están maniatados por no contar con herramientas informáticas que les faciliten el proceso de investigación. Un importante acuerdo entre el Centro de Ingeniería Genética y Biotecnología y la Universidad de las Ciencias Informáticas creó por primera vez en Cuba un proyecto que se dedicara a la creación de un simulador de sistemas biológicos, tarea de vital importancia para el país y la comunidad científica nacional e internacional.

Hasta las fechas actuales se han desarrollado variedades de software a nivel mundial que simulan o intentan simular ciertos comportamientos de estos sistemas o entidades biológicas, la mayoría de ellos son demasiado específicos, orientados a tareas muy simples, otros de gran calidad no son comercializados, o no cumplen con funcionalidades deseadas. Por tanto no hay una opción viable en el mercado para adoptar, o no existe una herramienta lo suficientemente poderosa como para evitar que este proyecto tenga que llevarse a cabo.

El proyecto BioSyS ha presentando hasta el momento actual la versión 1.0 del producto, el cual brinda una herramienta que permite simular computacionalmente sistemas biológicos descritos por Sistemas de Ecuaciones Diferenciales, utilizando para dar solución a tales sistemas una infraestructura de cálculo distribuido, en la cual se ven envueltas un grupo de computadoras preparadas en el polo productivo para realizar esta función y apoyándose en herramientas de cálculo matemático ajenas a BioSyS.

Resumen

El presente trabajo ampliará los horizontes del simulador BioSyS, potenciando sus posibilidades, su diversidad y su eficiencia.

Índice

Introducción

La Biología de Sistemas es una de las ramas de la biomedicina que más aportes puede hacer y de hecho hace a la medicina en general, su carácter experimentador e innovador le permite desentrañar los misterios de la vida. Sin embargo el funcionamiento de un organismo, así como su desarrollo bajo ciertas condiciones no es algo simple. De hecho la simulación de sistemas biológicos es algo tan complejo, que sin el uso de sistemas computacionales con altos niveles de cálculo, sería imposible resolver. El uso de herramientas informáticas ha ayudado desde la existencia misma de ellas al hombre a simplificar su vida y más aun a resolver problemas de forma rápida y eficiente. La biología de sistemas no está exenta a este fenómeno y por tanto su apoyo en la rama de la informática es vital, para el descubrimiento de nuevos fenómenos biológicos y sus relaciones con la vida y la salud del hombre.

Los sistemas biológicos pueden expresarse matemáticamente de diferentes formas, además podemos agruparlos para realizar análisis desde la forma más simple una molécula, hasta congregaciones complejas, en las que interactúa una diversidad de individuos conocida como ecosistemas.

La condición de un sistema biológico, de ser expresado matemáticamente, permite a su vez resolverlo de manera matemática y para ello existen disímiles herramientas, la versión 1.0 de BioSyS usó los SED o Sistemas de Ecuaciones Diferenciales, debido a que brindan facilidades de modelación y simulación. Estas ecuaciones en la aplicación se resuelven haciendo uso de unas librerías llamadas ODEtoJava o mediante un software que incorpora una gran cantidad de funcionalidades matemáticas llamado MatLab. Debido a que tanto las librerías utilizadas como el software matemático, solo pueden resolver un Sistema de Ecuaciones Diferenciales a la vez, se ha utilizado también un sistema distribuido conocido como T-arenal, que distribuye de manera eficiente los complejos cálculos en una infraestructura de computadoras.

La conexión de BioSyS al MatLab, a la infraestructura Grid y el uso del paquete ODEtoJava se realiza a través de funciones previamente programadas y que serán objeto de estudio en este trabajo de diploma. No obstante las soluciones previstas por BioSyS no se han realizado bajo diferentes entornos, por tanto su conjunto de soluciones se halla comprendido por una sola herramienta matemática.

Además el MatLab es un software que se distribuye bajo licencia propietaria, la misma tiene un valor cerca de los 10.000 USD, por tanto si el sistema dependiera exclusivamente de este asistente, sería

difícil que pequeñas empresas o centros de investigación con recursos económicos limitados pudieran hacer uso del software propuesto.

La idea de poseer diferentes vías que permitan solucionar un Sistema de Ecuaciones Diferenciales ha sido un objetivo básico para el proyecto, es por lo que el presente trabajo de diploma se propone lograr una interactividad entre BioSyS y el asistente matemático Octave, además adecuar e implementar otras funciones de las librerías ODEtoJava, esto posibilitaría al investigador ampliar su espectro de soluciones y realizar adecuadas comparaciones entre datos de una misma simulación usando entornos y herramientas diferentes.

Por tanto la solución derivada de estas incógnitas representaría el **Problema a Resolver** del presente trabajo, expresado en otras palabras de la siguiente manera: ¿Como proveer al simulador BioSyS de otras formas de simulación?

Para resolver este problema se define como **Objeto de Estudio**: La Simulación de Sistemas Biológicos y como **Campo de Acción** dentro de un área tan amplia: La Simulación de Sistemas Biológicos descritos por SED.

Como **Objetivo General**: Implementar un módulo que permita resolver Sistemas de Ecuaciones Diferenciales utilizando el asistente matemático Octave y rediseñar funciones matemáticas que forman parte de las librerías ODEtoJava.

Los **Objetivos Específicos** a partir del desarrollo de estas tareas son:

1- Realizar el análisis de un módulo de simulación.
2- Diseñar dicho módulo.
3- Implementar funcionalidades.
4- Incorporar funcionalidades al Simulador de Sistemas Biológicos BioSyS.

Los anteriores Objetivos tanto Generales como Específicos inducen varias **Tareas de la Investigación**:

1-Estudio de los algoritmos de simulación programados en BioSyS.

2-Determinación de tipos de simulaciones que se realizarán con las nuevas funcionalidades a implementar.

3-Realización de un estudio del asistente matemático Octave y su funcionamiento.

4-Análisis de las librerías ODEtoJava.

5-Estudio del funcionamiento del sistema T-arenal y su compatibilidad con el asistente matemático Octave.

6-Diseño de las clases necesarias para realizar estas simulaciones.

7-Implementación de cada una de las clases y sus funcionalidades.

Este Trabajo está conformado por 4 capítulos y la información en ellos está distribuida de la siguiente forma:

Capitulo 1: Fundamentación teórica

En este capítulo se mencionarán cada uno de los elementos que estarán incorporados en este trabajo de tesis, como son los Modelos Matemáticos, Ecuaciones Diferenciales, Métodos de Resolución, Asistente Matemático, Metodología de Desarrollo, Lenguaje de Programación, entre otros, así como sus características específicas y ventajas que brindan.

Capitulo 2: Características del Sistema

Este capítulo estará dedicado a realizar una descripción completa de las características del sistema que se debe desarrollar. De manera que garantice una visión global de cómo se constituye el sistema y qué funciones este debe cumplimentar.

Capitulo 3: Análisis y Diseño del Sistema

Este capítulo es el encargado de mostrar todo lo relacionado con la arquitectura del sistema, los paquetes y subsistemas definidos, los patrones de desarrollo que se usaron, así como los diagramas de Clases del Diseño e Interacción que se construyeron.

Capitulo 4: Implementación del Sistema

En este capítulo se realiza el análisis de la implementación del sistema, se describen cada uno de los componentes, se muestran las partes esenciales y definitivas del código fuente y por último se realizan pruebas de fiabilidad que garanticen un correcto funcionamiento de las nuevas funcionalidades.

Capitulo 1: Fundamentación Teórica

1.1 Introducción al Capítulo

En este capítulo se explicará qué es la simulación de un ente o sistema biológico, qué es un modelo matemático y cuáles son los modelos matemáticos asociados a los sistemas biológicos, así como qué métodos se usan para resolver estos sistemas y las herramientas matemáticas e informáticas que permiten darle solución. Se explicará qué es una infraestructura de cálculo distribuido y se hará énfasis en una específica que se usa actualmente para el correcto funcionamiento del sistema. Se mencionan además los algoritmos computacionales, las herramientas de desarrollo, la metodología de ingeniería a utilizar, artefactos que se generan a partir de estos y los roles que estarán presentes en el desarrollo de este modulo.

1.2 ¿Qué es una Simulación de un Sistema Biológico?

Responder esta pregunta primeramente requeriría definir el concepto de simulación, veamos algunas acepciones para el término:

La simulación es reproducir el ambiente, las variables (rasgos, apariencia, características, contexto) de un sistema real. Es imitar una situación del mundo real en forma matemática.

La simulación constituye una técnica económica que nos permite ofrecer varios escenarios posibles de una situación y nos permite equivocarnos sin provocar efectos sobre el mundo real (por ejemplo un simulador de vuelo o conducción) (1).

-Definición Estricta (H. Maisel y G. Gnugnoli)

Simulación es una técnica numérica para realizar experimentos en una computadora digital. Estos experimentos involucran ciertos tipos de modelos matemáticos y lógicos que describen el comportamiento de sistemas de negocios, económicos, sociales, biológicos, físicos o químicos a través de largos periodos de tiempo (1).

-Otra Definición (Robert E. Shannon)

Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema (1).

Por tanto, una simulación de un sistema biológico sería el proceso mediante el cual utilizamos su descripción matemática y la resolvemos haciendo variar sus parámetros hasta que describan un entorno determinado. Sin embargo, quizás sea la simulación de sistemas biológicos las más compleja de todas, debido a que la naturaleza misma de la biología es en extremo compleja.

Sin embargo, el resultado es tan valioso, que el esfuerzo por desentrañar estos problemas es válido, inclusive mucho más cuando vidas humanas dependen de estas soluciones.

1.3 Modelación de Sistemas Biológicos

La medicina moderna ha evolucionado de una manera vertiginosa en los últimos años, y la principal causa de ello ha sido los acontecimientos derivados de la revolución del DNA, que han permitido esclarecer cuales son, por ejemplo, los genes implicados en enfermedades monogénicas o los genes que confieren susceptibilidad en las enfermedades complejas.

Sin embargo, los esfuerzos se han concentrado en caracterizar por separado los componentes celulares (genes, proteínas y metabolitos), no considerando que muchos de los procesos llevados a cabo por la célula, exhiben complejidad, razón por la cual es necesaria una transición del enfoque reduccionista actual a una visión sistémica donde se incluyan cada uno de los componentes y sus interacciones.

Desde ésta visión, las aproximaciones al tratamiento de las enfermedades estarán orientadas por un conocimiento de los procesos biológicos más cercano a la realidad, que será obtenido de la integración de los datos adquiridos experimentalmente mediante sistemas que realizan mediciones a gran escala. Posteriormente, con estos datos se podrán formular modelos matemáticos con sus respectivas variables, parámetros y constantes, los cuales serán sometidos a simulación utilizando herramientas computacionales (2).

Lograr un correcto modelado del sistema biológico es de vital importancia para la investigación, debido a que los resultados que se obtengan dependerán directamente del modelo. La creación del mismo será compleja y requerirá un alto nivel de investigación y de conocimientos tanto matemáticos como biológicos sin embargo el mismo será la clave para descifrar comportamientos y descubrir pistas del sistema modelado.

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 (3).

1.4 Modelos Matemáticos

En ciencias aplicadas un **Modelo matemático** es uno de los tipos de modelos científicos, que emplea algún tipo de formulismo matemático para expresar relaciones, proposiciones sustantivas de hechos, variables, parámetros, entidades y relaciones entre variables y/o entidades u operaciones, para estudiar comportamientos de sistemas complejos ante situaciones difíciles de observar en la realidad (4).

1.4.1 Clasificaciones de Modelos

Se podría decir que un modelo de las ciencias físicas es una traducción de la realidad física para poder aplicar los instrumentos y técnicas de las teorías matemáticas para estudiar el comportamiento de sistemas complejos, y posteriormente hacer el camino inverso para traducir los resultados numéricos a la realidad física. Generalmente se introducen simplificaciones de realidad.

Los modelos matemáticos pueden clasificarse de la siguiente manera:

- **Determinista**. Se conoce de manera puntual la forma del resultado ya que no hay incertidumbre. Además, los datos utilizados para alimentar el modelo son completamente conocidos y determinados.
- **Estocástico**. Probabilístico, que no se conoce el resultado esperado, sino su probabilidad y existe por tanto incertidumbre.

Además con respecto a la función del origen de la información utilizada para construirlos los modelos pueden clasificarse de otras formas. Podemos distinguir entre modelos heurísticos y modelos empíricos:

- **Modelos heurísticos** (del griego *euriskein* 'hallar, inventar'). Son los que están basados en las explicaciones sobre las causas o mecanismos naturales que dan lugar al fenómeno estudiado.

- **Modelos empíricos** (del griego *empeirikos* relativo a la 'experiencia'). Son los que utilizan las observaciones directas o los resultados de experimentos del fenómeno estudiado.

Además los modelos matemáticos encuentran distintas denominaciones en sus diversas aplicaciones. Los siguientes son algunos tipos en los que se puede adecuar algún modelo matemático de interés. Según su campo de aplicación los modelos:

- **Modelos conceptuales**. Son los que reproducen mediante fórmulas y algoritmos matemáticos más o menos complejos los procesos físicos que se producen en la naturaleza
- **Modelo matemático de optimización**. Los modelos matemáticos de optimización son ampliamente utilizados en diversas ramas de la ingeniería para resolver problemas que por su naturaleza son indeterminados, es decir presentan más de una solución posible.

1.4.2 Representación del Modelo

La representación puede ser de la siguiente manera:

- **Conceptual**. Por una descripción cualitativa bien organizada que permite la medición de sus factores.
- **Matemático**. Se refiere a una representación numérica por aspectos lógicos y estructurados con aspectos de la ciencia matemática.

Pueden ser números, letras, imágenes, símbolos. Por ejemplo si se refiere a un modelo gráfico de matemáticas, se observan imágenes y gráficas matemáticas, que representan a un modelo numérico y de ecuaciones, los cuales son expresiones visuales basadas en aspectos cuantificables y de la ciencia matemática (5).

Se puede concluir entonces que:

Los modelos matemáticos son sistemas de ecuaciones y proposiciones lógicas que intentan representar las **relaciones entre variables** (propiedades mensurables del sistema cuyas magnitudes varían en el tiempo) **y parámetros** (cantidades temporalmente invariables que caracterizan al sistema). Cuando en la elaboración de modelos matemáticos se acude al uso de los ordenadores puede hablarse de **modelos computacionales** (6).

Estos modelos matemáticos de los que se ha hablado pueden ser y son representados mediante los siguientes objetos matemáticos:

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

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 sub-espacio de soluciones, donde tales soluciones tienen sentido biológico (7).

Todo el proceso a través del cual se llega a un modelo matemático, donde se formula, valida y se obtienen datos está representado en la fig.1.1.

Figura 1.1 Proceso de Modelado Matemático.

1.5 Sistemas de Ecuaciones Diferenciales

Todo el proceso de simulación que lleva a cabo la versión 1.0 del Software BioSyS se realiza mediante modelos matemáticos expresados y/o construidos con Sistemas de Ecuaciones Diferenciales, abordar este tema en el presente documento resulta de vital importancia para la compresión y análisis del problema.

*"Un **sistema de ecuaciones diferenciales** es un conjunto de varias ecuaciones diferenciales con varias funciones incógnitas y un conjunto de condiciones de contorno* (8)*."*

1.5.1 Sistemas de Ecuaciones Diferenciales Ordinarias

En un sistema de ecuaciones diferenciales ordinarias de cualquier orden, puede ser reducido a un sistema equivalente de primer orden, si se introducen nuevas variables y ecuaciones. Un sistema de ecuaciones diferenciales ordinarias de primer orden escrito en forma explícita es un sistema de ecuaciones de la forma:

$$\begin{cases} \dfrac{dx_1}{dt} = F_1(x_1, x_2, \ldots, x_n; t) \\ \dfrac{dx_2}{dt} = F_2(x_1, x_2, \ldots, x_n; t) \\ \ldots \\ \dfrac{dx_n}{dt} = F_n(x_1, x_2, \ldots, x_n; t) \end{cases}$$

1.5.2 Reducción a un Sistema de Primer Orden

Dado un sistema de ecuaciones diferenciales de orden *n* con *m* ecuaciones:

$$F_i\left(x_j, \frac{dx_j}{dt}, \ldots, \frac{d^n x_i}{dt^n}; t\right) = 0 \qquad \text{con } i, j \in \{1, 2, \ldots, m\}$$

Existe un sistema equivalente de primer orden con a lo sumo (*n*+1)x*m* ecuaciones. Para ver esto se considera un sistema en que intervienen *m* funciones incógnitas x_i y sus *n* derivadas, y se introduce un nuevo conjunto de variables $y_{i,k}$ definidas de la siguiente manera:

$$y_{i,k}(t) := \frac{d^k x_i(t)}{dt^k}$$

El sistema de primer orden equivalente en las variables $y_{i,k}$ resulta ser (9):

$$\begin{cases} y_{i,k+1} = \dfrac{dy_{i,k}}{dt} & k \in \{0, 2, \ldots, n-1\} \\ F_i\left(y_{j,0}, y_{j,1}, \ldots, y_{j,n}; t\right) = 0 & i, j \in \{1, 2, \ldots, m\} \end{cases}$$

1.6 Métodos para resolver Sistemas de Ecuaciones Diferenciales (SED)

Existen diferentes métodos para resolver sistemas de ecuaciones diferenciales los más comunes son:

1-Método de Euler

2-Método de Heun

3-Método de Taylor

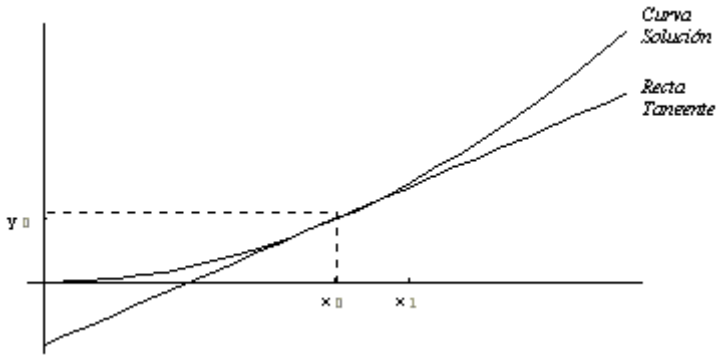
4-Método de Runge-Kutta

5-Método de Adams

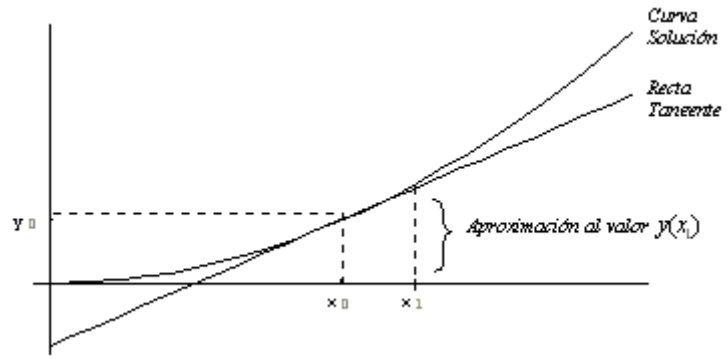
Para una mejor comprensión se hará una breve explicación del funcionamiento de algunos de ellos. Téngase en cuenta que la mayoría de estos métodos han sido implementados con anterioridad e incorporados a librerías que permiten su uso definiéndole una serie de parámetros.

1.6.1 Método de Euler

La idea del método de Euler es muy sencilla y está basada en el significado geométrico de la derivada de una función en un punto dado. Suponga que tuviera la curva solución de la ecuación diferencial y trace la recta tangente a la curva en el punto dado por la condición inicial.



Debido a que la recta tangente aproxima a la curva en valores cercanos al punto de tangencia, se puede tomar el valor de la recta tangente en el punto x_1 como una aproximación al valor deseado $y(x_1)$.



Si se calcula la ecuación de la recta tangente a la curva solución de la ecuación diferencial dada en el punto (x_0, y_0). Sabemos que la ecuación de la recta es:

$$y = m(x - x_0) + y_0$$

Donde *m* es la pendiente. En este caso, se sabe que la pendiente de la recta tangente se calcula con la derivada:

$$m = y'\big|_{(x_0, y_0)} = f(x_0, y_0)$$

Por lo tanto, la ecuación de la recta tangente es:

$$y = f(x_0, y_0)(x - x_0) + y_0$$

Ahora bien, suponga que x_1 es un punto cercano a x_0, y por lo tanto estará dado como $x_1 = x_0 + h$. De esta forma, se tiene la siguiente aproximación:

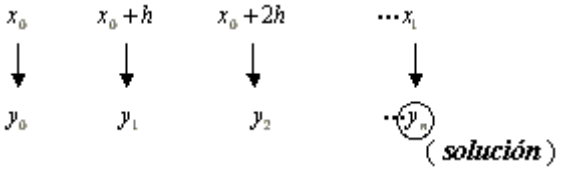
$$y(x_1) = y(x_0 + h) \approx f(x_0, y_0)(y_0 + h - x_0) + y_0$$

De aquí, se obtiene la fórmula de aproximación:

$$y(x_0 + h) \approx y_0 + h \cdot f(x_0, y_0)$$

Esta aproximación puede ser suficientemente buena, si el valor de *h* es realmente pequeño, digamos de una décima ó menos. Pero si el valor de *h* es más grande, entonces se puede cometer un gran error al aplicar dicha fórmula. Una forma de reducir el error y obtener de hecho un método iterativo, es dividir la distancia $h = |x_1 - x_0|$ en *n* partes iguales (procurando que estas partes sean de longitud suficientemente pequeña) y obtener entonces la aproximación en *n* pasos, aplicando la fórmula anterior *n* veces de un paso a otro, con la nueva *h* igual a $\frac{|x_1 - x_0|}{n}$.

En una gráfica, se tiene lo siguiente:



Ahora bien, se sabe que:

$$y_1 = y_0 + hf(x_0, y_0)$$

Para obtener y_2 únicamente hay que pensar que ahora el papel de (x_0, y_0) lo toma el punto (x_1, y_1), y por lo tanto, si se sustituyen los datos adecuadamente, se obtiene que:

$$y_2 = y_1 + hf(x_1, y_1)$$

De aquí se ve claramente que la fórmula recursiva general, está dada por:

$$y_{n+1} = y_n + hf(x_n, y_n)$$

Esta es la conocida fórmula de Euler que se usa para aproximar el valor de $y(x_1)$ aplicándola sucesivamente desde x_0 hasta x_1 en pasos de longitud *h* (10).

1.6.2 Método de Heun

Este método es una mejora que se efectúa sobre el Método de Euler. En Euler, se aproxima el siguiente valor de la función conociendo dónde se encuentra ahora y la pendiente en el punto actual.

Esto puede llevar a una mala aproximación, pues la pendiente de la función puede cambiar en el intervalo analizado, y Euler arrojaría un valor muy alejado.

Heun propone tomar en consideración tanto la pendiente en el punto actual como la pendiente en el siguiente punto, promediadas, quedando su sucesión de la siguiente manera:

$$x_{i+1} = x_i + h\frac{1}{2}(pend_{izq} + pend_{der})$$

Pero uno no sabe la pendiente en el siguiente punto dado que uno no conoce exactamente dónde será (si se supiera, ¡no tendría sentido aproximarlo!). Entonces, para aproximar la pendiente "derecha" se utiliza el Método de Euler, como puede verse a continuación.

$$\begin{cases} pend_{izq} = f(t_i, x_i) \\ pend_{der} = f(t_{i+1}, x_i + hf(t_i, x_i)) \end{cases}$$

Entonces, se obtiene finalmente la siguiente sucesión:

$$x_{i+1} = x_i + h\frac{1}{2}(f(t_i, x_i) + f(t_{i+1}, x_i + hf(t_i, x_i))) \quad (11)$$

1.6.3 Método de Runge-Kutta

El método de Runge-Kutta es un método genérico de resolución numérica de ecuaciones diferenciales. Este conjunto de métodos fue inicialmente desarrollado alrededor del año 1900 por los matemáticos C. Runge y M. W. Kutta. Se trata de un método por etapas que tiene la siguiente expresión genérica:

$$u^{n+1} = u^n + \Delta t \sum_{i=1}^{e} b_i k_i \quad ,$$

Donde:

$$k_i = F(u^n + \Delta t \sum_{j=i}^{e} a_{ij} k_j; t_n + c_i \Delta t) \qquad i = 1,...,e$$

Con a_{ij},b_i,c_i constantes propias del esquema numérico. Los esquemas Runge-Kutta pueden ser explícitos o implícitos dependiendo de las constantes a_{ij} del esquema. Si esta matriz es triangular inferior con todos los elementos de la diagonal principal iguales a cero; es decir, $a_{ij} = 0$ para $j = i,...,e$, los esquemas son explícitos.

La convergencia lenta del método de Euler y lo restringido de su región de estabilidad absoluta nos lleva a considerar métodos de orden de convergencia mayor. En cada paso el método de Euler se mueve a lo largo de la tangente de una cierta curva que esta "cerca" a la curva desconocida o buscada. Los métodos Runge-Kutta extienden esta idea geométrica al utilizar varias derivadas o

tangentes intermedias, en lugar de solo una, para aproximar la función desconocida. Los métodos Runge-Kutta más simples se obtienen usando dos de estas derivadas intermedias (12).

1.6.4 Método 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 (13).

1.7 Software que permiten solucionar SED

Desde el momento mismo en que se crearon las primeras herramientas informáticas, los asistentes matemáticos formaron parte de ellas, y es que el hombre siempre ha necesitado herramientas que le permitan explorar el complejo mundo de las matemáticas, si una vez se apoyó en el ábaco y en la calculadora mecánica, hoy todos los complejos mecanismo de cálculo están soportados por la computación.

De este tipo de herramientas se conocen una gran variedad, diferenciadas principalmente por su posibilidad de realizar complejos cálculos, por su rapidez, por las funciones que incorporan, los precios a los que se venden y las facilidades de integración con otros sistemas.

Algunos de estos asistentes son:

1- **Derive:** Asistente de cálculo matemático para PC.
2- **Octave:** Asistente de cálculo matemático, distribuido bajo la Licencia Publica General(GPL)
3- **Mathematica:** Herramienta de cálculo numérico y simbólico, visualización y manipulación de datos, gráficos y objetos, que proporciona un lenguaje de programación de alto nivel.
4- **webMathematica**: Tecnología para la generación de contenido HTML dinámico que posibilita el cálculo técnico en la Web.

5- **Control System Professional 2.0 :** Diseño, análisis y simulación de sistemas dinámicos, con capacidades simbólicas integradas (requiere Mathematica)
6- **gridMathematica 2 (basado en Mathematica 6)**: Cálculo distribuido y paralelizado con Mathematica (tecnología grid).
7- **MatLab:** Software de análisis y calculo numérico y simbólico (14).

La mayoría de los que se han mencionado son muy potentes y con altas funcionalidades que aseguran un trabajo científico de calidad. La implementación de la primera versión de BioSyS incluye la solución de sistemas de ecuaciones diferenciales utilizando como herramienta de cálculo al asistente matemático **MatLab**, el presente trabajo asegura además, la creación de funcionalidades dentro del módulo de simulación que permita resolver los SED haciendo uso del asistente **Octave**.

Usar el asistente matemático **Octave** ha sido una decisión estratégica, debido a que a pesar de ser quizás uno de los más potente, (por su capacidad para resolver problemas de altísima complejidad y de tener implementado funciones para resolver Sistemas de Ecuaciones Diferenciales numéricas y simbólicas), su licencia se obtiene sin ningún costo, por tanto no hay problemas a la hora de usarlo en múltiples ordenadores, lo que representaría una ventaja para cualquier organismo o empresa, que usara nuestro simulador y no existiría ningún inconveniente a la hora de realizar cálculo distribuido.

1.7.1 Asistente Matemático MatLab

MATLAB (abreviatura de *MATrix LABoratory*, "laboratorio de matrices") es un software matemático que ofrece un entorno de desarrollo integrado (IDE) con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Apple Mac OS X.

Entre sus prestaciones básicas se hallan: la manipulación de matrices, la representación de datos y funciones, la implementación de algoritmos, la creación de interfaces de usuario (GUI) y la comunicación con programas en otros lenguajes y con otros dispositivos hardware. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, Simulink (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Además, se pueden ampliar las capacidades de MATLAB con las *cajas de herramientas* (*toolboxes*); y las de Simulink con los *paquetes de bloques* (*blocksets*).

Es un software muy usado en universidades y centros de investigación y desarrollo. En los últimos años ha aumentado el número de prestaciones, como la de programar directamente procesadores digitales de señal o crear código VHDL (15).

Durante mucho tiempo hubo críticas porque MATLAB es un producto propietario de The Mathworks, ya que los usuarios están sujetos a un vendor lock-in. Recientemente se ha proporcionado una herramienta adicional llamada MATLAB Builder bajo la sección de herramientas Application Deployment para utilizar funciones MATLAB como archivos de biblioteca que pueden ser usados con ambientes de construcción de aplicación .NET o Java. Pero la desventaja es que el computador donde la aplicación tiene que ser utilizada necesita MCR (MATLAB Component Runtime) para que los archivos MATLAB funcionen correctamente. MCR puede ser distribuido libremente con archivos de biblioteca generados por el compilador MATLAB.

Este asistente matemático brinda una gran cantidad de métodos para la solución de Sistemas de Ecuaciones Diferenciales, ver la siguiente tabla.

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 implícitas	BDFs

Tabla 1.1: Métodos para la solución de SED brindados por el MatLab.

Este asistente matemático puede ser usado además en una gran diversidad de sistemas operativos, es decir, es multiplataforma y su lenguaje de modelado M es sencillo, inductivo y fácil de aprender. Permite además operaciones con vectores y matrices. Dispone también en la actualidad de un amplio

abanico de programas de apoyos especializados, denominados Toolboxes, que extienden significativamente el número de funciones incorporadas en el programa principal.

Estos Toolboxes cubren en la actualidad, prácticamente casi todas las áreas principales en el mundo de la ingeniería y la simulación, destacando entre ellos el 'toolbox' de proceso de imágenes, señal, control robusto, estadística, análisis financiero, matemáticas simbólicas, redes neurales, lógica difusa, identificación de sistemas, simulación de sistemas dinámicos, etc. es un entorno de cálculo técnico, que se ha convertido en estándar de la industria, con capacidades no superadas en computación y visualización numérica (16).

Todas estas funcionalidades y características propias del asistente indujeron a que se usara en la versión 1.0 de BioSyS.

Sin embargo incompatibilidades del lenguaje en el que se desarrolló la aplicación (**Java**) y este asistente mientras se corre sobre el Sistema Operativo Windows hicieron necesaria la búsqueda de soluciones alternativas, además el alto costo de su licencia influye negativamente, el uso de otro asistente matemático es un pilar básico de la estrategia del proyecto BioSyS, debido a que tanto garantizar el buen funcionamiento del software sobre el sistema operativo más usado del mundo como proveer a los usuarios de una herramienta suficientemente económica es de vital importancia.

1.7.2 Asistente Matemático Octave

Octave o **GNU Octave** es un programa libre para realizar cálculos numéricos. Como indica su nombre es parte de proyecto GNU. MATLAB es considerado su equivalente comercial. Entre varias características que comparten se puede destacar que ambos ofrecen un intérprete permitiendo ejecutar órdenes en modo interactivo. Octave no es un sistema de álgebra computacional como podría ser Maxima, sino que usa un lenguaje que está orientado al análisis numérico (17).

El proyecto fue creado alrededor del año 1988 pero con una finalidad diferente: Ser utilizado en un curso de diseño de reactores químicos. Posteriormente en el año 1992, se decide extenderlo y comienza su desarrollo a cargo de John W. Eaton. La primera versión alpha fue lanzada el 4 de enero de 1993. Un año más tarde, el 17 de febrero, 1994 aparece la versión 1.0.

El nombre surge del nombre de un profesor que fue uno de los autores conocido por sus buenas aproximaciones por medio de cálculos mentales a problemas numéricos (18).

-Detalles Técnicos

- Octave está escrito en C++ usando la librería STL.
- Tiene un intérprete de su propio lenguaje (de sintaxis similar a MatLab), y permite una ejecución interactiva o por lotes.
- Puede extenderse el lenguaje con funciones y procedimientos por medios de módulos dinámicos.
- Utiliza otros programas GNU para ofrecer al usuario crear gráficos para luego imprimirlos o guardarlos.
- Dentro del lenguaje también se comporta como una consola de órdenes (shell). Esto permite listar contenidos de directorios, por ejemplo.
- Además de correr en plataformas Unix también lo hace en Windows.
- Puede cargar archivos con funciones de Matlab de extensión *.m*.

-Ventajas que ofrece:

Este asistente matemático además de ser potente, brinda una gran cantidad de funcionalidad y ventajas que invita a incorporarlo a la solución presentada por BioSyS. Algunas de estas ventajas son:

- La sintaxis es similar a la utilizada en MATLAB.
- Es un lenguaje interpretado.
- No permite pasar argumentos por referencia. Siempre se pasan por valor.
- No permite punteros.
- Se pueden generar scripts.
- Soporta gran parte de las funciones de la librería estándar de C.
- Puede extenderse para ofrecer compatibilidad a las llamadas al sistema UNIX.
- El lenguaje está pensado para trabajar con matrices y provee mucha funcionalidad para trabajar con éstas.
- No es un lenguaje de programación orientado a objetos. Por lo tanto, no tiene clases ni objetos.
- Soporta estructuras similares a los "struct"s de C.

Su licencia es la licencia pública general de GNU, por lo que puede ser copiado, modificado y utilizado libremente.

Específicamente se centrará la atención en este asistente, porque representa una alternativa real frente a la vía tan costosa como es el uso del MatLab. La solución de Sistemas de Ecuaciones Diferenciales mediante Octave se puede realizar a través de varios métodos matemáticos que usan básicamente una rutina Runge-Kutta para darle solución. Los objetivos iniciales se basaban en el uso del método Lsode, propuesta en cada una de las versiones de Octave que han salido al mercado, sin embargo al asistente se le puede incorporar un paquete llamado OdeSolver que contiene las funciones Ode23, Ode45 y Ode78. Por tanto se convierte entonces en una alternativa real incorporar estos métodos al Simulador.

La versión 1.0 de BioSyS, además de darle solución a los modelos matemáticos haciendo uso de asistentes matemáticos, tiene incorporadas funcionalidades que hacen uso de las librerías ODEtoJava, paquetes nativos del lenguaje java que resuelven Sistemas de Ecuaciones Diferenciales bajo diferentes métodos, incorporando la posibilidad de que la aplicación en sí, resuelva un Sistema de Ecuaciones Diferenciales sin tener que depender de un asistente matemático.

1.7.3 Paquete Librerías 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 esté instalada la máquina virtual de Java.

Esta librería utilizada con éxito en la primera versión de BioSyS, brinda varios métodos para la solución de SED, ellos son:

- ErkTriple
- Erk
- ErkSD
- Dormand Price

Todos hacen uso de la rutina de solución propuesta por el método Runge-Kutta, pero están concebidos de manera que en su forma primitiva son imposibles de usar por el simulador, debido a que los parámetros que reciben y el tipo de datos que devuelven no forman parte de la arquitectura del proyecto, por tanto se requiere de un trabajo de programación que permita adaptarlos para usarlos en la aplicación a desarrollar.

1.8 Cálculo Distribuido

Se habló en la introducción de esta tesis de grado sobre la utilidad de tener un sistema que posibilitara distribuir de manera ordenada y segura los cálculos matemáticos que se realizaban. Sin embargo, ahora surge la pregunta ¿Es realmente necesario utilizar una herramienta de este tipo?

Se puede afirmar que el siglo XX fue el padre de las computadoras, desde el momento mismo que surgió el primer equipo que lograba realizar una operación numérica básica, surgió la necesidad de aumentar la velocidad con la que este equipo calculaba. Este proceso se fue repitiendo hasta contar con las supercomputadoras de hoy en día que realizan miles de millones de operaciones en nanosegundos, sin embargo, estos equipos cuestan tales cantidades de dinero, que es irreal para un centro de estudios, una empresa pequeña o un grupo investigativo adquirirlas, estando solo disponibles para entidades o personas que pueden manejar estas sumas, lo que impide la investigación y el desarrollo de manera justa y equitativa. Generalmente se puede decir que los simuladores biológicos y en este caso especifico BioSyS tendrán que manejar volúmenes gigantescos de informaciones, datos y cálculos, de lo que se deriva que un equipo de súper cómputo sería lo ideal, sin embargo esa no es una solución viable, es por ello que se han dirigido los esfuerzos a una alternativa barata, no muy compleja y eficiente: Los Sistemas Distribuidos.

Actualmente se pueden adquirir en el mercado internacional equipos de cómputo personal a un precio relativamente cómodo y con altos niveles de procesamiento, estos niveles en si no son suficientes, pero cuando se cuenta con una gran cantidad de máquinas interconectas, se puede aprovechar la potencia de cálculo de cada una y sumarla de manera que se pueda ir resolviendo un problema gigantesco, por pequeños lotes.

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.

1.8.1 Clúster de Computadoras

El término clúster se aplica a los conjuntos o conglomerados de computadoras construidos mediante la utilización de componentes de hardware comunes y que se comportan como si fuesen una única computadora. Hoy en día juegan un papel importante en la solución de problemas de las ciencias, las ingenierías y del comercio moderno.

La tecnología de clúster ha evolucionado en apoyo de actividades que van desde aplicaciones de súper computo y software de misiones críticas, servidores Web y comercio electrónico, hasta bases de datos de alto rendimiento, entre otros usos.

El cómputo con clúster surge como resultado de la convergencia de varias tendencias actuales que incluyen la disponibilidad de microprocesadores económicos de alto rendimiento y redes de alta velocidad, el desarrollo de herramientas de software para cómputo distribuido de alto rendimiento, así como la creciente necesidad de potencia computacional para aplicaciones que la requieran.

Simplemente, clúster es un grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

Los clúster son usualmente empleados para mejorar el rendimiento y/o la disponibilidad por encima de la que es provista por un solo computador típicamente siendo más económico que computadores individuales de rapidez y disponibilidad comparables.

La construcción de los ordenadores del clúster es más fácil y económica debido a su flexibilidad: pueden tener todos la misma configuración de hardware y sistema operativo (**clúster homogéneo**), diferente rendimiento pero con arquitecturas y sistemas operativos similares (**clúster semi-homogéneo**), o tener diferente hardware y sistema operativo (**clúster heterogéneo**). Lo que hace más fácil y económica su construcción.

Para que un clúster funcione como tal, no basta solo con conectar entre sí los ordenadores, sino que es necesario proveer un sistema de manejo del clúster, el cual se encargue de interactuar con el usuario y los procesos que corren en él para optimizar el funcionamiento (19).

En este caso la computación Grid ofrece una solución viable, específicamente la herramienta T-arenal es la encargada de gestionar todo el procesamiento distribuido.

1.8.2 Computación Grid

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

El término *Grid* se refiere a una infraestructura que permite la integración y el uso colectivo de ordenadores de alto rendimiento, redes y bases de datos que son propiedad y están administrados por diferentes instituciones. Puesto que la colaboración entre instituciones envuelve un intercambio de datos, o de tiempo de computación, el propósito del *Grid* es facilitar la integración de recursos computacionales.

Universidades, laboratorios de investigación o empresas se asocian para formar *Grid* para lo cual utilizan algún tipo de software que implemente este concepto (20).

-¿Específicamente que representa?

La Grid representa al sistema de computación distribuido que permite compartir recursos no centrados geográficamente para resolver problemas de gran escala. Los recursos compartidos pueden ser ordenadores (PCs, estaciones de trabajo, supercomputadoras, PDA, portátiles, móviles, etc.), software, datos e información, instrumentos especiales (radio, telescopios, etc.) o personas/colaboradores.

-Ventajas

La computación Grid ofrece muchas ventajas frente a otras tecnologías alternativas. La potencia que ofrecen una multitud de computadores conectados en red usando Grid es prácticamente ilimitada, además de que ofrece una perfecta integración de sistemas y dispositivos heterogéneos, por lo que las conexiones entre diferentes máquinas no generarán ningún problema. Se trata de una solución altamente escalable, potente y flexible, ya que evitarán problemas de falta de recursos (cuellos de botella) y nunca queda obsoleta, debido a la posibilidad de modificar el número y características de sus componentes.

Estos recursos se distribuyen en la red de forma transparente pero guardando unas pautas de seguridad y políticas de gestión de carácter tanto técnico como económico. Así pues, su objetivo será el de compartir una serie de recursos en la red de manera uniforme, segura, transparente, eficiente y fiable, ofreciendo un único punto de acceso a un conjunto de recursos distribuidos geográficamente en diferentes dominios de administración.

Todas estas ventajas de la computación Grid han definido que se desarrollara en la universidad y específicamente por la facultad 6 perteneciente al Polo de Bioinformática, una herramienta llamada T-Arenal, que basa su funcionamiento en los principios de la computación Grid.

1.8.3 Herramienta Computacional 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 (3).

-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.

-Ventajas que brinda:

- Es software libre, de fácil instalación, se dispone del código fuente y se le pueden realizar modificaciones y distribuir sin problemas.
- 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.
- 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 esté 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.
- 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.
- Es posible reajustar la granularidad o tamaño de las sub-tareas en que se descompone el problema de forma dinámica en dependencia del comportamiento y el rendimiento que se esté obteniendo en el cómputo distribuido.

- El sistema es de propósito general, es decir, que puede ser utilizado para dar solución a cualquier problema que pueda ser descompuesto en sub-tareas independientes. Además, se puede cambiar el algoritmo que utilizan los nodos esclavos para realizar los cálculos sin necesidad de reiniciar todo el sistema.
- Se pueden correr varios cómputos distribuidos de forma simultánea y darle a prioridad a los problemas que demandan de mayor procesamiento.
- Es posible administrar y gestionar todo el sistema de cómputo de forma remota mediante una aplicación de interfaz gráfica.
- 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.
- 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.
- 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 (3).

1.9 Metodología de Desarrollo OpenUP

Con una buena metodología se pretende reducir costos y retrasos de proyectos así como mejorar la calidad del software. La metodología de desarrollo cobra gran importancia en proyectos empresariales pues al no utilizarla adecuadamente se puede desembocar en la frustración del equipo de desarrollo y en la insatisfacción de los clientes. Por tanto el uso de una metodología es necesario para controlar el ciclo de vida de un proyecto.

La Metodología de desarrollo usada en el proyecto y para el desarrollo del simulador es OpenUp, se brinda una pequeña descripción de sus fortalezas y ventajas, principales elementos que influyeron para que fuera usada.

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

1.9.1 Principios del OpenUP

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

1.9.2 Ventajas que brinda

-Es un proceso de desarrollo del software completo en el sentido que puede ser manifestado como todo el proceso para construir un sistema.

-Es extensible ya que en el proceso se pueda agregar o adaptar según lo vayan requiriendo los sistemas. OpenUp es un proceso ágil.

-Es ligero y proporciona una comprensión detallada del proyecto, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad.

-Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.

-OpenUp es la metodología utilizada por desarrolladores de alto nivel en casi todo el mundo por sus altas cualidades administrativas (21).

1.10 Lenguaje de Modelado

Utilizar algún lenguaje de modelado representa un pilar fundamental en cualquier proyecto de desarrollo, por su universalidad, sencillez y potencia será usado en proyectos por cada una de las tareas que lo requieran.

Lenguaje Unificado de Modelado (**UML**, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software (en nuestro caso es altamente compatible con OpenUp), pero no especifica en sí mismo qué metodología o proceso usar (22).

UML cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de las entidades representadas. En UML 2.0 hay 13 tipos diferentes de diagramas.

Los ***Diagramas de Estructura*** enfatizan en los elementos que deben existir en el sistema modelado:

- Diagrama de clases
- Diagrama de componentes
- Diagrama de objetos
- Diagrama de estructura compuesta (UML 2.0)
- Diagrama de despliegue
- Diagrama de paquetes

Los ***Diagramas de Comportamiento*** enfatizan en lo que debe suceder en el sistema modelado:

- Diagrama de actividades
- Diagrama de casos de uso
- Diagrama de estados

Los ***Diagramas de Interacción*** son un subtipo de diagramas de comportamiento, que enfatiza sobre el flujo de control y de datos entre los elementos del sistema modelado (23):

- Diagrama de secuencia
- Diagrama de comunicación, que es una versión simplificada del Diagrama de colaboración (UML 1.x)
- Diagrama de tiempos (UML 2.0)
- Diagrama de vista de interacción (UML 2.0)

1.11 Herramienta Case

CASE es una sigla, que corresponde a las iniciales de: **C**omputer **A**ided **S**oftware **E**ngineering; y en su traducción al Español significa Ingeniería de Software Asistida por Computación.

El concepto de CASE es muy amplio; y una buena definición genérica, que pueda abarcar esa amplitud de conceptos, sería la de considerar a la Ingeniería de Software Asistida por Computación (CASE), como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación (24).

Se puede definir además a las Herramientas CASE como un conjunto de programas y ayudas que dan asistencia a los analistas, ingenieros de software y desarrolladores, durante todos los pasos del Ciclo de Vida de desarrollo de un Software. Como es sabido, los estados en el Ciclo de Vida de desarrollo de un Software son: Investigación Preliminar, Análisis, Diseño, Implementación e Instalación (25).

La ingeniería de software aplicada al software desarrollado ha estado asistida específicamente por la herramienta case Visual Paradigm.

1.11.1 Herramienta Case Visual Paradigm

Es una herramienta CASE que utiliza “UML”: como *lenguaje* de modelaje. Se integra con las siguientes herramientas Java:

- Eclipse/IBM WebSphere
- JBuilder
- NetBeans IDE
- Oracle JDeveloper
- BEA Weblogic

Está disponible en varias ediciones, cada una destinada a unas necesidades: Enterprise, Professional, Community, Standard, Modeler y Personal.

Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación (26).

-Algunas características de la Herramienta

1. Soporte de UML versión 2.1.
2. Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio.
3. Documento Modelado colaborativo con CVS y Subversion (nueva característica).
4. Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI (nueva característica).
5. Ingeniería de ida y vuelta.
6. Ingeniería inversa - Código a modelo, código a diagrama.
7. Ingeniería inversa Java, C++, Esquemas XML, XML, .NET exe/dll, CORBA IDL.
8. Generación de código - Modelo a código, diagrama a código.
9. Editor de Detalles de Casos de Uso - Entorno todo-en-uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
10. Diagramas EJB - Visualización de sistemas EJB.
11. Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
12. Diagramas de flujo de datos.
13. Soporte ORM - Generación de objetos Java desde la base de datos.

14. Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
15. Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.
16. Generador de informes para generación de documentación.
17. Distribución automática de diagramas - Reorganización de las figuras y conectores de los diagramas UML.
18. Importación y exportación de ficheros XMI.
19. Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio.

Entre otras muchas características que lo hace una de las herramientas case más potentes.

-Editor de figuras

Otras herramientas y plugins.

Plataforma Java (Windows/Linux/Mac OS X):
SDE para Eclipse
SDE para NetBeans
SDE para Sun ONE
SDE para Oracle JDeveloper
SDE para JBuilder
SDE para IntelliJ IDEA
SDE para WebLogic Workshop

1.12 Lenguaje de Programación

El Simulador de Sistemas Biológicos BioSyS ha sido desarrollado en su totalidad haciendo uso del lenguaje Java, este modulo incorporará funcionalidades que serán desarrolladas en este mismo lenguaje, por su potencia y compatibilidad con el sistema en general. Una breve descripción del lenguaje permite comprender su uso por el equipo de desarrollo.

Java es un lenguaje de programación (como C, C++, BASIC, Pascal o Logo) que sirve para crear aplicaciones informáticas. Algunas de sus características más destacables son:

1- Una misma aplicación puede funcionar en diversos tipos de ordenadores y sistemas operativos: Windows, Linux, Solaris, MacOS-X, así como en otros dispositivos inteligentes.

2- Los programas Java pueden ser aplicaciones independientes (que corren en una ventana propia) o "applets": pequeños programas interactivos que se encuentran incrustados en una página web y pueden funcionar con cualquier tipo de navegador: Explorer, Netscape, Ópera...
3- Se trata de un lenguaje "orientado a objetos". Esto significa que los programas se construyen a partir de módulos independientes, y que estos módulos se pueden transformar o ampliar fácilmente. Un equipo de programadores puede partir de una aplicación existente para extenderla con nuevas funcionalidades.
4- Su desarrollo está impulsado por un amplio colectivo de empresas y organizaciones, y conecta con la filosofía de software abierto y entorno colaborativo.
5- 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.
6- 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.
7- 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 (27).

La aplicación del lenguaje de programación Java ha estado asociada en nuestro proyecto al Entorno de Desarrollo Integrado NetBeans.

1.12.1 Entorno de Desarrollo Integrado (IDE) NetBeans

Desarrollado por Sun MicroSystem usando la plataforma NetBeans, su última versión es la 6.5 (19 de noviembre del 2008), es multiplataforma, licencia CCDL y actualmente se encuentra traducido al español.

La plataforma 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 (manifest file) 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 (28).

Soporta el desarrollo de todos los tipos de aplicación Java (J2SE, Web, EJB y aplicaciones móviles), además todas las funciones del IDE son provistas por módulos. Cada módulo provee una función bien definida, tales como el soporte de Java, edición, o soporte para el sistema de control de versiones. NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente. A todo esto se le suma que este entorno de desarrollo es libre y gratuito.

1.12.2 Herramienta ORM (Mapeo de Objetos Relacional)

El mapeo objeto-relacional (Object-Relational-Mapping) 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 relacional. En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo) (29).

El uso de una herramienta ORM ha sido significativo a la hora de entrelazar el sistema y la base de datos, básicamente el empleo de ORM lo ha brindado en el proyecto la herramienta Hibernate.

1.12.3 Herramienta ORM Hibernate

Hibernate es una herramienta de Mapeo objeto-relacional para la plataforma Java, que facilita el mapeo de atributos entre una base de datos relacional tradicional y el modelo de objetos de una aplicación, mediante archivos declarativos (XML) que permiten establecer estas relaciones. Hibernate es software libre, distribuido bajo los términos de la licencia GNU LGPL (30).

Esta herramienta soporta una gran cantidad de bases SQL, una de sus mayores potencialidades es que puede mapear clases descritas en el lenguaje de programación Java a tablas de una base de datos y también tipos de datos en Java a tipos de datos de SQL. Para lograr la persistencia de objetos usa la API de JAVA JDBC.

A partir del trabajo con esta herramienta se genera la API BSDAO.jar, la misma tiene implementada toda la lógica para lograr la comunicación con la Base de Datos.

1.13 Roles y Artefactos

De acuerdo a la metodología que se usa: OpenUp, y respetando los roles, artefactos y trabajadores que la misma define, se ha decidido desempeñar los roles de Analista y Desarrollador, así como los artefactos que cada uno genera por sí solo.

Rol: Analista

Las personas en este rol representan al cliente y los usuarios finales involucrados, obteniendo información desde los stakeholders para entender el problema a ser resuelto y capturar y ajustar las prioridades para los requerimientos (31).

Un analista necesita los siguientes conocimientos, competencias y habilidades:

- Experticia en identificar y entender problemas y oportunidades
- Habilidad para articular las necesidades asociadas con los principales problemas a resolver u oportunidad para ser realizados
- Habilidad para colaborar efectivamente con otros miembros del grupo a través de sesiones de trabajo colaborativo, workshops, sesiones JAD y otras técnicas.
- Buenas competencias comunicativas, verbales y de escritura
- Conocimiento del negocio y dominio de la tecnología o habilidad para absorber y entender rápidamente tal información.

Fig.1.2 Artefactos que genera el rol Analista.

Además el analista es el encargado de generar los siguientes artefactos:

- Analyze the Architectural Requirements (Análisis de los Requerimientos de Arquitectura)
- Assess Results (Evaluación de los Resultados)
- Create Test Cases (Diseño de Casos de Prueba)
- Design the Solution (Diseño de la Solución)
- Develop the Architecture (Desarrollo de la Arquitectura)
- Manage Iteration (Administración de las Iteraciones)
- Plan Iteration (Plan de Iteración)
- Plan Project (Plan de Proyecto)

Y modifica:

- Glossary (Glosario)
- Supporting Requirements (Requerimientos de Soporte)
- Use Case (Casos de Uso)
- Use-Case Model (Diagrama de Casos de Uso)
- Vision (Visión)
- Work Items List (Lista de hitos de Trabajo)

Rol: Desarrollador

La persona en este rol es responsable por desarrollar una parte del sistema, incluyendo diseñar esta para que se ajuste a la arquitectura, posiblemente prototipar la interfaz de usuario y entonces implementar, hacer pruebas unitarias e integrar los componentes que son parte de la solución (31).

Una persona en este rol necesita la habilidad necesaria para desempeñarse bien en estas tareas:

- Define y crea soluciones técnicas con la tecnología del proyecto.
- Entiende y se adapta a la arquitectura.
- Identifica y construye casos de prueba que cubren los comportamientos requeridos de los componentes técnicos.
- Comunica decisiones en una forma que otros miembros del grupo entienden.

Fig. 1.3 Artefactos que genera el rol Desarrollador.

Adicionalmente crea un modelo visual del sistema, este rol necesita la habilidad para representar el diseño en el Lenguaje de Modelado Unificado (Unified Modeling Language - UML).

Además el desarrollador es el encargado de generar los siguientes artefactos:

- Analyze the Architectural Requirements(Análisis de los Requerimientos de Arquitectura)
- Assess Results (Evaluación de los Resultados)
- Create Test Cases (Diseño de Casos de Prueba)
- Detail Requirements (Detalles de Requerimientos)
- Develop the Architecture (Desarrollo de la Arquitectura)
- Find and Outline Requirements (Búsqueda de Requisitos)
- Manage Iteration (Administración de Iteraciones)
- Plan Iteration (Plan de Iteración)
- Plan Project (Plan de Proyecto)

Y modifica:

- Build (Componentes)
- Design (Diseño)
- Developer Test (Pruebas de Desarrollo)
- Implementation (Implementación)
- Supporting Requirements (Requerimientos de Soporte)
- Test Log (Prueba de Trazas)
- Use Case (Caso de Uso)

1.14 Patrones de arquitectura y patrones de diseño

1.14.1 Concepto y Características de un Patrón

En la tecnología de objetos un **Patrón** es una descripción de un problema y la solución, a la que se le da un nombre, y que se puede aplicar a nuevos contextos (32).

Un patrón es un modelo que podemos seguir para realizar algo. Los patrones surgen de la experiencia de seres humanos de tratar de lograr ciertos objetivos. Los patrones capturan la experiencia existente y probada para promover buenas prácticas.

Los patrones:

- Ayudan a construir la experiencia colectiva de Ingeniería de Software.
- Son una abstracción de "problema – solución".
- Se ocupan de problemas recurrentes.
- Identifican y especifican abstracciones de niveles más altos que componentes o clases individuales.
- Proporcionan vocabulario y entendimiento común (33).

1.14.2 Patrones de Arquitectura

Aquéllos que expresan un esquema organizativo estructural fundamental para sistemas software (34).

Algunos de los Patrones de Arquitectura son:

1. Patrones de Estructura:
   - Capas
   - Tuberías y Filtros
   - Tableros
2. Sistemas Distribuidos:
   - Intermediario (Broker)
3. Sistemas Interactivos:
   - Modelo-Vista-Controlador
   - Presentación-Abstracción-Control
4. Sistemas Adaptables:
   - Microkernel

- Reflexión (35)

Dada la definición y de acuerdo a la función que realizan estos patrones de arquitectura se ha decidido para el diseño del módulo que se desea implementar utilizar el siguiente patrón:

Patrón Modelo-Vista-Controlador (Model-View-Controller):

El Modelo vista controlador es un patrón usado en ingeniería del software. La idea fundamental es separar en distintas capas la lógica del programa, los datos y la presentación. El Modelo representa la información, las vistas corresponden a los elementos de interfaz de usuario y los controladores gestionan la lógica de negocio que manipula dichos datos. Con los lenguajes orientados a objetos es mucho más cómodo escribir aplicaciones MVC ya que nos permite separar el código en bloques contenedores de operaciones y datos (36).

Modelo: Esta es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o portes en un carrito de la compra.

Vista: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista (37).

1.14.3 Patrones de Diseño

Los patrones de diseño son un conjunto de prácticas de óptimo diseño que se utilizan para abordar problemas recurrentes en la programación orientada a objetos.

Un patrón de diseño puede considerarse como un documento que define una estructura de clases que aborda una situación particular (38).

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

Se puede clasificar a estos patrones según su propósito:

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 (39).

Cada clasificación cuenta con varios tipos de patrones, que difieren de acuerdo a su funcionalidad propia, de allí los siguientes:

Patrones de creación (40):

- Abstract Factory. (Factoría Abstracta)
- Builder.
- Factory Method. (Método Factoría)
- Prototype. (Prototipo)
- Singleton.

Patrones estructurales:

- Adapter. (Adaptador)
- Bridge. (Puente)
- Composite. (Composición)
- Decorator. (Decorador)
- Facade. (Fachada)
- Flyweight.
- Proxy. (Proxy)

Patrones de comportamiento:

- Chain of Responsibility
- Command.
- Interpreter. (Intérprete)
- Iterator. (Iterador)
- Mediator. (Mediador)

- Memento. (Memento)
- Observer. (Observador)
- State. (Estado)
- Strategy. (Estratégico)

Se decide para el desarrollo del módulo, debido a la funcionalidad que los caracteriza, utilizar los siguientes patrones GoF:

Facade (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.

Observer (Observador): 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.

Strategy (Estratégico): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.

Patrones de diseño GRASP

Los patrones GRASP describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable.

En cuanto a las responsabilidades UML define una responsabilidad como “un contrato u obligación de un clasificador”.

Las responsabilidades están relacionadas con las obligaciones de un objeto en cuanto a su comportamiento.

Se pueden destacar 5 patrones Principales que son:

- Experto.
- Creador.
- Alta cohesión.

- Bajo acoplamiento.
- Controlador.

Y los cuatro patrones GRASP adicionales que son (32):

- Fabricación Pura.
- Polimorfismo.
- Indirección.
- No hables con extraños.

Debido a las características, función e importancia que presentan los siguientes patrones, se decide que se utilizarán para el desarrollo del sistema:

Expert – Experto: Tiene como problema: ¿Cuál es un principio general para asignar responsabilidades a los objetos? Y la solución: Asignar una responsabilidad al experto en información – la clase que tiene la información necesaria para la realización de la asignación.

Creator – Creador: Asigna a la clase B la responsabilidad de crear una instancia de clase A si se cumple uno o más de los casos siguientes:

- B agrega objetos de A
- B contiene objetos de A
- B registra instancias de objetos de A
- B utiliza más estrechamente objetos de A.
- B tiene datos de inicialización que se pasarán a un objeto de A cuando sea creado (por tanto, B es un Experto con respecto a la creación de A).
- B es un creador de los objetos A.

High Cohesion - Alta cohesión: Asigna una responsabilidad de manera que la cohesión permanezca alta.

Controller – Controlador: Asigna una responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase que representa una de las opciones siguientes:

- Representa el sistema global, dispositivo o subsistema.
- Representa un caso de uso en el que tiene lugar el evento del sistema a menudo denominado <nombre del caso de uso> Manejador, <nombre del caso de uso> coordinador, <nombre del caso de uso> Sesión.
- Utilice la misma clase controlador para todos los eventos del sistema en el mismo escenario de caso de uso.
- Informalmente, una sesión es una instancia de una conversación con un actor. Las sesiones pueden tener cualquier duración, pero se organizan a menudo en función de casos de uso.

1.15 Conclusiones del Capítulo

La creación o modificación de componentes de software necesita un correcto planteamiento y descripción del proceso a utilizar y de las herramientas involucradas. En este capítulo se han descrito detalladamente cada uno de estos parámetros, de lo cual se puede concluir de forma resumida que: Se le implementaron componentes al simulador de sistemas biológicos BioSyS, los mismos permiten la solución de Sistemas de Ecuaciones Diferenciales, que representan el núcleo de los modelos matemáticos que describen poblaciones de organismos, apoyados en el asistente matemático Octave, el paquete de librerías ODEtoJava y el Sistema de Computación Distribuida T-arenal.

Se utilizó como metodología de desarrollo OpenUp y como lenguaje de modelado UML, todo el proceso de software se apoyó en la herramienta case Visual Paradigm. Para la creación de los componentes programables se utilizó Java y como Entorno de Desarrollo Integrado Netbeans.

Capitulo 2: Características del Sistema

2.1 Introducción al capítulo

Este capítulo está dedicado a realizar un completo análisis sobre el sistema que se debe implementar, se mostrarán cada uno de los elementos vinculados. Se definen primeramente los requisitos funcionales y no funcionales como características inherentes al sistema que se desea automatizar, además se precisarán el actor y los casos de uso involucrados. Se hará énfasis además en la descripción de cada caso de uso, así como se mostrarán una serie de diagramas que permiten comprender mejor cómo se desarrolla el flujo de eventos en el sistema. En resumen este capítulo permitirá comprender de manera detallada qué se desea implementar y cuáles serán las funcionalidades definitivas que se adicionarán al simulador.

2.2 Modelo del Dominio

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema. Representa conceptos del mundo real, no de los componentes de software.

Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión). El modelo de dominio se representa en UML con un Diagrama de Clases en los que se muestra:

– conceptos u objetos del dominio del problema: clases conceptuales

– asociaciones entre las clases conceptuales

– atributos de la clase conceptuales (41)

A continuación se hará una breve descripción de los conceptos que se representan en el modelo de dominio asociado, facilitando de esta manera que se cree un mayor vínculo de información entre el cliente y el proveedor.

Conceptos del Modelo del Dominio

EcuaciónDiferencial*:* Es un componente de los Sistemas de Ecuaciones Diferenciales (SED).

Párametros*:* Los parámetros necesarios para resolver los SED.

Variables*:* Las variables necesarias para el solucionamiento de SED.

***Modelo**:* La descripción matemática que define una población.

***Simulador**:* Parte estructural encargada de gestionar las simulaciones.

***ODEtoJava**:* Librería encargada de resolver los SED.

***Rutina Erk**:* Rutina perteneciente a la librería ODEtoJava encargada de dar solución a los SED.

***Rutina ErkSD**:* Rutina perteneciente a la librería ODEtoJava encargada de dar solución a los SED.

***Rutina DormandPrince**:* Rutina perteneciente a la librería ODEtoJava encargada de dar solución a los SED.

***Simulación**:* Proceso resultante de dado un modelo matemático, y a través de una herramienta matemática, resolver los SED y obtener resultados.

***ResultadoSimulación**:* Son los resultados que se obtienen al realizarle una simulación a un modelo matemático.

***ComunicaciónOctave**:* Representa el traductor que envía la información hacia el asistente matemático Octave.

***ControlOctave**:* Controla todas las funcionalidades del asistente matemático Octave.

***Rutina LSode**:* Rutina perteneciente al asistente matemático Octave.

***Rutina Ode23**:* Rutina perteneciente al asistente matemático Octave.

***Rutina Ode45**:* Rutina perteneciente al asistente matemático Octave.

***Rutina Ode78**:* Rutina perteneciente al asistente matemático Octave.

El **Modelo de Dominio** correspondiente al sistema es el siguiente:

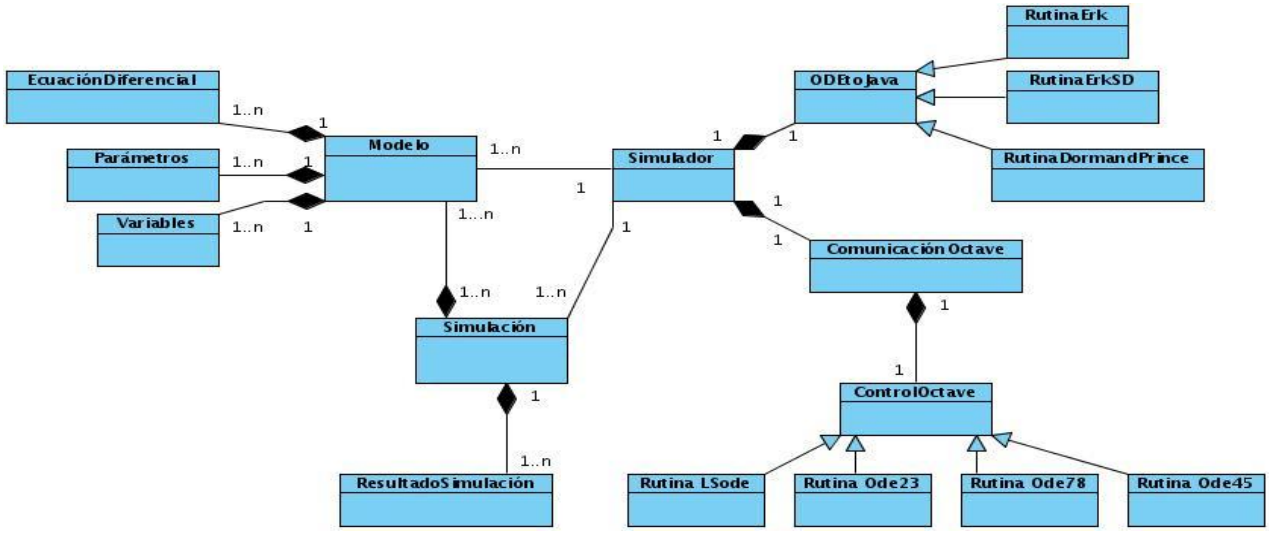


Figura 2.1 Modelo del Dominio

2.3 Actores del Sistema

El actor de un sistema representa a la persona o ente que inicia las acciones en un sistema determinado, además se puede especificar que 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.

Además los actores del sistema:

• No son parte de él.

• Pueden intercambiar información con él.

• Pueden ser un recipiente pasivo de información (42).

En este caso el actor del sistema es el investigador, el cual representa a la persona que introducirá, cargará, modelará o modificará los datos del modelo matemático y procederá a realizar la simulación.

Actores del Sistema	Descripción
Investigador	Representa la persona que interactúa con la aplicación y que realiza la simulación

Tabla 2.1 Actores del Sistema.

2.4 Requisitos Funcionales

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

El sistema a desarrollar debe cumplir los siguientes requisitos funcionales:

R1. Editar Preferencias.

R2. Realizar Simulación.

R2.1 Realizar Simulación utilizando la librería ODEtoJava.

R2.2 Realizar Simulación utilizando el asistente matemático Octave.

A continuación se describirá brevemente cada uno de los requisitos funcionales definidos en este capítulo:

El requisito **R1. Editar Preferencias** define las preferencias del sistema, para ello el Investigador debe seleccionar la herramienta matemática con la cuál simulará los modelos matemáticos, dentro de las que cabe mencionar: MatLab, ODEtoJava y Octave, además de elegir el directorio donde se encontrará la herramienta y el servidor Grid a utilizar.

El requisito **R2. Realizar Simulación** le permite al Investigador dado un modelo matemático y con ayuda de las herramientas que brinda la aplicación, realizar una simulación.

El **R2.1 Realizar Simulación utilizando la librería ODEtoJava** posibilita al Investigador después de haber cargado de la base de datos o creado un modelo matemático, realizar una simulación usando métodos matemáticos tales como DormandPrince, ErkSD y Erk, los cuales se encuentran albergados en la librería ODEtoJava.

Con el **R2.2 Realizar Simulación utilizando el asistente matemático Octave** el usuario final, en este caso el Investigador, podrá realizar simulaciones usando el asistente matemático Octave, el cual cuenta con varias rutinas como son: LSode, Ode23, Ode45 y Ode78.

2.5 Requisitos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del

producto, pues si se conoce que el mismo cumple con toda la funcionalidad requerida, las propiedades no funcionales, como cuán usable, seguro, conveniente y agradable sea, pueden marcar la diferencia entre un producto bien aceptado y uno con poca aceptación.

Existen múltiples categorías para clasificar a los requerimientos no funcionales, siendo las siguientes representativas de un conjunto de aspectos que se deben tener en cuenta, aunque no limitan a la definición de otros (42).

- Requerimientos de Software.
- Requerimientos de Hardware.
- Restricciones en el Diseño y la Implementación.
- Requerimientos de Apariencia o interfaz externa.
- Requerimientos de Seguridad.
- Requerimientos de Usabilidad.
- Requerimientos de Soporte.
- Requerimientos de Configuración

De acuerdo a lo antes definido, para el sistema a implementar se han obtenido los siguientes Requisitos no Funcionales:

2.5.1 Requerimientos de Software

- La aplicación está diseñada para ser multiplataforma, pero su correcto funcionamiento no se asegura en sistemas como Solaris y MacOS.
- Debe tener instalada la máquina virtual de Java 1.6.
- Puede tener cualquier versión de Linux o Windows, como sistema operativo en las máquinas clientes.
- En cada una de las máquinas que se vayan a utilizar como clientes de la plataforma distribuida deberá estar instalado el asistente matemático Octave, el mismo debe contar con la librerías adecuadas para solucionar SED.
- Para realizar simulaciones distribuidas el cliente debe poseer un servidor Grid y una red de equipos de cómputo a su disposición.

2.5.2 Requerimientos de Hardware

- Los equipos requerirán 512 MB de memoria RAM como mínimo.
- Deberán contar con procesadores Pentium IV o superior.
- La capacidad de almacenamiento deberá ser como mínimo 5GB, debido al grueso de la información que se almacena en el Servidor de Base de Datos, aunque si el cliente y el servidor están ubicados en la misma PC, la capacidad de almacenamiento debe ser relativa a la cantidad de información que se almacenará.

2.5.3 Restricciones en el Diseño y la Implementación

- Como lenguaje de programación Java.
- Como IDE de desarrollo NetBeans.
- Como herramienta CASE Visual Paradigm.
- Como gestor de Base de Datos PostgreSQL.

2.5.4 Requerimientos de Apariencia o Interfaz Externa

- La interfaz de la aplicación deberá ser fácil de entender y manejar, así como sencilla y amigable para todo aquel usuario que la pueda consultar.

2.5.5 Requerimientos de Seguridad

- Confidencialidad: Para tener acceso a la aplicación se requiere de un usuario y la contraseña correspondiente.
- Integridad: Se protegerá la información referente a la aplicación, para evitar posibles copias de códigos u otro tipo de datos.
- Disponibilidad: Si se posee los permisos necesarios para acceder a la aplicación, dígase user y password, se podrá tener el control de la información sin ningún tipo de obstáculo.

2.5.6 Requerimientos de Usabilidad

- El investigador podrá con el uso de la aplicación realizar simulaciones distribuidas o locales a través de una o más herramientas matemáticas, ya que tendrá la opción de una herramienta privativa y de una libre, posibilitándole opciones cuando haga uso de la misma.

2.5.7 Requerimientos de Soporte

- Una vez concluido el software se le dará una continuación con el mantenimiento del mismo.
- Se le harán las pruebas necesarias para comprobar su correcto funcionamiento.
- Se prestará el servicio de instalación y configuración del mismo.
- Se debe conectar el módulo con los restantes que conforman la plataforma.

2.5.8 Requerimientos de Configuración

- Si el simulador se usa sobre un sistema operativo libre, dígase cualquier versión de GNU/Linux, el sistema es capaz de configurar el fichero de dirección de funciones de Octave, adicionándole el path del directorio que posee la funciones que contienen el SED a simular. Generalmente el path de este directorio será /directorio_simulador/OctaveFunctions. Este fichero de configuración necesita para ser editado privilegios de administración, por tanto, si las simulaciones se realizan usando Octave, la aplicación debe ser ejecutada en modo superusuario.

2.6 Casos de Uso del Sistema

Como casos de uso del sistema se definen los siguientes:

1- Editar Preferencias.
2- Realizar Simulación.
3- Realizar Simulación utilizando la librería ODEtoJava.
4- Realizar Simulación utilizando el asistente matemático Octave.

2.7 Diagrama de Casos de Uso del Sistema

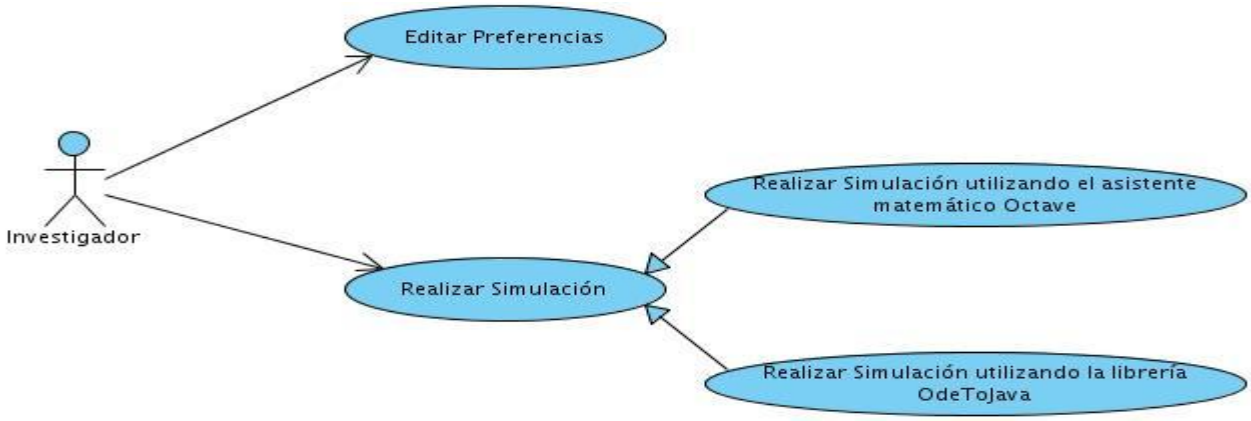


Figura 2.2 Diagrama de Casos de Uso del Sistema.

2.8 Descripción de los Casos de Uso del Sistema

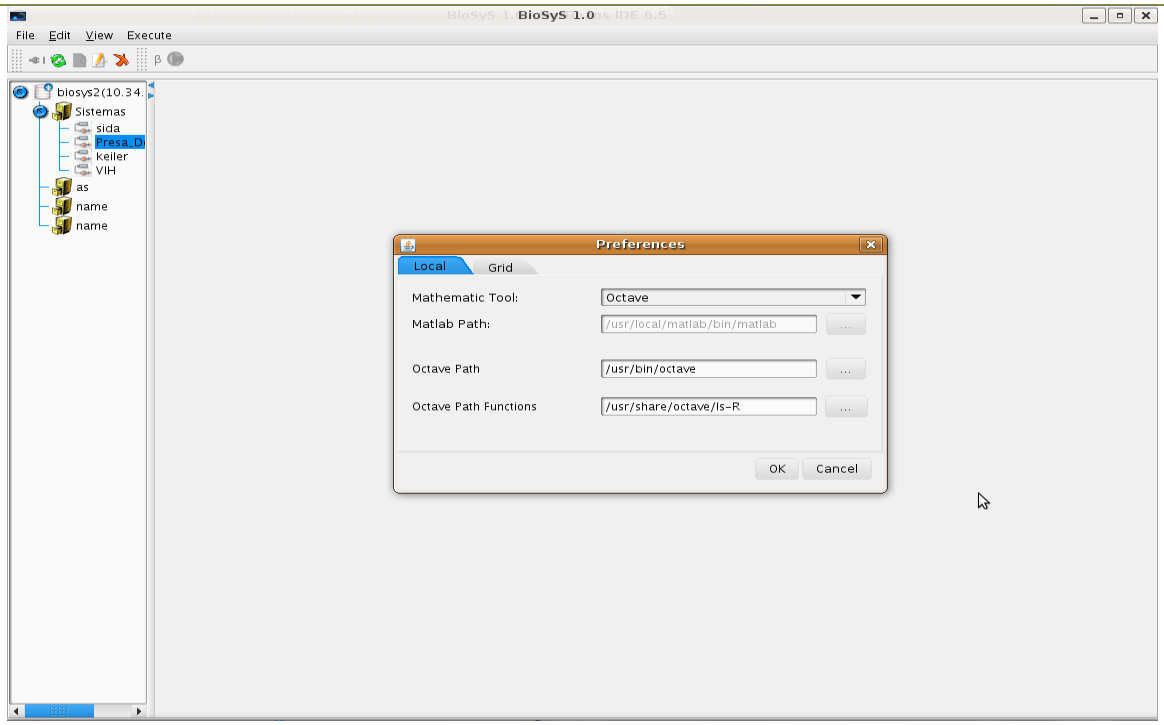
A continuación se realiza una descripción detallada de cada caso de uso involucrado en el sistema.

2.8.1 Descripción del Caso de Uso Editar Preferencias

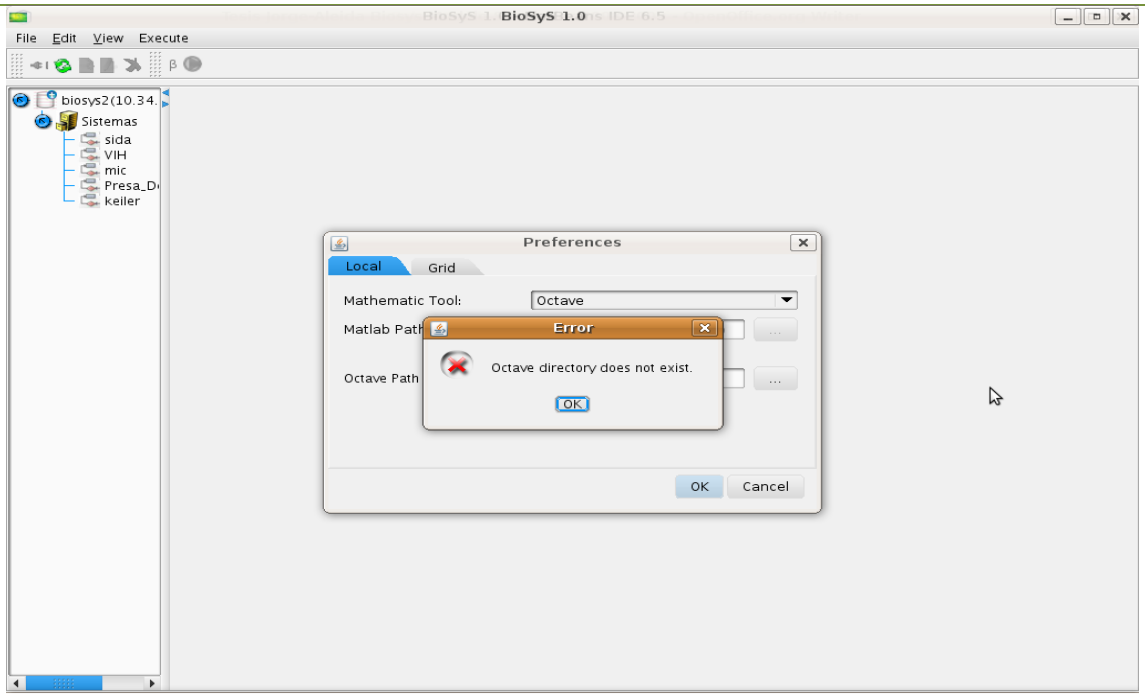
En este caso de uso el Investigador decide definir las preferencias del sistema, para ello debe seleccionar la herramienta matemática con la cual simulará los modelos matemáticos, contará con tres opciones: MatLab, Octave y ODEtoJava. Se definirá el directorio donde se encontrará la herramienta y se definirá el servidor Grid a utilizar.

Caso de Uso:	editar preferencias
Actores:	investigador(inicia)
Resumen:	El investigador seleccionará la opción que representa la herramienta matemática que desea usar para simular, contara con tres opciones MatLab, Octave y ODEtoJava. Además de elegir el servidor Grid a utilizar.
Precondiciones:	El investigador debe haberse autenticado en la aplicación.
Referencias	R1.
Prioridad	Crítico.
Flujo Normal de Eventos	
Sección "Editar Preferencias"	
Acción del Actor	**Respuesta del Sistema**
1. El Investigador selecciona la opción "Preferences" del menú del sistema.	2. El sistema muestra la interfaz correspondiente a las preferencias.
3. El Investigador puede en la ventana correspondiente a las preferencias, en la pestaña "Local", elegir la herramienta	4. El sistema verifica que los datos entrados estén correctos.

matemática a usar: Mathematic Tool: • MatLab. • Octave. • ODEtoJava. Y además llenar el campo "Path" correspondiente al directorio de la herramienta que usará. El investigador en la pestaña "Grid" podrá llenar los siguientes campos: • Introducir el IP del servidor Grid en la opción "Server". • Introducir el puerto socket a utilizar en la opción "Socket Port". • Introducir el puerto RMI a utilizar en la opción "RMI Port". • Selecciona la herramienta matemática a utilizar: MatLab, Octave u ODEtoJava en la opción "Mathematic Tool". • Introducir las PC's cliente a utilizar (Opcional) en la opción "Clients".	
	5. El sistema almacena los datos de las preferencias y finaliza el CU.
Prototipo de Interfaz	



Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
	4.1 Si los datos no son correctos, el sistema muestra un mensaje de error y finaliza el CU.
Prototipo de Interfaz	



Poscondiciones	Las preferencias serán almacenadas en el sistema.

2.8.2 Descripción del Caso de Uso Realizar Simulación

Después de haber cargado de la base de datos o creado un modelo matemático, este caso de uso permite al Investigador realizar una simulación.

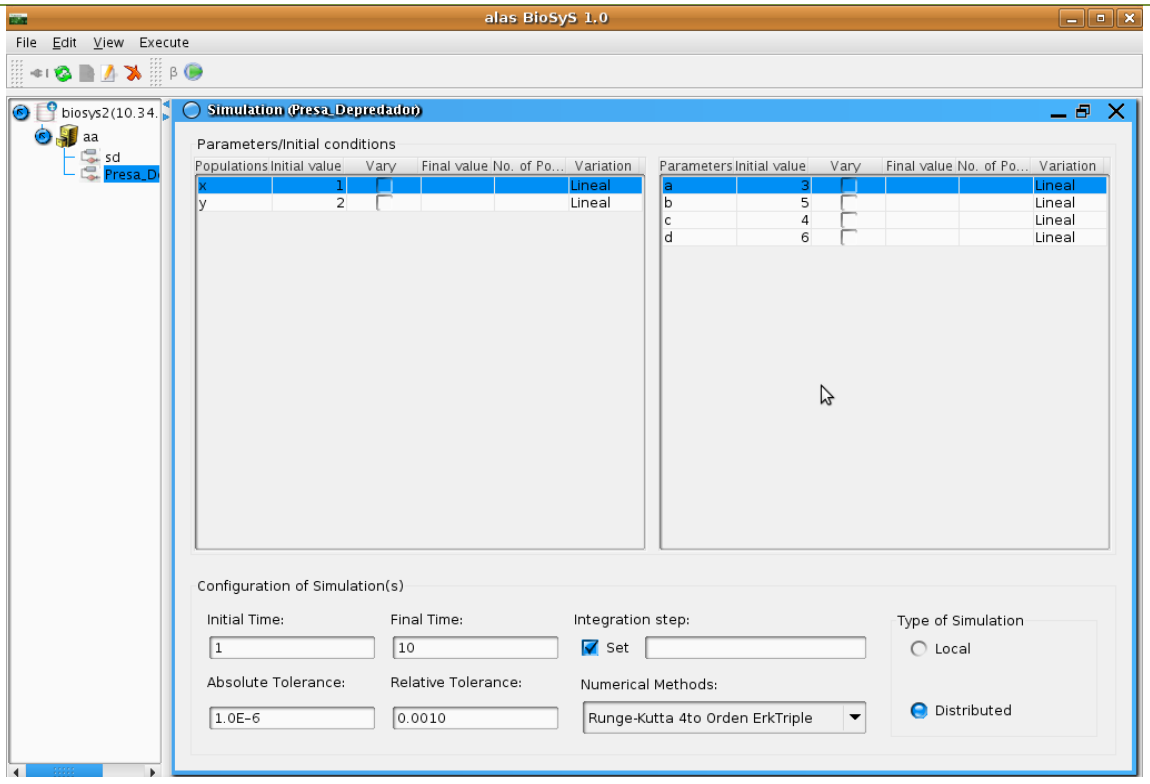
Caso de Uso:	realizar simulación
Actores:	investigador(inicia)
Resumen:	El Investigador podrá someter un modelo a una simulación, mediante diferentes métodos de simulación, pudiendo obtener una o muchas simulaciones.
Precondiciones:	El investigador debe haberse autenticado en la aplicación. El Investigador debe poseer un modelo matemático. El Investigador debe haber editado la sección preferencias.

<table>
<tr><td>Referencias</td><td colspan="2">R2. R1</td></tr>
<tr><td>Prioridad</td><td colspan="2">Crítico.</td></tr>
<tr><td colspan="3">Flujo Normal de Eventos</td></tr>
<tr><td colspan="3">Sección “Proceso inicial de la simulación”</td></tr>
<tr><td colspan="2">Acción del Actor</td><td>Respuesta del Sistema</td></tr>
<tr><td colspan="2">1. El Investigador elige el modelo matemático que desea simular.</td><td></td></tr>
<tr><td colspan="2">2. El Investigador elige la opción “Simulate Model” en el modelo matemático elegido.</td><td>3. El sistema muestra la interfaz de simulación, con los siguientes valores:
• Valores de parámetros y condiciones iniciales en Parameters/Inicial Conditions.
• Inicial Time.
• Final Time.
• Relative Tolerance.
• Absolute Tolerance.
• Integration step.
• Type of simulation.
“Local” para realizar las simulaciones localmente.
“Distribuida” para realizar simulaciones distribuidas.
• Numerical Method.

En caso de que haya editado las preferencias:
Para “Octave” los métodos matemáticos son:
▪ LSode.</td></tr>
</table>

	▪ Ode23. ▪ Ode45. ▪ Ode78. Para "ODEtoJava" los métodos numéricos son: • DormandPrince. • ErkSD. • Erk. • ErkTriple.

Prototipo de Interfaz

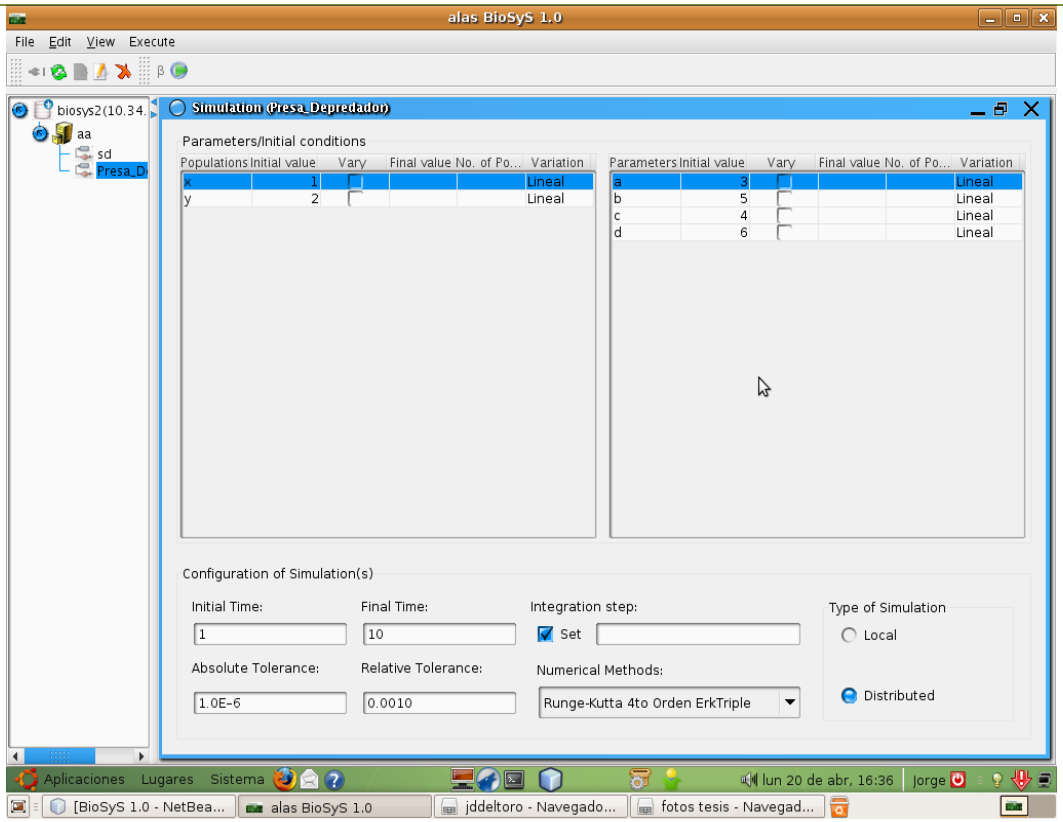


Flujos Alternos

Acción del Actor	Respuesta del Sistema
	3.1 En caso de que el Investigador no haya

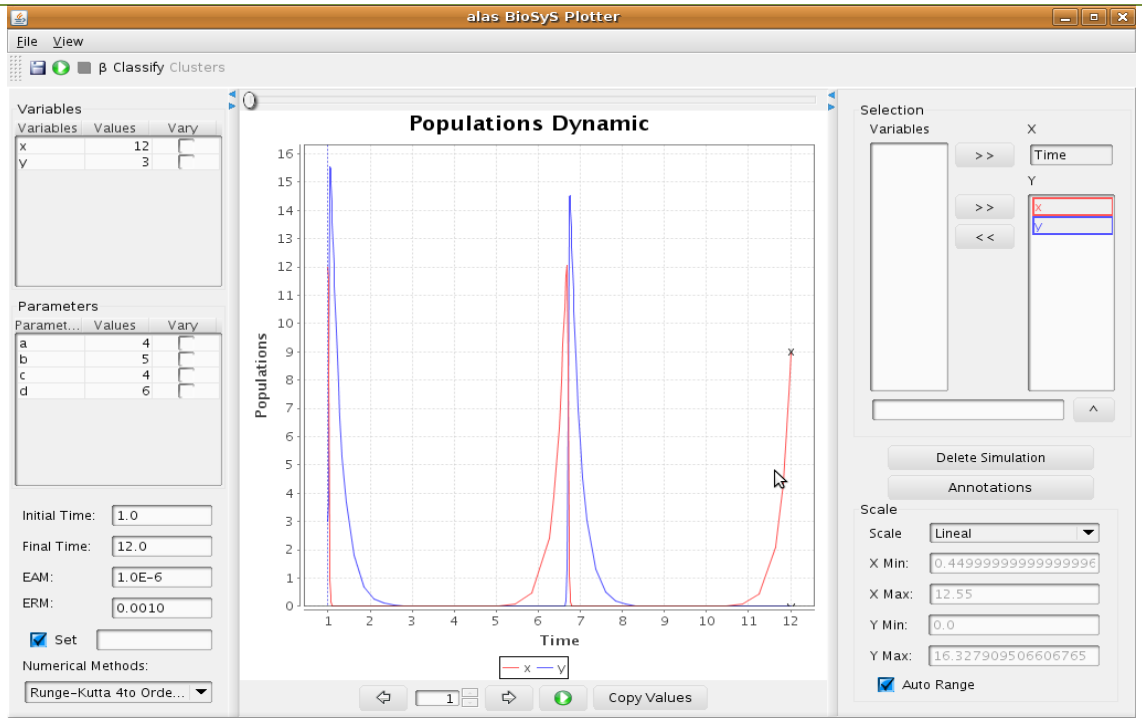
	editado las preferencias aparecerá por defecto la herramienta ODEtoJava.

Prototipo de Interfaz



Sección “Desarrollo de la simulación”

Acción del Actor	Respuesta del Sistema
	1. Cuando la herramienta guardada en las preferencias sea ODEtoJava ver CU Realizar Simulación utilizando la librería ODEtoJava, sección “Desarrollo de la simulación”. Cuando la herramienta guardada en las preferencias sea Octave ver CU Realizar Simulación utilizando el asistente

	matemático Octave, sección “Desarrollo de la simulación”.
Sección “Resultados de la simulación”	
Acción del Actor	**Respuesta del Sistema**
	9. El sistema muestra los resultados de la simulación. 10. El sistema almacena los datos de la simulación en la base de datos y finaliza el CU.
Prototipo de Interfaz	
	
Poscondiciones	Se obtendrá un resultado de la simulación. El resultado de la simulación se almacenará en la base de datos.

2.8.3 Descripción del Caso de Uso Realizar Simulación utilizando la librería ODEtoJava

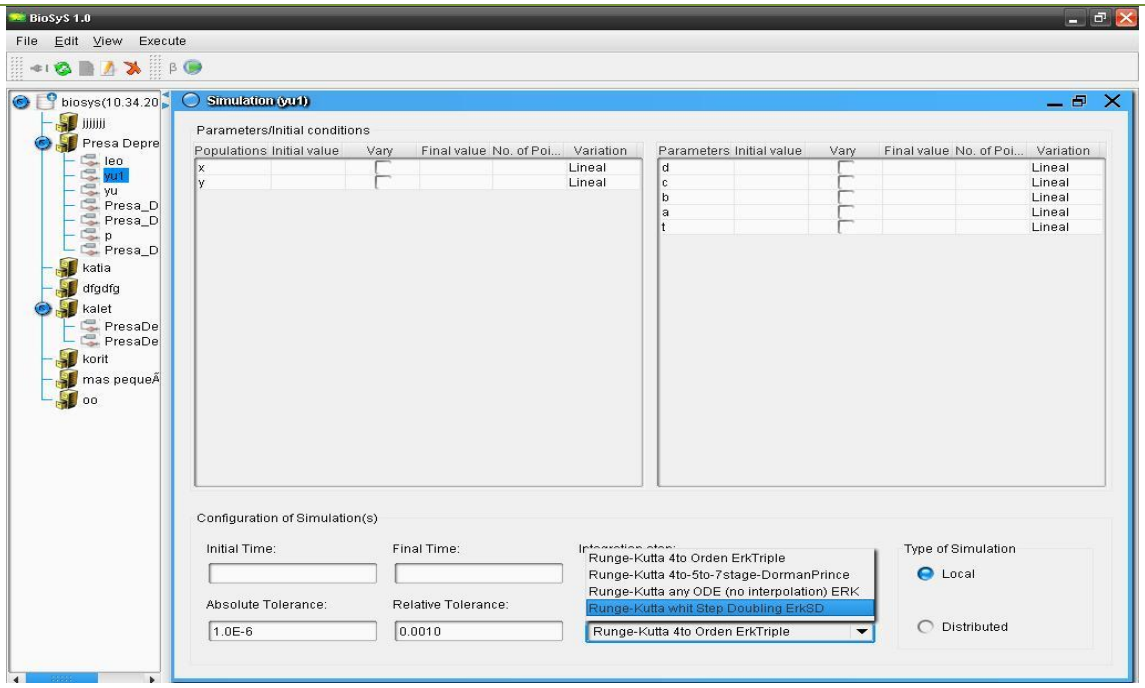
Después de haber cargado de la base de datos o creado un modelo matemático, este caso de uso permite al Investigador realizar una simulación usando métodos matemáticos que se encuentran albergados en la librería ODEtoJava.

Caso de Uso:	Realizar simulación utilizando la librería ODEtoJava.
Actores:	investigador(inicia)
Resumen:	El Investigador podrá someter un modelo a una simulación, mediante diferentes métodos matemáticos nativos de ODEtoJava.
Precondiciones:	El Investigador debe haberse autenticado en la aplicación. El Investigador debe poseer un modelo matemático. El Investigador debe haber editado la sección preferencias y haber seleccionado como herramienta matemática para simular ODEtoJava.
Referencias	R2.1.R2
Prioridad	Crítico.

Flujo Normal de Eventos	
Sección "Proceso inicial de la simulación"	
Acción del Actor	**Respuesta del Sistema**
	1. Ver CU Realizar Simulación, sección "Proceso inicial de la simulación".
Sección "Desarrollo de la simulación"	
Acción del Actor	**Respuesta del Sistema**
4. El investigador introduce los valores necesarios para realizar la simulación y selecciona uno de los cuatro métodos numéricos albergados en la librería ODEtoJava.	5. El sistema verifica la validez de los datos.
	6. El sistema guarda los valores definidos.

7. El Investigador elige comenzar simulación (Botón correspondiente).	8. El sistema comienza a simular el modelo usando el método numérico seleccionado.
	9. El sistema muestra la barra de progreso que representa el estado de la simulación en ese momento, dando la posibilidad de pausar o detener la simulación.

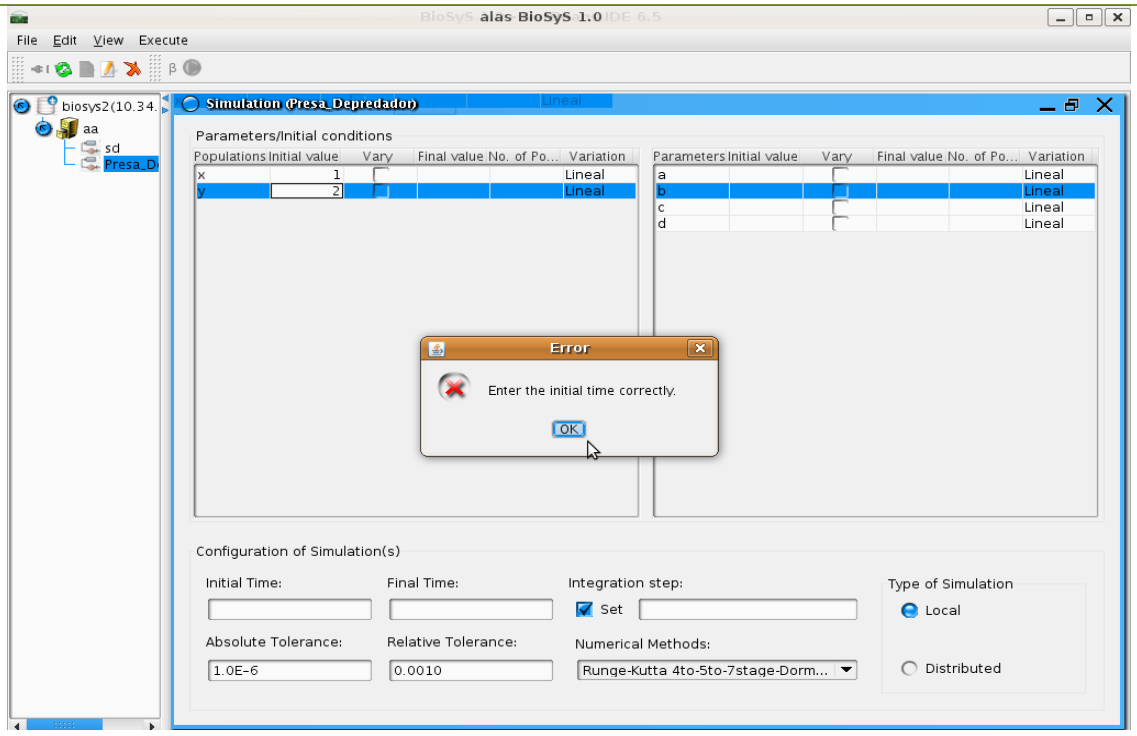
Prototipo de Interfaz



Flujos Alternos

Acción del Actor	Respuesta del Sistema
	5.1 Si los datos entrados no son correctos el sistema lanzará un mensaje de error y finaliza el CU.

Prototipo de Interfaz



Sección "Resultados de la simulación"	
Acción del Actor	**Respuesta del Sistema**
	1. Ver CU Realizar Simulación, sección "Resultados de la simulación".
Poscondiciones	Se obtendrá un resultado de la simulación. El resultado de la simulación se almacenará en la base de datos.

2.8.4 Descripción del Caso de Uso Realizar Simulación utilizando el asistente matemático Octave

Después de haber cargado de la base de datos o creado un modelo matemático, este caso de uso permite al Investigador realizar una simulación usando el asistente matemático Octave, el cual cuenta con 4 rutinas para esta operación.

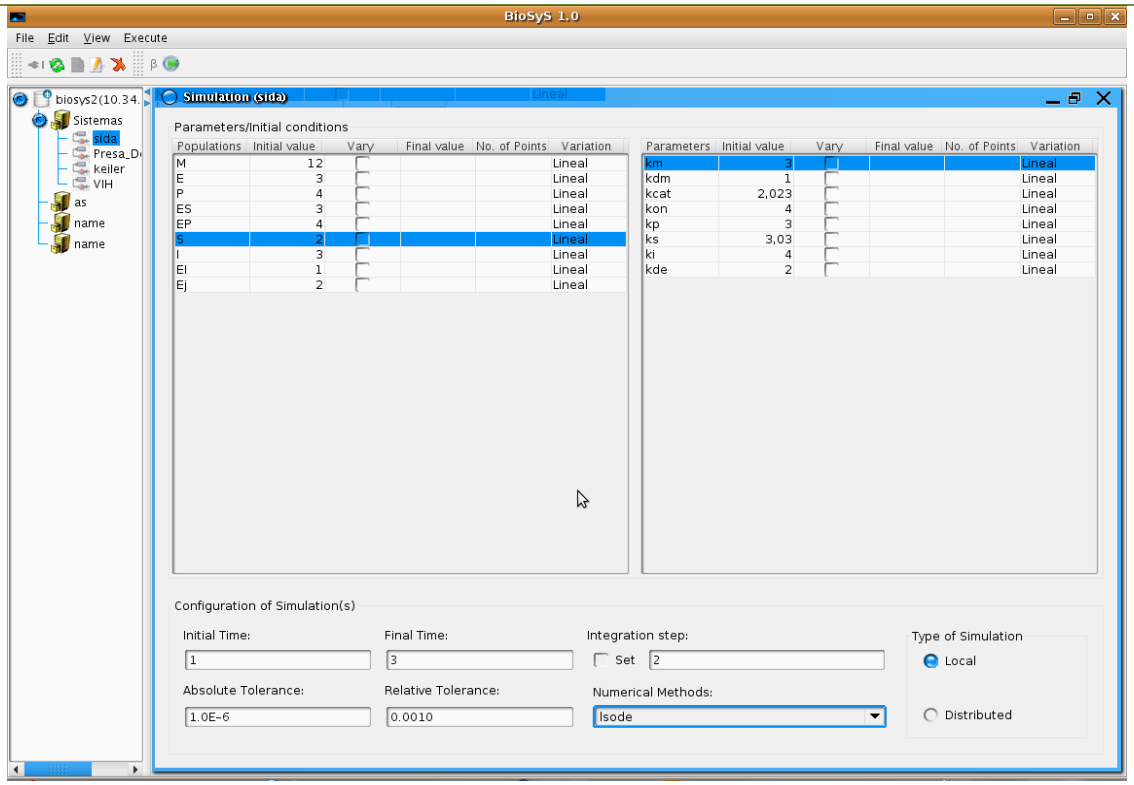
Caso de Uso:	Realizar simulación utilizando el asistente matemático Octave.

Actores:	investigador(inicia)
Resumen:	El Investigador podrá someter un modelo a una simulación, usando al asistente matemático Octave.
Precondiciones:	El Investigador debe haberse autenticado en la aplicación. El Investigador debe poseer un modelo matemático. El Investigador debe haber editado la sección preferencias y haber seleccionado como herramienta para simular Octave.
Referencias	R2.2.R2
Prioridad	Crítico.

Flujo Normal de Eventos	
Sección "Proceso inicial de la simulación"	
Acción del Actor	**Respuesta del Sistema**
	1. Ver CU Realizar Simulación, sección "Proceso inicial de la simulación".
Sección "Desarrollo de la simulación"	
Acción del Actor	**Respuesta del Sistema**
4. El investigador introduce los valores necesarios para realizar la simulación y selecciona uno de los 4 métodos matemáticos que nos brinda Octave para simular.	5. El sistema verifica la validez de los datos.
	6. El sistema guarda los valores definidos.
7. El Investigador elige la opción que le permite comenzar a realizar una simulación.	8. El sistema comienza a simular el modelo matemático usando el método de la herramienta Octave que el investigador seleccionó.
	9. El sistema muestra una barra que indica el progreso de la simulación que se está

	desarrollando en ese momento, dando la posibilidad de pausar o detener el proceso.

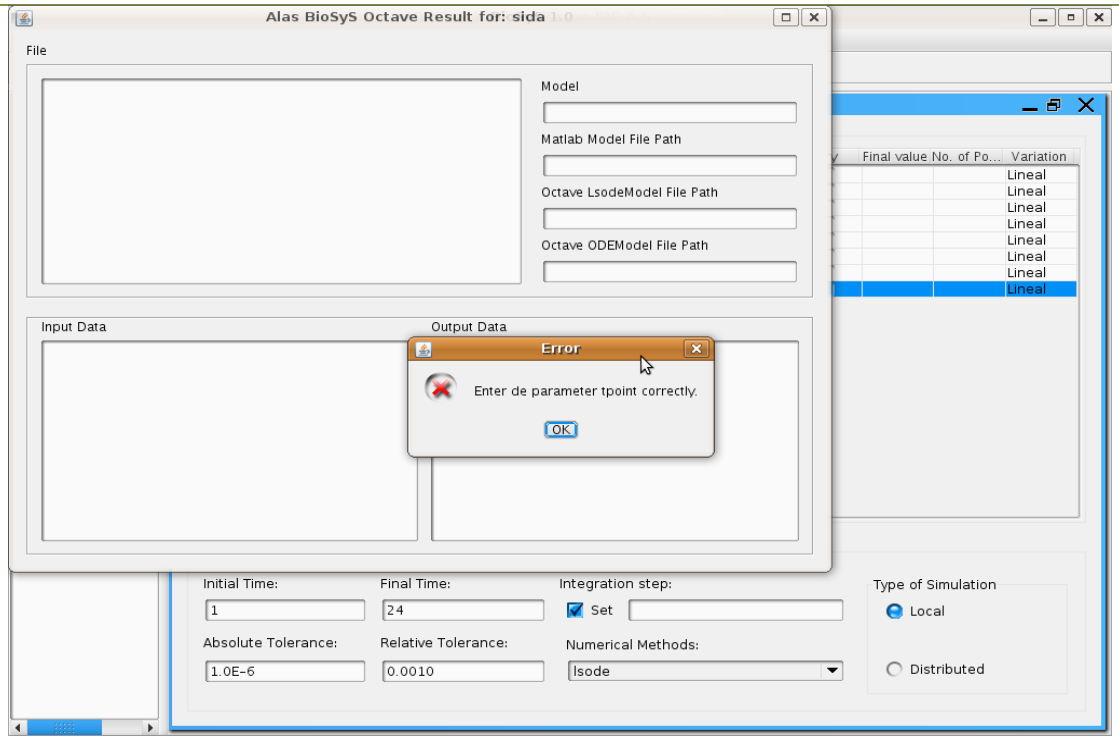
Prototipo de Interfaz



Flujos Alternos

Acción del Actor	Respuesta del Sistema
	5.1 Si los datos introducidos por el investigador no son los correctos, el sistema mostrará el mensaje de error correspondiente y finaliza el CU.

Prototipo de Interfaz



Sección “Resultados de la simulación”	
Acción del Actor	**Respuesta del Sistema**
	1. Ver CU Realizar Simulación, sección “Resultados de la simulación”.
Poscondiciones	Se obtendrá un resultado de la simulación. El resultado de la simulación se almacenará en la base de datos.

2.9 Conclusiones del Capítulo

Este capítulo se basó en una amplia descripción del sistema que se quiere automatizar. Para ello fue necesario definir el actor del sistema, así como los casos de uso implicados en el mismo. Para una mayor comprensión se describió detalladamente cada caso de uso, además de confeccionar el Diagrama de Casos de Uso del Sistema. Se tuvo en cuenta las cualidades y funcionalidades requeridas para el software a implementar, definidas como los requisitos funcionales y no funcionales que aparecen en esta sección.

Capitulo 3: Análisis y Diseño del Sistema.

En este capítulo se describen detalladamente los patrones que fueron utilizados en el desarrollo de la aplicación, dígase los patrones de arquitectura y de diseño que se utilizaron. Además se definen los paquetes y subsistemas asociados y se construyen diagramas como el Diagrama de Paquetes, Diagramas de Clases del Diseño y los Diagramas de interacción para esta etapa de desarrollo del sistema, los cuales ayudan a una mayor comprensión del funcionamiento interno del Simulador.

3.1 Descripción de los Patrones de Arquitectura y Diseño

3.1.1 Patrones de Arquitectura

Se utiliza el patrón arquitectónico Modelo-Vista-Controlador (MVC), el cual ayuda a definir en la aplicación tres paquetes importantes:

Modelo: Aquí se reúne la clase encargada de guardar y almacenar los datos persistentes, así como de gestionar todo tipo de información relacionada con el sistema, ésta clase es la BSPreferencia.

Vista: Este paquete contiene las clases que permiten la interacción con los usuarios, la visualización de los datos manejables en la aplicación, así como la apariencia y el formato establecido para las ventanas, menúes y reportes de la aplicación. Dentro de las clases que se encuentran en este paquete se puede mencionar: BioSyS, BSVistaPreferencia, VistaSimulacion y BSVistaProgreso.

Controlador: Cuenta con las clases encargadas de gestionar el procesamiento, de acuerdo a las peticiones que se les hacen desde las clases interfaces, se hacen notar las siguientes: BioSySControl, ControlPreferencia, VistaSimulacionManager, BSControlOctave, BSControlProgreso, BSSimular, BSControlJopas, BSControlSimulaciones y la librería BSSimulacion.

3.1.2 Patrones de Diseño

Para el desarrollo del sistema se encontró la necesidad de usar los patrones de diseño Gof y Grasp:

Patrones Gof

- ***Facade*** *(Fachada):* Este patrón proporciona una interfaz unificada para un conjunto de interfaces, en la aplicación la interfaz BioSyS posibilita el acceso a otras interfaces como a BSVistaPreferencia y VistaSimulacion. (*Ver Anexo 1*)

- ***Strategy*** *(Estratégico):* Este patrón permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución, en este caso se tiene a la clase VistaSimulacion, la cual cuenta con más de un constructor y escoge en tiempo de ejecución cuál utilizar. (*Ver Anexo 2*)

- ***Observer*** *(Observador)*: Este patrón se utiliza cuando se muestra el estado del progreso de las simulaciones que la aplicación es capaz de realizar, con los cambios se actualiza automáticamente la interfaz BSVistaProgreso. *(Ver Anexo 3)*

Patrones Grasp

- ***Experto****:* Las clases que se utilizan contienen información, por tanto cada una es responsable de velar por el acceso que se tenga a dicha información. Este patrón se ve muy bien reflejado en la clase BSControlOctave, ya que esta es la encargada de manejar todos los procesos del asistente y de la información que él requiere para realizar las simulaciones, a su vez solo se puede acceder a esta información mediante instancias de esta clase.

- ***Creador****:* Para acceder a las funcionalidades de ciertas clases, se deben crear instancias de ellas, pero no todas tienen el privilegio de crear estos, por tanto recae una gran responsabilidad sobre las clases creadoras. En la aplicación se ejemplifica esto con la clase BiosysControl.

- ***Alta cohesión****:* Cada clase presente en el sistema tiene un volumen de información bien distribuido, de manera que la cohesión permanezca alta. No se permiten clases de la interfaz control en procesos de la simulación por ejemplo.

- ***Controlador****:* Con este patrón algunas de las clases de la aplicación son las encargadas de gestionar la información entre las clases interfaces y las clases del paquete Modelo, que son las que guardan la información persistente, entre ellas se tiene a: BioSySControl, ControlPreferencia, BSControlOctave, entre otras.

3.2 Diagramas de Clases del Diseño

Un Paquete de Diseño es un elemento agrupador, que sirve para contener a otros elementos, que pueden ser Clases u otros Paquetes y un subsistema de Diseño cumple la misma función que un paquete, pero tiene la diferencia que aquí el Todo (subsistema) es mayor que la simple suma de las

partes que lo componen, ya que estas se encuentran interrelacionadas de forma tal que puedan satisfacer ciertos servicios (43).

En el presente trabajo se definen tres paquetes necesarios para una mejor organización del sistema y un subsistema: el T-arenal, dentro de los paquetes encontramos al Paquete Modelo, el Paquete Vista y el Paquete Control. A continuación se muestra una breve descripción de cada uno:

Paquete Vista: Este paquete agrupa las clases que representan las interfaces del sistema, cada una de ellas por sí solas varían en cuanto al funcionamiento que permiten, ya que unas nos permiten las configuraciones iniciales, ya sea la entrada de datos referentes a las preferencias, la configuración de directorios de la herramienta a usar, así como la recogida de los datos necesarios para realizar una simulación.

Paquete Control: Las clases que recoge este paquete están relacionadas con el control y el flujo de la información, pues son las encargadas de gestionar los procesos de la aplicación y permiten así la comunicación entre diversas clases.

Paquete Modelo: Contiene la clase encargada de almacenar la información gestionada en el sistema.

Subsistema T-arenal: Este subsistema es el encargado de distribuir las simulaciones.

Dada las relaciones que presentan los tres paquetes y el subsistema definido, queda conformado el Diagrama de Paquetes característico:

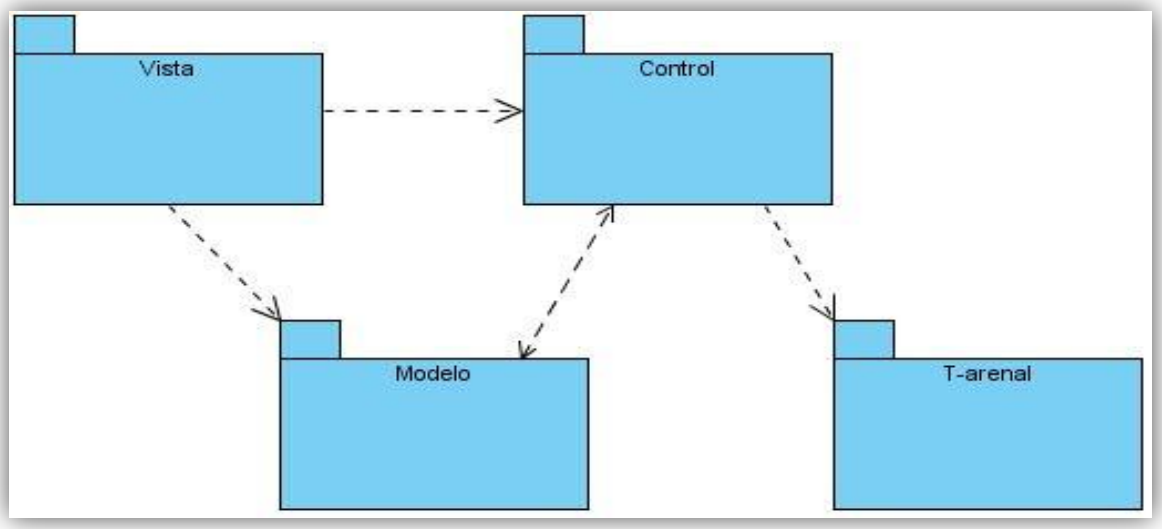


Figura 3.1 Diagrama de Paquetes.

3.2.1 Paquete Vista

BioSyS es la interfaz principal del sistema, a través de ella se puede acceder a las demás interfaces. Con la vista *BSVistaPreferencia* se editan las preferencias necesarias para realizar una simulación, aquí se definen datos como la herramienta matemática a usar, el directorio donde encontrar a dicha herramienta, entre otros, de no configurarse esta vista el sistema tomará las preferencias por defecto. A través de la interfaz *VistaSimulacion* se carga la función matemática que se simulará y el Investigador puede introducir los valores de parámetros y condiciones iniciales para simular con ODEtoJava, MatLab u Octave. La *BSVistaProgreso* es otra de las interfaces, que posibilita que se le muestre al usuario final el estado de la simulación y le da la posibilidad de pausar o detener la misma.

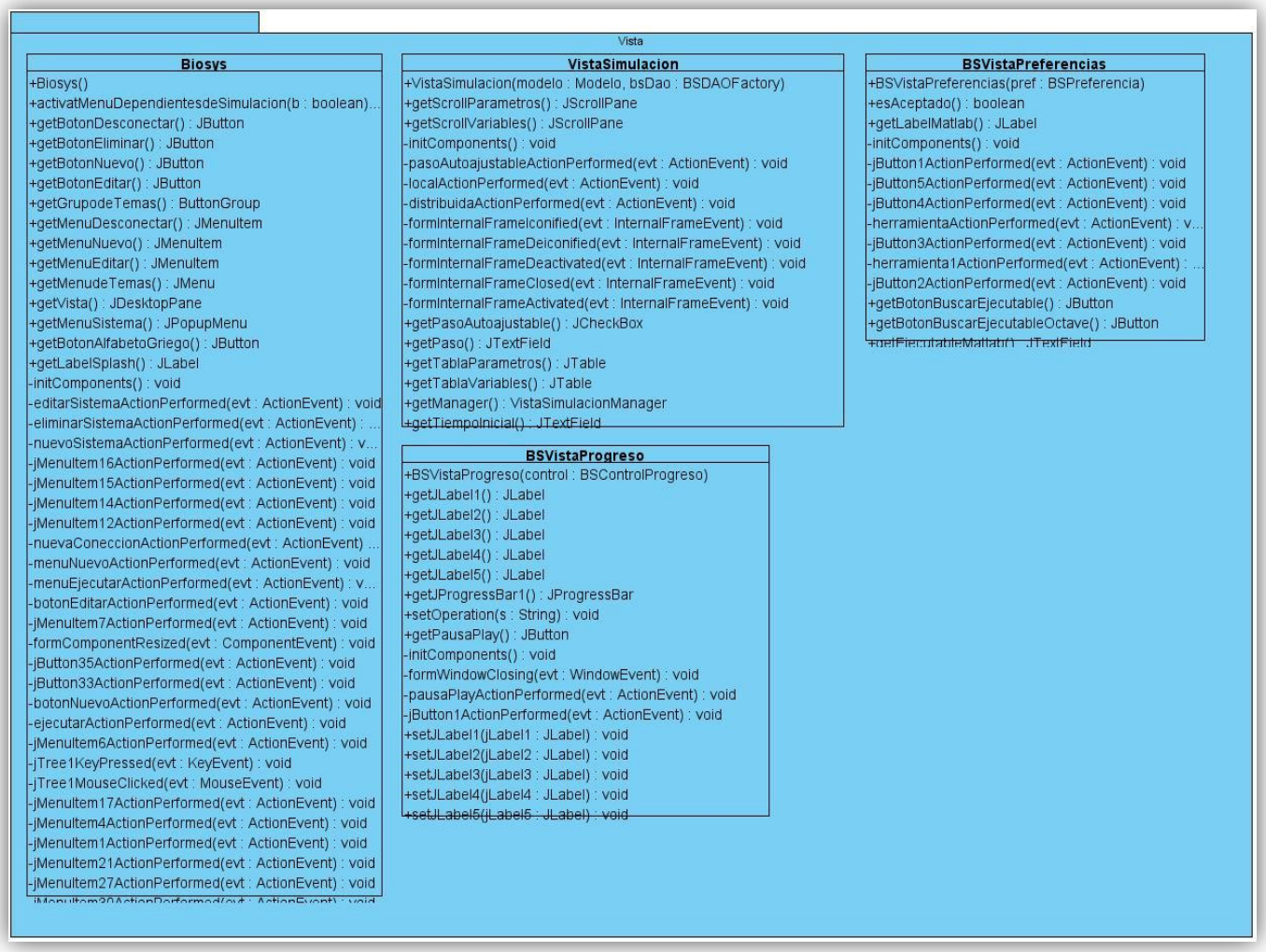


Figura 3.2 Diagrama de Clases del Diseño Paquete Vista.

3.2.2 Paquete Control

BioSySControl es la clase controladora que posibilita el flujo de información entre las clases interfaces y la clase BSPreferencia del paquete Modelo. La *ControlPreferencia* gestiona los datos relacionados con las preferencias del sistema y la clase *VistaSimulacionManager* maneja los datos tomados de las vistas de ODEtoJava y Octave para realizar las simulaciones. A su vez la *BSSimular* controla las simulaciones que realizará la aplicación con una herramienta u otra, donde la clase *BSControlOctave* es la encargada de las simulaciones con el asistente matemático Octave y el paquete *BSSimulacion* el responsable de las que se realicen con la librería ODEtoJava. La clase *BSControlJopas* permite la

comunicación con Octave. La *BSControlSimulaciones* permite insertar las simulaciones en la Base de Datos.

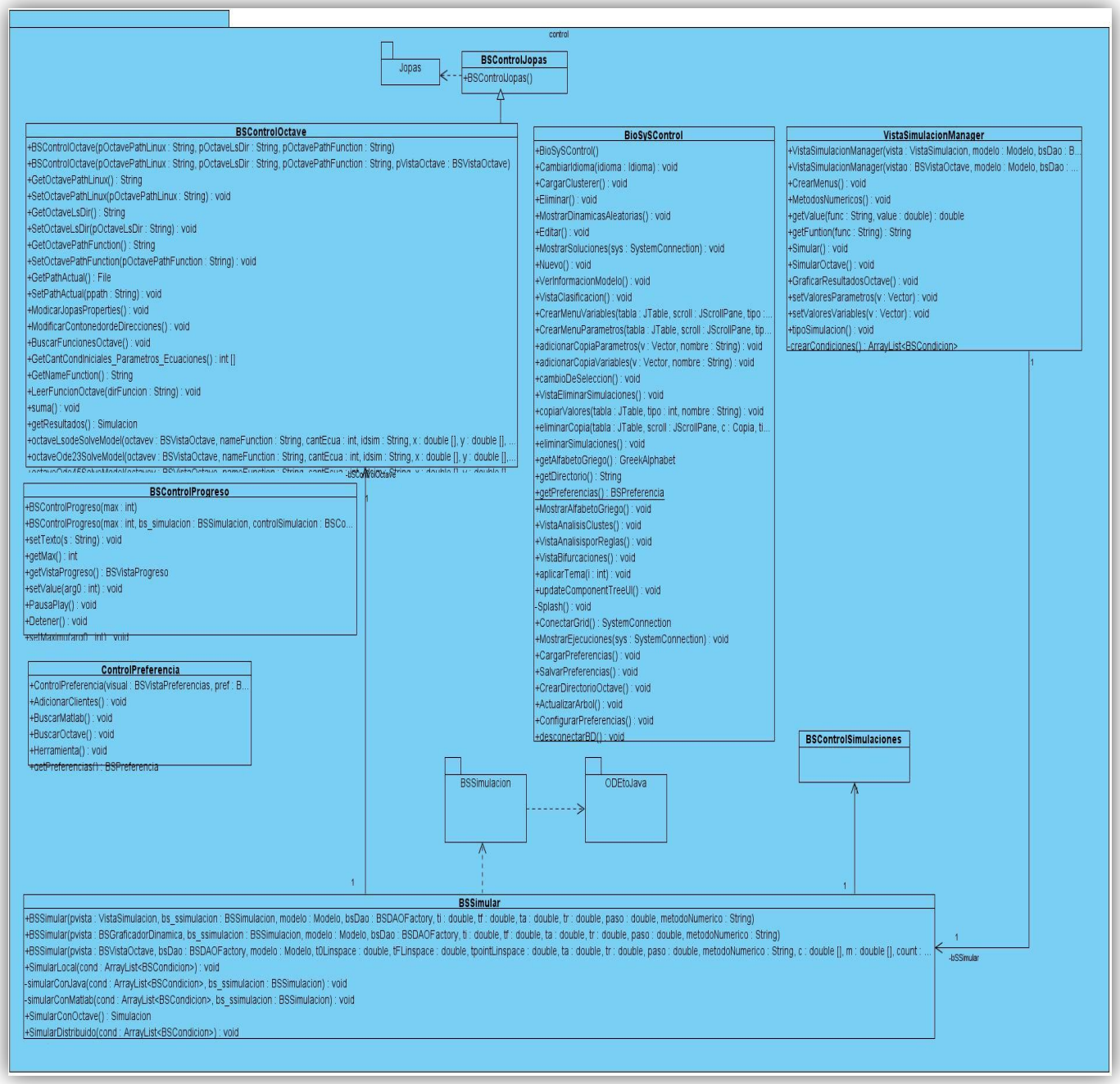


Figura 3.3 Diagrama de Clases del Diseño Paquete Control.

3.2.3 Paquete Modelo

Este paquete contiene a la clase *BSPreferencia*, la cual consta de la información referente a las preferencias que se configuran y almacenan en el sistema. La misma permite que se conozca en cada momento como debe realizarse el flujo de información entre una clase y otra.

Figura 3.4 Diagrama de Clases del Diseño Paquete Modelo.

3.2.4 Subsistema T-arenal

Este subsistema contiene varios componentes que le permiten fragmentar problemas y distribuirlos en una red de cómputo que tiene a su favor y que serán los encargados de resolver pequeñas porciones del problema original, dentro de los componentes se encuentran el *BSAlgorithm, BSDataManager* y la *Grid.*

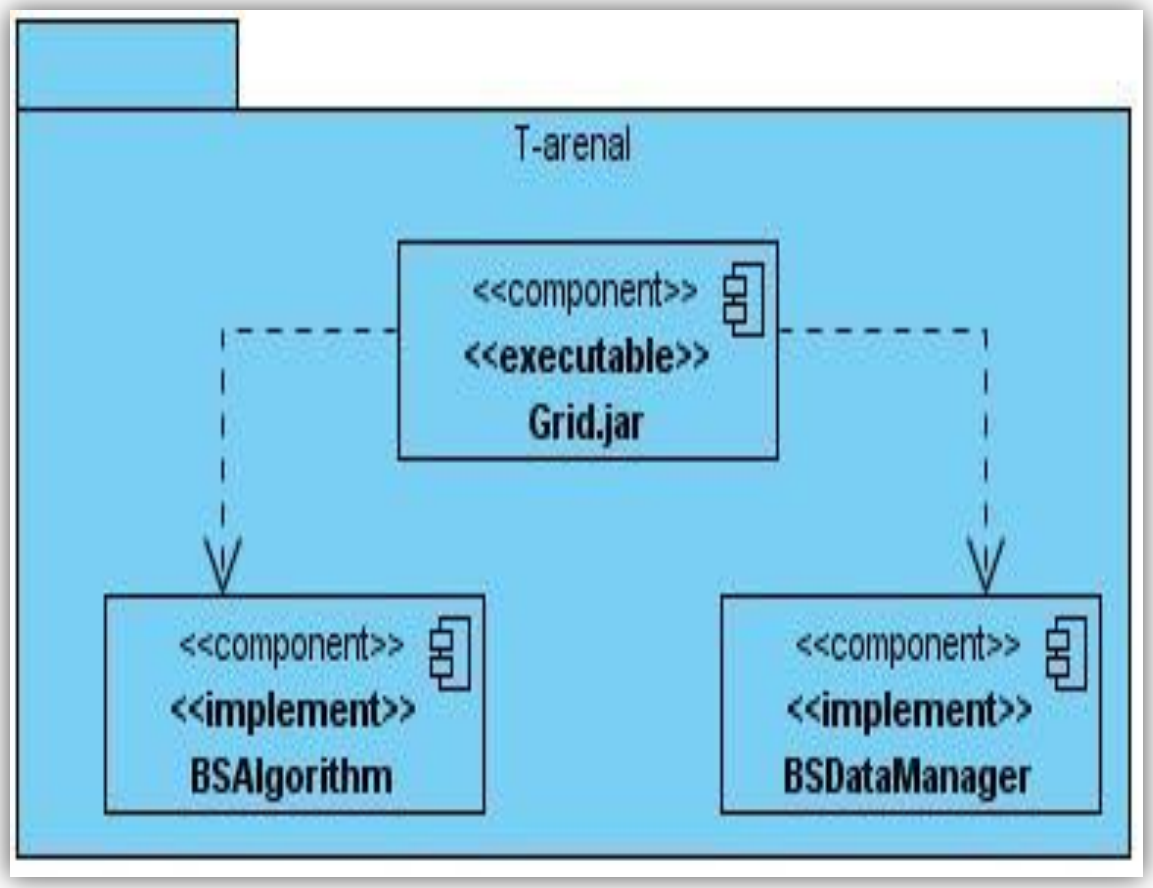


Figura 3.5 Subsistema T-arenal.

Para reflejar de una forma más definida las relaciones que existen entre las clases de cada paquete del diseño y ayudar a una mejor comprensión del usuario sobre el sistema a desarrollar, se ha confeccionado un Diagrama de Paquetes detallado, el cual se les muestra a continuación:

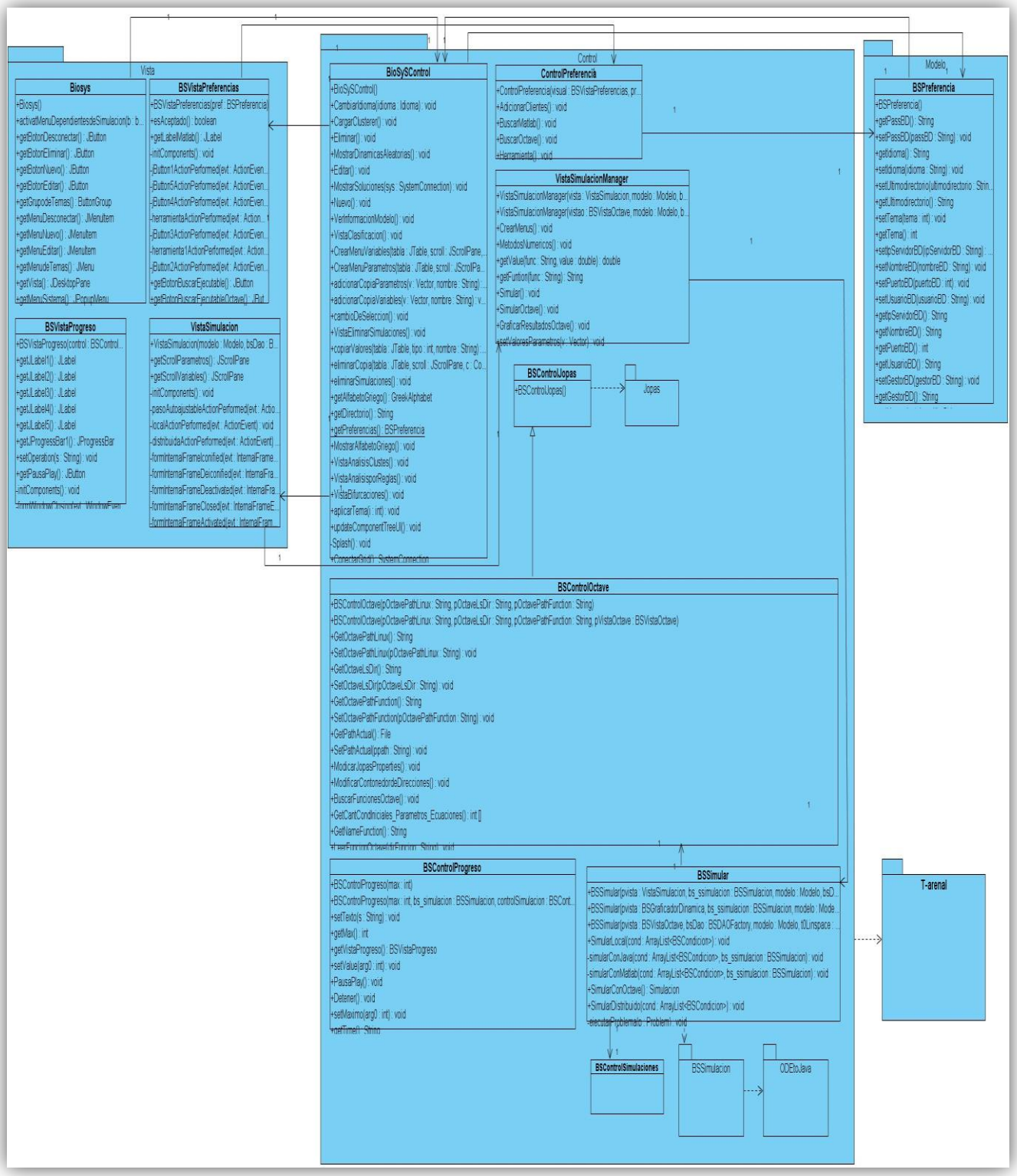


Figura 3.6 Diagrama de Paquetes detallado

3.3 Diagramas de Interacción

En estos diagramas los objetos interactúan para realizar colectivamente los servicios ofrecidos por las aplicaciones. Los diagramas de interacción muestran cómo se comunican los objetos. Existen dos tipos de diagramas de interacción:

- Diagrama de Secuencia.
- Diagrama de Colaboración.

Son útiles para la documentación de un Caso de Uso: en términos próximos al usuario y sin detallar la sincronización existente y para la representación precisa de las interacciones entre objetos.

Un diagrama de secuencia muestra la secuencia cronológica de mensajes entre objetos durante un escenario concreto (44). A continuación se realiza una breve descripción y representación de los Diagramas de Secuencia construidos:

DS Editar Preferencias:

El Investigador a la hora de editar las preferencias debe elegir en la interfaz **BioSyS** la opción correspondiente. Una vez realizada esta acción se le solicita a la clase **BiosysControl** configurar dichas preferencias a través del método **ConfigurarPreferencias()** y se le pide a **BSPreferencia** las preferencias que almacena por defecto. La controladora **BiosysControl** envía las preferencias a través de **ShowinterfazPreferencias()** a la interfaz **BSVistaPreferencia**. El Investigador edita las preferencias y la interfaz manda a la clase **ControlPreferencia** los datos editados a través de **GuardarPreferencias()**. La controladora verifica que estén correctos los datos con **VerificarDatosPreferencias()** y se los envía a **BSPreferencia**, la cual guarda los mismos con **GuardarDatos()**.

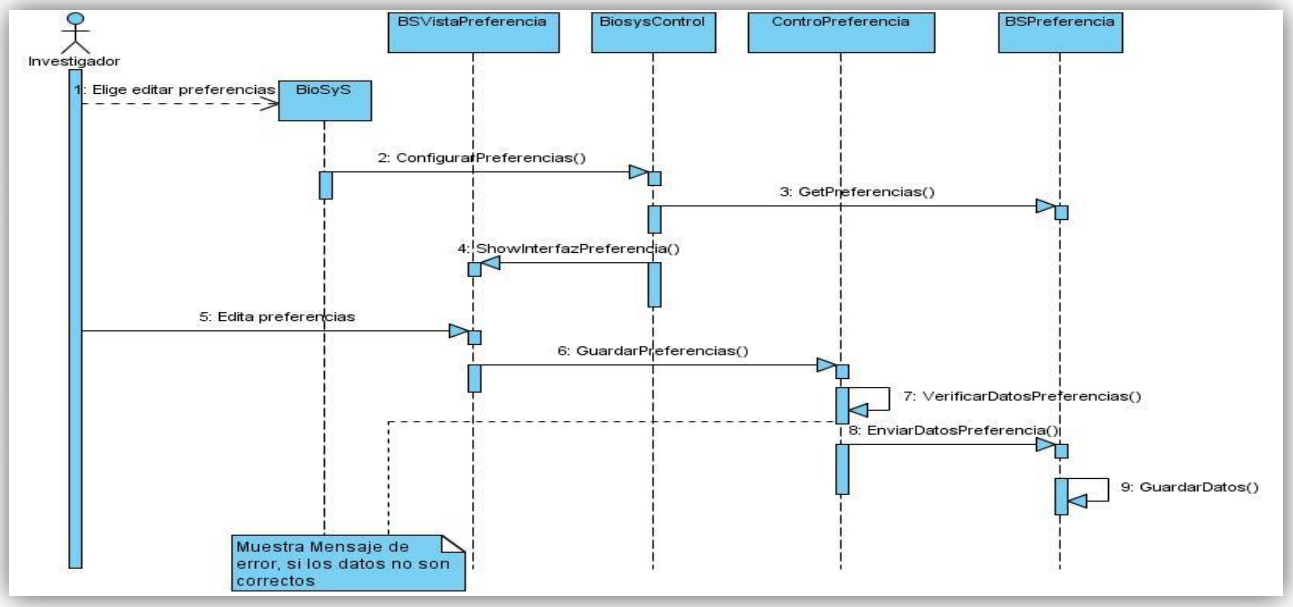


Figura 3.7 Diagrama de Secuencia Editar Preferencias.

DS Realizar Simulación utilizando la librería ODEtoJava:

El investigador elige la opción **Simular Modelo** en la interfaz **BioSyS**, seguidamente se genera un flujo de información a través del método **SelecciónModelo()**, que permite almacenar el modelo seleccionado en un objeto y se prosigue a verificar la preferencias para determinar qué tipo de simulación se realizará. Definido el tipo de simulación se muestra la vista encargada de recoger los datos, la **VistaSimulación**, mediante el método **ShowVistaSimulación().** Una vez que el investigador obtiene la vista indicada, introduce los datos de la simulación y elige simular el sistema, en este instante se invoca al método **EjecutarSimulación()** perteneciente a la interfaz principal, donde el mismo solicita a la clase control **BiosysControl**. Esta clase mediante la función **Simular()** llama al método **Ejecutar()** de la interfaz de simulación que se esté mostrando, en este caso de **VistaSimulación**, este método invoca a la clase **VistaSimulacionManager** a que tome el control de la simulación, esta construye a partir de los datos de la interfaz todos los requerimientos de la simulación y facilita esta información a la clase **BSSimular**, la misma invoca al método **SimularConJava()**. En esta instancia se resuelve el Sistema de Ecuaciones Diferenciales bajo las condiciones iniciales, se obtiene la simulación y se devuelve esta a la clase **VistaSimulacionManager**, la cual adiciona la simulación en la Base de Datos, mediante la clase **BSControlSimulaciones** y usando el método **AddSimulación()**.

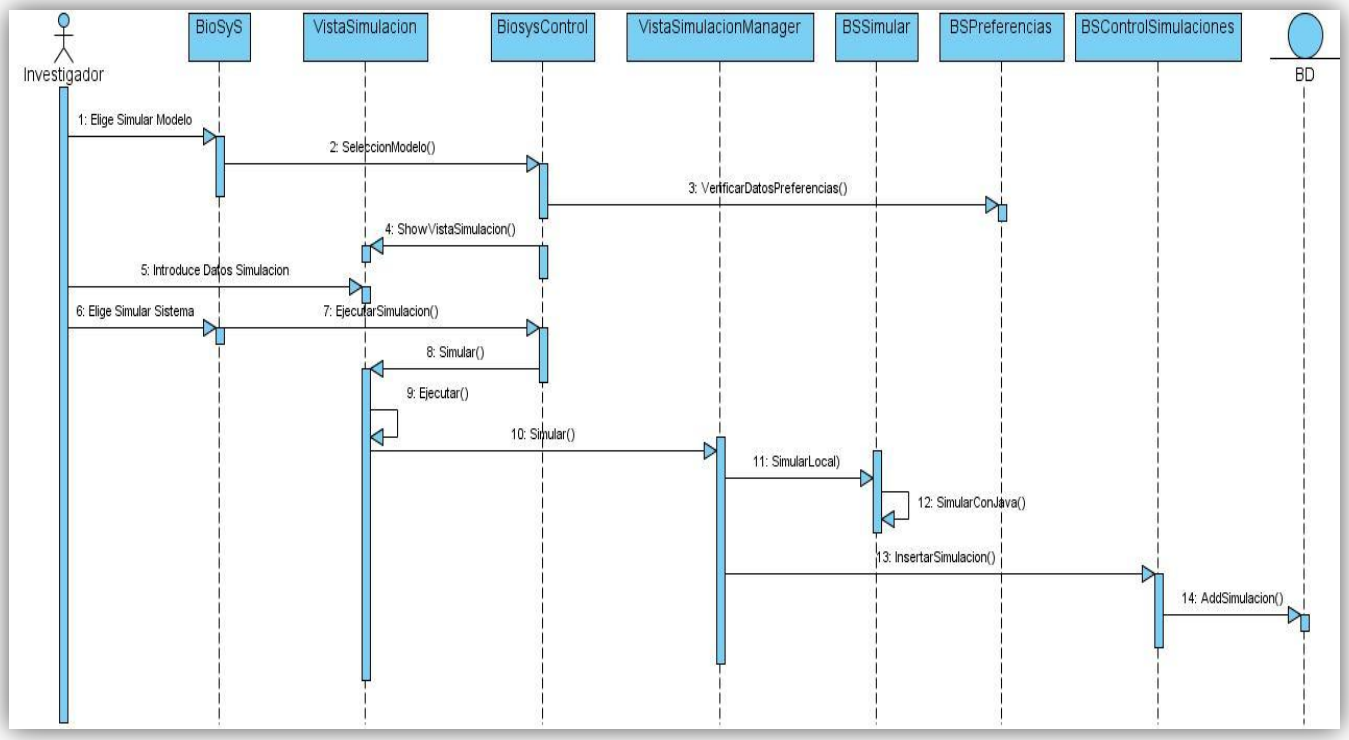


Figura 3.8 Diagrama de Secuencia Realizar Simulación utilizando la librería ODEtoJava.

DS Realizar Simulación utilizando el asistente matemático Octave:

El investigador elige la opción **Simular Modelo** en la interfaz **BioSyS**, seguidamente se genera un flujo de información a través del método **SelecciónModelo(),** que permite almacenar el modelo seleccionado en un objeto y se prosigue a verificar la preferencias para determinar qué tipo de simulación se realizará. Definido el tipo de simulación se muestra la vista **VistaSimulación**, encargada de recoger los datos, esto lo hace mediante el método **ShowVistaSimulación().** Una vez que el investigador obtiene la vista indicada introduce los datos de la simulación y elige la opción de simular el sistema, en este instante se invoca al método **EjecutarSimulación()** perteneciente a la interfaz principal, el mismo solicita a la clase control **BiosysControl**, esta clase mediante la función **Simular()** llama al método **Ejecutar()** de la interfaz de simulación que se esté mostrando, en este caso la **VistaSimulación,** donde este método invoca a la clase **VistaSimulacionManager** a que tome el control de la simulación, esta construye a partir de los datos de la interfaz todos los requerimientos de la simulación y facilita esta información a la clase **BSSimular**, ésta se apoya en la clase controladora de los eventos de Octave **BSControlOctave**. En esta instancia control se resuelve el Sistema de Ecuaciones Diferenciales bajo las condiciones iniciales, se obtiene la simulación y se devuelve esta a

la clase **VistaSimulacionManager**, la cual adiciona la simulación mediante la clase **BSControlSimulaciones** en la Base de Datos y usando el método **AddSimulación()**.

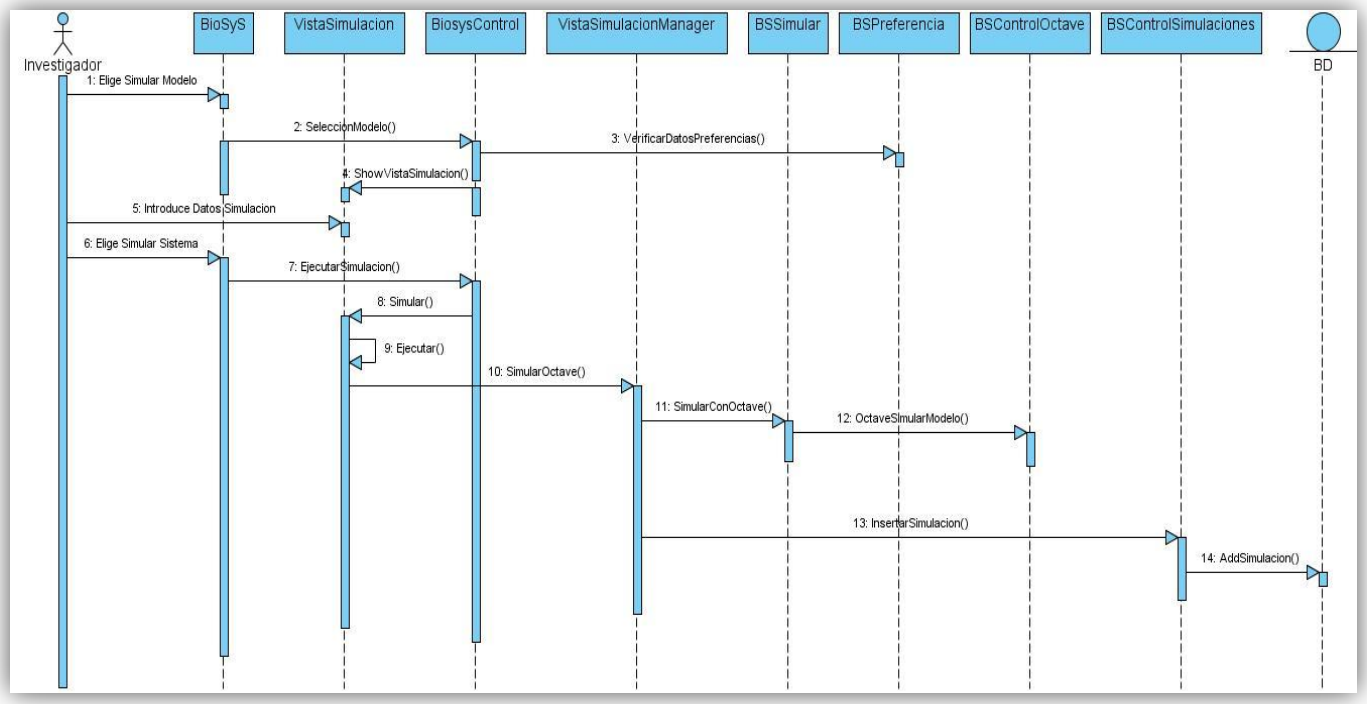


Figura 3.9 Diagrama de Secuencia Realizar Simulación utilizando el asistente matemático Octave

3.4 Modelo de Despliegue

El Modelo Físico/de Despliegue provee un modelo detallado de la forma en la que los componentes se desplegarán a lo largo de la infraestructura del sistema. Detalla las capacidades de red, las especificaciones del servidor, los requisitos de hardware y otra información relacionada al despliegue del sistema propuesto.

El modelo físico muestra dónde y cómo se desplegarán los componentes. Es un mapa específico de la instalación física del sistema. Un diagrama de despliegue ilustra el despliegue físico del sistema en un ambiente de producción (o prueba). Muestra dónde se ubicarán los componentes, en qué servidores, máquinas o hardware. Puede ilustrar vínculos de red, ancho de banda de LAN, etc. (45).

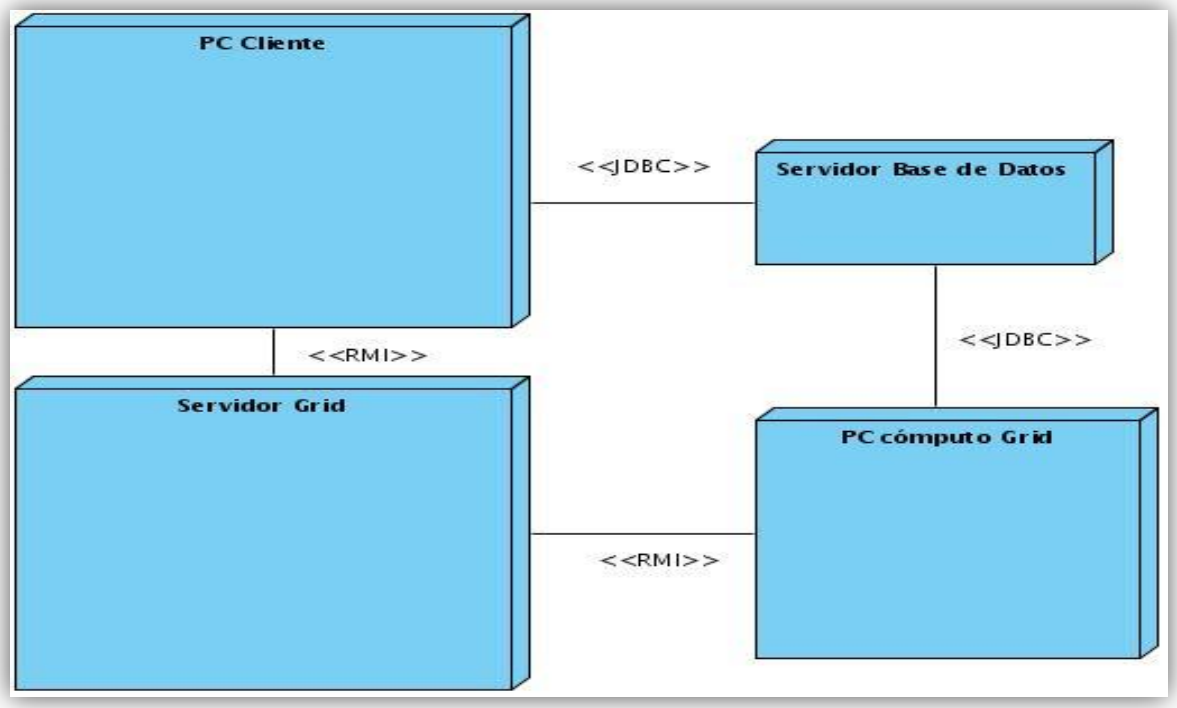


Figura 3.10 Modelo de Despliegue

3.5 Conclusiones del capítulo

Este capítulo dio la posibilidad de una mayor visión del funcionamiento interno del sistema, ya que se pudo conocer el diseño del software en general, a través de los diagramas de clases del diseño que se generaron. Esto permite que el flujo de implementación se realice de manera rápida y eficiente, logrando una solución técnica con calidad. Además se definió claramente la distribución física necesaria para un correcto funcionamiento y a la vez rendimiento de la aplicación.

Capítulo 4: Implementación

En este capítulo se hará una descripción de cómo los elementos del diseño se implementan en términos de componentes, se muestran los diagramas de componentes y de despliegue, así como del código fuente responsable de realizar las principales operaciones, dígase simulación con ODEtoJava y Octave.

4.1 Diagrama de Componentes

Un diagrama de componentes describe los elementos físicos de un sistema y sus relaciones, muestra las opciones de realización incluyendo código fuente, binario y ejecutable. Los componentes mostrados representan todos los tipos de elementos software que entran en la fabricación de aplicaciones informáticas. Pueden ser simples archivos, paquetes, bibliotecas cargadas dinámicamente, etc.

Los principales componentes que integran las funcionalidades añadidas al simulador BioSyS serán descritas a continuación, así como otros elementos usados y que son de gran importancia.

Vista: Reúne las clases visuales con las que el usuario interactuará con la aplicación. Estas son encargadas de obtener toda la información referente a configuraciones de preferencia del simulador, definición de herramienta matemática para simular, muestra de los modelos de la base de datos, obtención de los datos de la simulación, entre otras.

Control: Reúne a las clases encargadas de manejar los datos que serán usados para realizar simulaciones ya sea con ODEtoJava o con Octave. Es la encargada de gestionar los procesos que permiten levantar instancias de Octave, de suministrarle la información necesaria para la simulación, además es capaz de obtener los resultados de la simulación y almacenarlos en la Base de Datos.

Modelo: Se incluye en este la clase encargada de mantener los datos referentes a las preferencias del sistema y que puede ser usada en cualquier momento para obtener datos como Tipo de simulación, Método Numérico, Herramienta Matemática, Direcciones de aplicaciones externas, Direcciones de funciones a ejecutar, entre otras.

BSDAO: Capa de acceso a datos, representa el puente de comunicación entre el simulador y la Base de Datos donde ese encuentran almacenados los modelos y la información de sus respectivas simulaciones.

BD: Sistema de Base de Datos donde se encuentran almacenados los modelos matemáticos y el resto de la información referente a ellos.

BSSimulacion: Librería que contiene las funciones encargadas de realizar las simulaciones directamente con ODEtoJava.

ODEtoJava: Librería que contiene un conjunto de funciones capaces de resolver sistemas de ecuaciones diferenciales utilizando rutinas de solución, que se apoyan en el método Runge-Kutta.

Jopas: Librerías que contienen la lógica necesaria para servir de comunicación entre el lenguaje de programación Java y el Asistente Matemático Octave.

Octave: Representa al Asistente Matemático encargado de resolver los sistemas de ecuaciones diferenciales suministrados por el simulador.

FileOctaveFunction: Contiene la función Octave que utilizará el asistente.

jdsGUILibrary: Librería que permite la comunicación entre el Simulador y el 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 *Simulación*.

4.1.1 Diagrama de Componentes Simulación Local usando Octave u ODEtoJava

El componente **Vista** muestra los modelos de la Base de Datos y las interfaces necesarias para introducir los datos de la simulación, además de las posibles herramientas a usar. El tipo de herramienta y el método numérico mostrado quedan almacenados en el componente **Modelo**, del cual se nutre **Control** para determinar qué tipo de simulación realizar, si es **ODEtoJava** la comunicación se realiza a través de la librería **BSSimulación**, la cual a su vez usa las funcionalidades de **ODEtoJava** para resolver el sistema de ecuaciones diferenciales, las mismas son recibidas del componente **Control,** el cual las obtiene del modelo matemático seleccionado**.** Si la simulación debe realizarse usando el asistente matemático **Octave**, el componente **Control** es el encargado de escribir el sistema de ecuaciones diferenciales en un fichero de salida y de comunicarle al asistente **Octave** mediante las librerías **Jopas** lo que debe resolver. Finalmente los resultados son almacenados en la **BD** usando la capa de acceso a datos **BSDAO**.

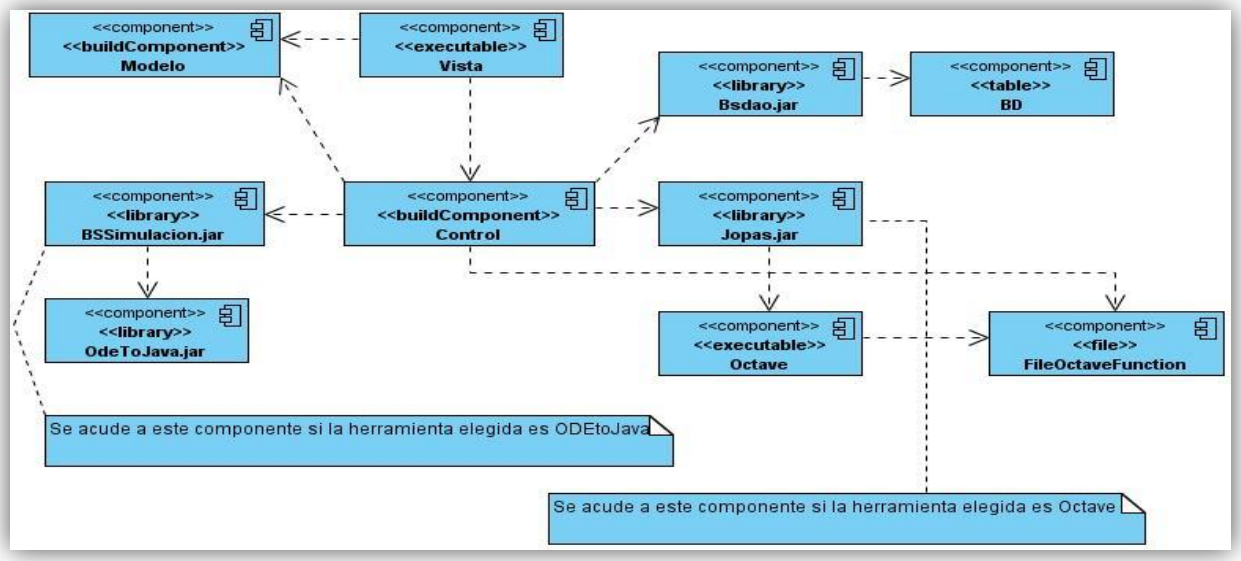


Figura 4.1 Diagrama de Componentes-Simulación Local.

4.1.2 Diagrama de Componentes Simulación Distribuida usando ODEtoJava

El componente **Vista** muestra los modelos de la Base de Datos y las interfaces necesarias para introducir los datos de la simulación, además de las posibles herramientas a usar. El tipo de herramienta y el método numérico mostrado quedan almacenados en el componente **Modelo**, del cual se nutre **Control** para determinar qué tipo de simulación realizar. Al determinar que esta será distribuida, **Control** se comunica con el **Subsistema T-arenal** mediante el componente **jdsGuiLibrary**, proveyéndole los datos de la simulación. A partir de este momento el **Subsistema T-arenal** es el encargado de generar las unidades de trabajo dejando esta responsabilidad al componente **BSDataManager**. Estas unidades serán enviadas a los equipos pertenecientes a la red distribuida, junto con los demás componentes del **Subsistema T-arenal**. Ya en el equipo de cómputo el componente **BSAlgorithm** es el responsable de realizar la simulación usando el componente **BSSimulación**. Al terminar la simulación, **BSAlgorithm** obtiene los resultados y los almacena en la **Base de Datos**.

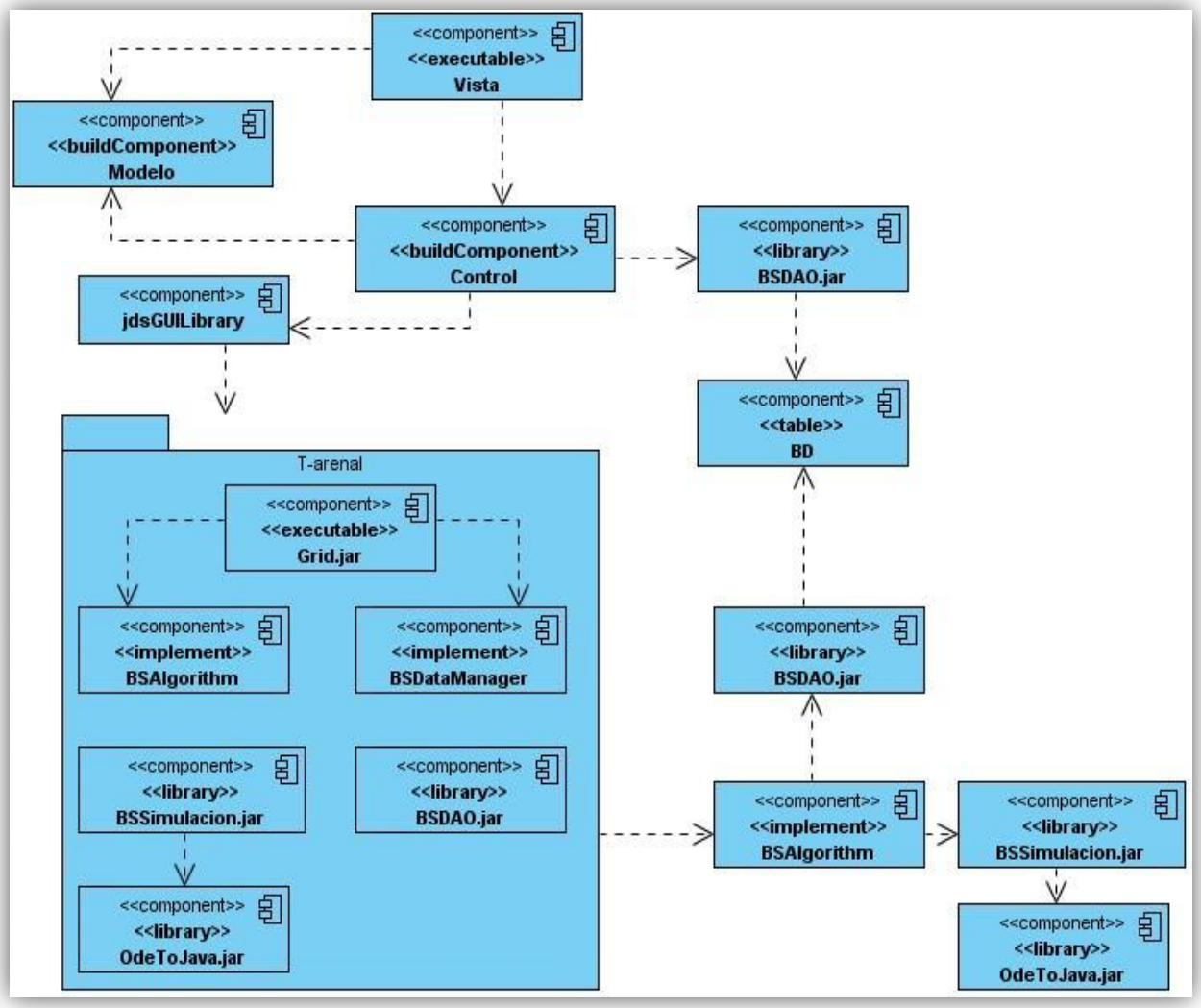


Figura 4.2 Diagrama de Componentes-Simulación Distribuida usando OdeToJava.

4.1.3 Diagrama de Componentes Simulación Distribuida usando Octave

A pesar de que la simulación distribuida usando **Octave** como herramienta de cálculo hasta el momento en que se escriben estas líneas no es posible prácticamente, debido a que la capa de acceso a datos **BSDAO** y la **Base de Datos (BD)** en sí misma no poseen un tipo de modelo matemático compatible con **Octave**, sí es posible en teoría. El siguiente diagrama de componentes define la lógica de este proceso.

En este caso la simulación distribuida se realizaría de la misma manera que al simular usando **ODEtoJava**, con la diferencia de que el componente **BSAlgorithm** en el equipo de cómputo crearía una instancia de **BSControlOctave**, el cual usando **Jopas** crearía el puente de comunicación con el

ejecutable de **Octave**. A su vez **BSAlgorithm** escribiría en un fichero el problema a resolver, dicho problema sería resuelto por **Octave** y la solución se almacena posteriormente en la Base de Datos.

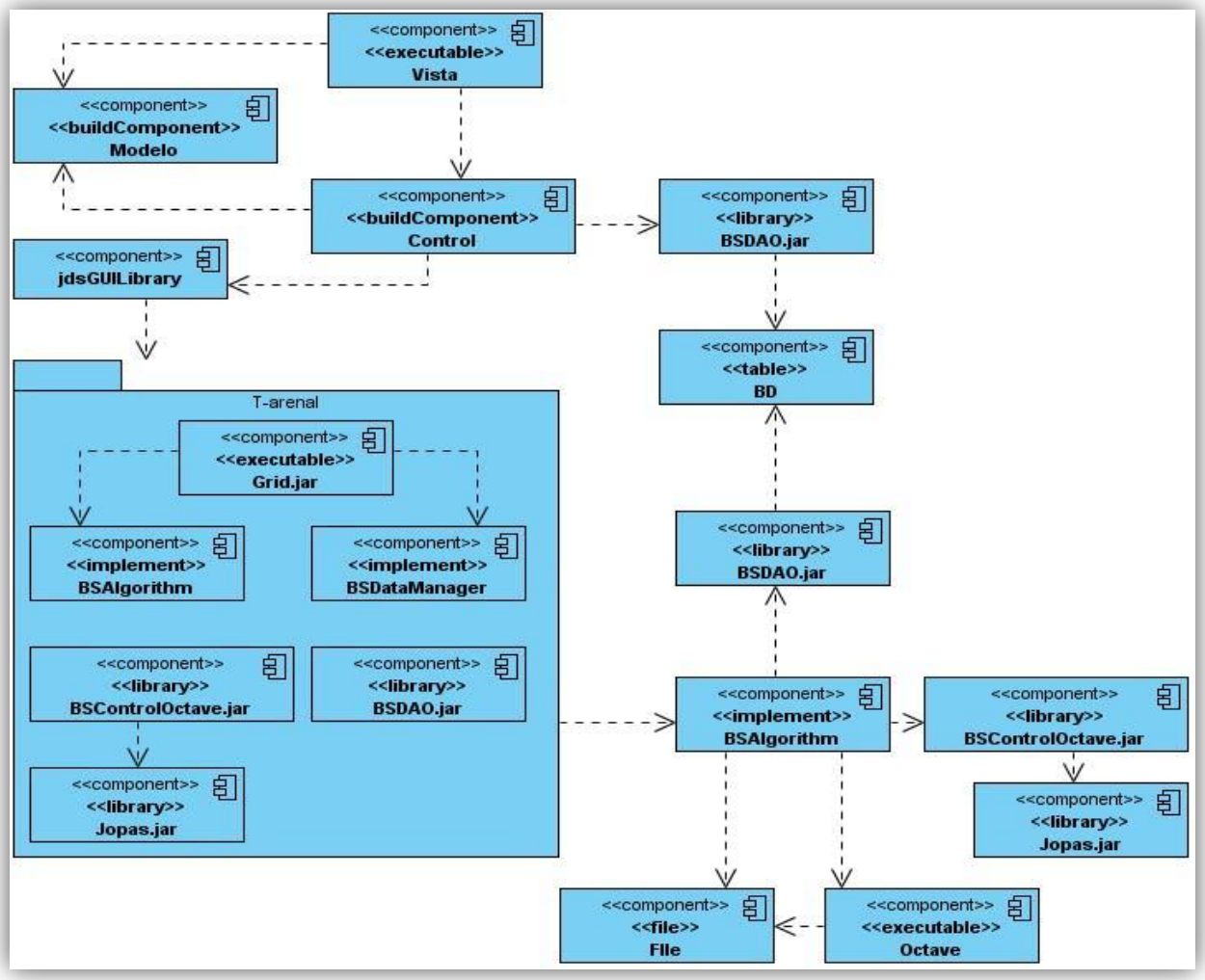


Figura 4.3 Diagrama de Componentes Simulación Distribuida usando Octave.

4.2 Diagrama de Despliegue Detallado

PC Cliente: Representa el equipo donde el investigador tiene instalado el cliente del simulador, en este equipo está toda la lógica del sistema, el cual puede realizar tanto simulaciones locales como distribuidas.

Desde este nodo comienza el flujo de información que permite realizar la simulaciones, posee la funcionalidad de lograr la comunicación cliente servidor a través de la vía RMI y de consultar y almacenar datos en la Base de Datos mediante el protocolo JDBC.

Servidor Grid: Representa al servidor que posee la capacidad de crear procesamiento distribuido, recibe un paquete que contiene un problema complejo, lo divide en paquetes más que pequeños y los distribuye en un rango de equipos que tiene a su disposición y que serán encargados de resolver porciones del problema original. El servidor además envía los componentes *BSAlgorithm*, BS*Simulacion* y *BSDAO,* estos componentes proveen a la PC de Cómputo de herramientas para llevar a cabo las operaciones. El flujo de información se realiza a través de sockets, puertas de entrada y salida de datos entre el cliente y el servidor Grid.

PC Computo Grid: Equipo a disposición del servidor Grid, en el cual ubican los componentes capaces de realizar la solución de un problema, estos equipos son los encargados de llevar a cabo el trabajo enviado por el servidor de objetos. En caso de que las simulaciones utilicen Octave como herramienta de cálculo de Sistemas de Ecuaciones Diferenciales, el nodo PC Cómputo debe tener instalado Octave y poseer las librerías de resolución de ODE que vienen con la versión 2.1.50.

Servidor de Base de Datos: Representa al Servidor que contiene toda la información relacionada con los Modelos Matemáticos, así como las simulaciones y otros datos asociados a cada uno de estos. Para obtener una comunicación segura, confiable y sin errores con este Sistema de Datos, se usa 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.

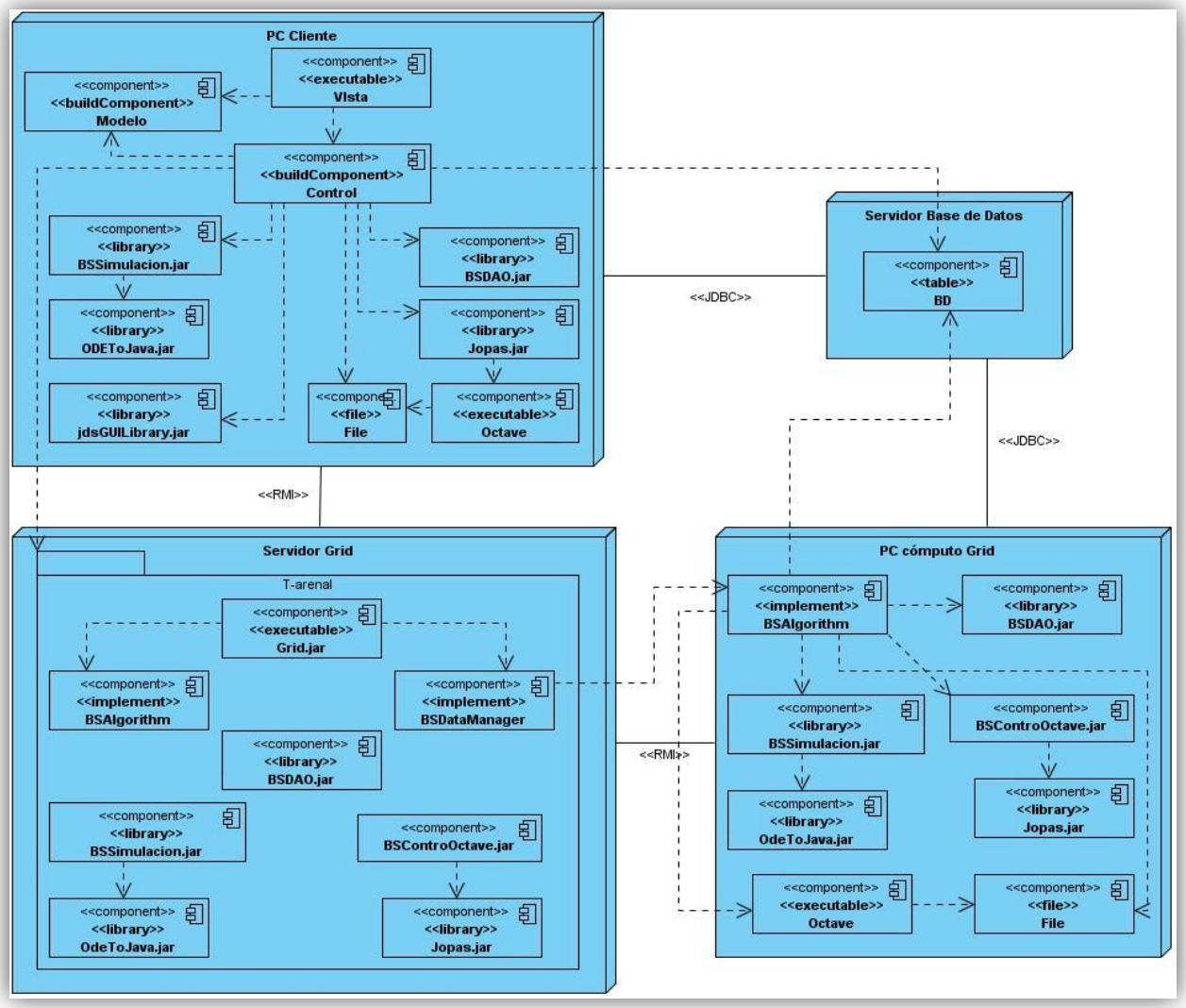


Figura 4.4 Diagrama de Despliegue Detallado.

4.3 Código Fuente de las Principales Clases y Funcionalidades

Se realizará un análisis de cada una de las clases y funcionalidades que básicamente permiten la simulación o resolución de Sistemas de Ecuaciones Diferenciales con ODEtoJava y con Octave, y que han sido el objetivo principal de este trabajo de diploma.

4.3.1 Función Simular de la clase BSSimular librería BSSimular.jar

Esta función representa básicamente gran parte del proceso de simular con ODEtoJava, la misma

recibe la función ODE que contiene el sistema de ecuaciones diferenciales, recibe además una colección de condiciones iniciales, un String que identifica la simulación, los valores de tiempo inicial y final, los valores de la tolerancia absoluta y relativa, el paso de integración y por último, pero no menos importante un valor entero que determinará el método de simulación que se usará. Este valor podrá alcanzar 4 valores, si corresponde con 1 el método a usar será el ErkTriple de la librería ODEtoJava, 2 DormandPrince, 3 Erk y 4 o cualquier otro número ErkSD.

```
public Simulacion Simular(ODE iface, ArrayList<Double> initialCond, ArrayList<Double> par, String idsim, double ti, double tf, double eam, double erm, double step, int Metodo) throws ClassNotFoundException, InstantiationException, IllegalAccessException, IOException {
    //<editor-fold defaultstate="collapsed" desc="Simular con Java">
    //*********************************************************************************************
    //Objeto que contiene el rango de tiempo en el que se va a simular
    Span span;
    if (step == 0)
    {
        //Si el paso es 0 se crea el objeto con el rango tiempo inicial y final el paso será autoajustable
        span = new Span(ti, tf);
    } else
    {
        //Si el paso es un valor se crea el objeto con el rango tiempo inicial y final y el paso
        span = new Span(ti, tf, step);
    }
    //Arreglo con el error absoluto máximo para cada una de las ecuaciones del sistema
    double[] atol1 = new double[initialCond.size()];

    //Arreglo con el error relativo máximo para cada una de las ecuaciones del sistema
    double[] rtol1 = new double[initialCond.size()];

    for (int i = 0; i < initialCond.size(); i++)
    {
        atol1[i] = eam;
        rtol1[i] = erm;
    }
    //Se crea el arreglo de condiciones iniciales
    double[] c = new double[initialCond.size()];

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

    //Se crea el arreglo de parámetros
    double[] m = new double[par.size()];
    for (int j = 0; j < m.length; j++)
```

```
        {
            m[j] = par.get(j);
        }
        //Se resuelve el modelo matemático como los parámetros y condiciones
        if(Metodo == 1)
          return ErkTriple.erk_triple(iface, span, c, m, idsim, -1.0, new Btableau("dopr54"), 5.0, atol1, rtol1,
"StiffDetect_Off", "EventLoc_Off", "Stats_Off");

          else if(Metodo == 2)
            return DormandPrince.dormand_prince(iface, span, c, m, -1.0, atol1, rtol1,
idsim,"StiffDetect_Off", "EventLoc_Off", "Stats_Off");

            else if(Metodo == 3)
              return Erk.erk(iface, span, c, m, -1.0, new Btableau("dopr54"), idsim,"Stats_Off");
                else   return ErkSD.erk_sd(iface, span, c, -1.0,  new Btableau("dopr54"), atol1, rtol1, idsim,
"Stats_Off");
//**********************************************************************************************
//</editor-fold>
}
```

Los métodos de ErkTriple, DormandPrince, Erk y ErkSD fueron modificados de manera que pudieran devolver objetos de tipo Simulación, originalmente estas funciones fueron creadas para devolver un vector de valores double, estos representan las soluciones de los Sistemas de Ecuaciones Diferenciales. Sin embargo el código intrínseco de las funcionalidades de ODEtoJava no fue modificado, ya que estos métodos han sido probados con anterioridad y su correcto funcionamiento es innegable.

El código de dichos métodos no será mostrado en este documento debido a su extensión, alto contenido matemático y además porque no constituyen objetivo de este trabajo de diploma. De cualquier manera su código puede ser descargado directamente desde Internet o se puede tener acceso a él a través de la aplicación en cuestión.

4.3.2 Clase BSControlOctave paquete Control

En esta clase se encuentra toda la lógica que se necesita para realizar simulaciones usando el asistente matemático Octave, se muestra a continuación una descripción detallada de los atributos y funciones que la componen, incluyendo qué tipo de funcionalidad desempeñan y el proceso que se lleva a cabo para lograr la funcionalidad.

Atributos de la Clase

El atributo **jopas** es el objeto encargado de brindar las funcionalidades de la librería Jopas, este objeto pertenece a la clase **BSControlJopas**, que hereda todas la funcionalidades de **Jopas**. El atributo **octavePathFunctions** se refiere y contendrá la dirección del directorio en el cual el simulador almacena las funciones **Octave**, que contienen los sistemas de ecuaciones diferenciales, y que serán resueltos por la aplicación **Octave**.

Los atributos **octavePathlinux** y **octavePathWindows** contienen respectivamente la dirección del asistente **Octave**, los mismos permitirán verificar que no existen errores a la hora de utilizar el asistente, además posibilitan modificar el fichero **JopasProperties**, asignando un path de direcciones correcto al simulador en dependencia del tipo de sistema operativo usado por el usuario. El atributo **octaveLsDir** se refiere a la dirección de un fichero que en los sistemas operativos Linux, le permite al **Octave** determinar la dirección de las funciones que debe ejecutar, este atributo posibilitará modificar dicho fichero al añadirle el atributo **octavePathFunctions** como dirección de ejecución.

El atributo **vista** representa una instancia de la clase visual en la que se manejan los datos de la simulación realizada con **Octave**, permitirá a esta clase control realizar algunas operaciones sobre la clase visual. Por último las variables de simulación que son capaces de contener los datos relacionados con la simulación: **resultados** contiene una colección de **Resultado**, los mismos se obtienen a partir de una simulación. El atributo **correcta** determina si una simulación ha sido correcta o no, **idsim** representa el id con el cual se identifica la simulación y **pathActual** contiene la información del fichero con el sistema de ecuaciones diferenciales a resolver.

```
private BSControlJopas jopas;
 private String octavePathFunctions;
 private String octavePathlinux;
 private String octavePathWindows;
 private String octaveLsDir;
 BSVistaOctave vista;
 //--variables de simulación--------
 private Set<Resultado> resultados = new HashSet<Resultado>();
 private boolean correcta = true;
 private String idsim;
 File pathActual;
```

4.3.3 Constructor de la clase BSControlOctave paquete Control

Constructor encargado de inicializar los atributos de la clase.

```
public BSControlOctave(String pOctavePathLinux, String pOctavePathWindows, String pOctaveLsDir,
String pOctavePathFunction, BSVistaOctave pVistaOctave)
 {
  octavePathlinux = pOctavePathLinux;
  octavePathWindows = pOctavePathWindows;
  octaveLsDir = pOctaveLsDir;
  octavePathFunctions = pOctavePathFunction;
  jopas = new BSControlJopas();
  vista = pVistaOctave;
  pathActual = new File(this.GetOctavePathFunction());
    if(!pathActual.exists())
        pathActual.mkdir();
 }
```

4.3.4 Función getResultados de la clase BSControlOctave paquete Control

Este método sencillamente crea un nuevo objeto de tipo Simulación, inicializa los atributos de la simulación y usa la colección de **Resultados** declarada como atributo de la clase, para modificar los resultados de la simulación creada, por último retorna la Simulación.

```
public Simulacion getResultados()  // Método que devuelve los resultados de una simulación
    {
        Simulacion simulacion = new Simulacion();
        simulacion.setCorrecta(correcta);
        simulacion.setIdsim(idsim);
        simulacion.setResultados(resultados);
        return simulacion;   // return profile array
    }
```

4.3.5 Función octaveLsodeSolveModel de la clase BSControlOctave paquete Control

Esta función es la encargada de realizar la simulación usando para ello la función lsode, algoritmo capaz de resolver sistemas de ecuaciones diferenciales y que se provee en el paquete ODE del sistema de librerías y funciones que usa Octave como asistente matemático. A pesar de que la función contiene una gran cantidad de comentarios, se muestra a continuación una descripción detallada de sus funcionalidades.

Los parámetros del método son los siguientes: **BSVistaOctave octavev**, se refiere a la vista o interface donde se obtienen los datos de la simulación y en la cual se mostrarán los resultados de la simulación. **String nameFunction** representa el nombre del fichero en el cual está almacenado el sistema de ecuaciones diferenciales, este nombre le permitirá reconocer a **Octave** qué función debe resolver, **int cantEcua** este parámetro tiene una gran importancia a la hora de realizar la simulación, la misma permite que los resultados se obtengan de manera correcta, **String idsim** define el id de la simulación, **double[] x, double[]** y representan los parámetros y condiciones iniciales que se reciben del sistema para realizar la simulación, **timeaLinspace**, **double timebLinspace**, **double tpointLinspace** especifican los datos necesarios para el papel que juega la variable tiempo a la hora de resolver el sistema de ecuaciones diferenciales, respectivamente representan el tiempo inicial, tiempo final y la distribución de paso entre las iteraciones del tiempo. El modo de cargarle datos a las variables de entorno de Octave, se realiza a través del método **Load()**, el cual viene implementado en la librería Jopas, a partir de este instante todo proceso de interacción entre la aplicación y Octave tendrá lugar mediante el objeto jopas. Esta librería brinda tres funcionalidades básicas: **Load()** del cual hablamos ya, **Save()** que permite tomar una variable del entorno de Octave y salvarla en una variable local y por último **Execute()**, esta función permite hacer llamadas a funciones del entorno de Octave u otras funciones previamente programadas.

La línea básica de la función se desarrolla de la manera siguiente: se cargan los valores del tiempo en tres variables **ta**, **tb**, y **tp**, siempre validando condiciones lógicas. Seguidamente se cargan los parámetros y condiciones iniciales, posteriormente se crea un vector llamado **truevalor** que contendrá cada uno de las condiciones iniciales y parámetros en ese orden, luego se carga ese vector en una variable **z,** se calcula el valor del tiempo mediante la función **linspace** y se almacena ese resultado en la variable **t**. Con estos valores se procede a realizar el cálculo del sistema de ecuaciones diferenciales mediante la función **lsode**, ésta recibe como parámetros el nombre de la función que contiene el sistema (**String nameFunction**), el vector de condiciones iniciales, parámetros (**z**) y el valor del tiempo (**t**). El resultado de este cálculo se almacena en una variable llamada **valor,** después de este proceso se comienza a realizar una selección de los resultados para obtener una matriz de resultados de tantas columnas como ecuaciones tenga el sistema de ecuaciones diferenciales, por esa razón se crea una variable llamada **valorSel**, en la cual se almacenará esta reducción de resultados.

Finalmente se carga en una matriz local **T** el resultado del tiempo y en una **M** el conjunto de resultados almacenados en **valorSel**, se procede a recorrer ambas matrices y cada resultado obtenido se adiciona a la colección de resultados (**resultado**), que formará parte de la **Simulación** devuelta.

Finalmente se llama al método **getResultados(),** el cual se encarga de crear una nueva simulación, se le insertan los resultados obtenidos y devuelve dicha simulación, terminando de esta manera la ejecución de la función.

```
public Simulacion octaveLsodeSolveModel(BSVistaOctave octavev,String nameFunction,int cantEcua,
String idsim, double[] x, double[] y, double timeaLinspace, double timebLinspace, double
tpointLinspace)
 {
   octavev.ClearOutputConsole();

  if (timeaLinspace< timebLinspace)          // comprobación de errores al introducir el tiempo
     {
        jopas.Load(timeaLinspace, "ta");
        jopas.Load(timebLinspace, "tb");
        jopas.Load(tpointLinspace,"tp");
     }
  else
  {
      octavev.GetOutPutConsole().append("timea0Linspace must be lower than timefLinspace\n");
      System.out.println("timea0Linspace must be lower than timefLinspace");
  }
  for (int i = 0; i < y.length; i++)
     {
        String varAsby = "z"+(i+1);
        jopas.Load(y[i],varAsby);
        octavev.GetOutPutConsole().append("Octave is Loading: " + varAsby+ ": " + y[i] + "\n");
        System.out.println("Octave is Loading: " + varAsby+ ": " + y[i]);
     }
   for (int j = 0; j <x.length; j++)
    {
        String varAsbx = "z"+(j+y.length+1);
        jopas.Load(x[j],varAsbx);
        octavev.GetOutPutConsole().append("Octave is Loading: " +varAsbx+": " + x[j] + "\n");
        System.out.println("Octave is Loading: " +varAsbx+": " + x[j]);
    }

   int totalParm = y.length + x.length;
   String truevalor = new String();

   for(int i = 1; i<= totalParm; i++)      //creación del vector de parámetros y condiciones iniciales
    {
     if(i == 1)
     {
      truevalor = "[z" + i;
     }
     else if(i == totalParm)
     {
      truevalor = truevalor +  "; z" + i + "];";
```

```
      }
      else
      {
       truevalor = truevalor +  "; z" + i;
      }
     }
   octavev.GetOutPutConsole().append("z = " + truevalor +"\n\n");
   System.out.println("z = " + truevalor);
   jopas.Execute("z = " + truevalor);                    //se carga el vector creado
   jopas.Execute("t=linspace(ta,tb,tp)';");              //se inicializa la variable tiempo
   String nameFunctionConv = "\""+ nameFunction +"\"";  //se crea un string con el nombre del
fichero donde está almacenado el sistema de ecuaciones diferenciales

   octavev.GetOutPutConsole().append("Executing Rutina OctaveLSode....\n");
   System.out.println("Executing Rutina OctaveLSode....");

   jopas.Execute("valor = lsode("+ nameFunctionConv +",z,t)");       //se resuelve el sistema de
ecuaciones diferenciales usando el método lsode y se almacena en una variable

octavev.GetOutPutConsole().append("OCTAVE Function Executing: valor = lsode("+
nameFunctionConv  +",z,t)\n\n");   //esto es una verificación del string que se carga en la
instancia de Octave

   System.out.println("OCTAVE Function Executing: valor = lsode("+ nameFunctionConv  +",z,t);");

   String valorSel = "";                  //aquí se hace una selección de los resultados, adecuándolos a
la cantidad de ecuaciones a resolver.

   for(int i= 1; i<=cantEcua; i++)
   {
     if(i==1)
        valorSel = "valorSel=[valor(:,"+i+") ";
     else if(i == cantEcua)
        valorSel = valorSel + "valor(:,"+i+")];";
     else
        valorSel = valorSel + "valor(:,"+i+") ";
   }
   jopas.Execute(valorSel);                                  //se cargan los resultados seleccionados

   octavev.GetOutPutConsole().append("Reduccion de los resultados: "+valorSel + "\n\n");
//esto es una verificación del string que se carga en la instancia de Octave

   System.out.println("Reducción de los resultados: "+valorSel);

  Matrix T = jopas.Save("t");                                 //se almacena en una matriz el valor de la
variable tiempo, obteniendo los mismos de la instancia de Octave

  Matrix M = jopas.Save("valorSel");                  //se almacena en una matriz el valor de la variable
resultado, obteniendo los mismos de la instancia de Octave
```

```
for(int i=0;i<M.getRows();i++)                    //se recorre la matriz y se almacenan los elementos internos en un resultado de simulación listos para ser devueltos

      for(int j=0;j<M.getColumns();j++)
      {
         double valor = M.getRealAt(i, j);
         double tiempo= T.getRealAt(i, 0);
         resultados.add(new Resultado((j+1), tiempo, valor, idsim));
         octavev.GetOutPutConsole().append("Equation: " + (j+1) +" Iteration: " + (i+1) + " Result: " + valor + " Time: " + tiempo + "\n");
         System.out.println("Equation: " + (j+1) +" Iteration: " + (i+1) + " Result: " + valor + " Time: " + tiempo);
       }

   jopas.Execute("plot(t,valorSel)");
   return this.getResultados();      //se devuelven los resultados de la simulación
 }
```

4.3.6 Función octaveOde23SolveModel de la clase BSControlOctave paquete Control

Este método tiene la particularidad de ser casi idéntico a las funciones **octaveOde45SolveModel** y **octaveOde78SolveModel**, solo se diferencian en la función en la que se apoyan para resolver el sistema de ecuaciones diferenciales, por tanto se hará el análisis en este documento de solo uno de ellos, en este caso **octaveOde23SolveModel.** Los parámetros que recibe esta función son en la misma medida que en **octaveLsodeSolveModel** consecuentes con las variables que necesita la función Octave para calcular, sin embargo ambas necesitan un modelo matemático, que aunque puede contener el mismo sistema de ecuaciones diferenciales tienden a ser diferentes el uno del otro debido al modo en que se resuelven. Básicamente los nuevos parámetros que se reciben en esta función son: **double ode_fcn_format,** el cual define la forma en que está definido el sistema de ecuaciones diferenciales, que puede ser de la forma xprime = fun(t,x) cuando (ode_fcn_format=0, Valor por defecto) o: xprime = fun(x,t) (ode_fcn_format=1). El parámetro **double tol,** define la precisión deseada para el cálculo, es decir, la tolerancia a errores, el parámetro **double trace** define si se realiza una traza en cada paso de integración y por último **double count** identifica a un contador para conocer la cantidad de evaluaciones del sistema.

El proceso de resolución del sistema continúa casi del mismo modo que en el método **octaveLsodeSolveModel**, se cargan las condiciones iniciales y parámetros, se llama a la función

ode23, se hace una selección de resultados, se almacenan en matrices y por último se crea una simulación con los resultados obtenidos y se devuelve dicha simulación.

```
Public Simulacion octaveOde23SolveModel(BSVistaOctave octavev,String nameFunction,int
cantEcua, String idsim, double[] x, double[] y, double timeT0, double timeTF, double ode_fcn_format,
double tol, double trace, double count)
{
   String ListaRecursos = "";
   String nameFunctionConv = "\""+ nameFunction +"\"";

   for (int i = 0; i < y.length; i++)
    {
        String varAsby = "z"+(i+1);
        jopas.Load(y[i],varAsby);
        System.out.println(varAsby+ " " + y[i]);
        octavev.GetOutPutConsole().append("Octave is Loading: " + varAsby+ ": " + y[i] + "\n");
    }

    for (int j = 0; j <x.length; j++)
    {
        String varAsbx = "z"+(j+y.length+1);
        jopas.Load(x[j],varAsbx);
        System.out.println(varAsbx+ " " + x[j]);
        octavev.GetOutPutConsole().append("Octave is Loading: " + varAsbx+ ": " +x[j] + "\n");
    }

    int totalParm = y.length + x.length;
    String truevalor = new String();

    for(int i = 1; i<= totalParm; i++)        //creación del vector de parámetros y condiciones iniciales
    {
     if(i == 1)
     {
      truevalor = "[z" + i;
     }
     else if(i == totalParm)
     {
      truevalor = truevalor +  "; z" + i + "]);";
     }
     else
     {
      truevalor = truevalor +  "; z" + i;
     }
    }

   ListaRecursos = "[tout, xout]=ode23("+ nameFunctionConv + ",["+ timeT0 + "," + timeTF+"],"+
truevalor;
       octavev.GetOutPutConsole().append(ListaRecursos+"\n\n");
```

```
        System.out.println(ListaRecursos);
        octavev.GetOutPutConsole().append("Ejecutando Rutina OctaveOde23....");
        System.out.println("Ejecutando Rutina OctaveOde23....");

        jopas.Execute(ListaRecursos);
        System.out.println("OCTAVE Function Executing: " + ListaRecursos); //esto es una verificación del string que se carga en la instancia de Octave

        octavev.GetOutPutConsole().append("OCTAVE Function Executing: " + ListaRecursos +"\n\n");
   String valorSel="";            //selección de los resultados, adecuándolos a la cantidad de ecuaciones a resolver

   for(int i= 1; i<=cantEcua; i++)
   {
     if(i==1)
        valorSel = "valorSel=[xout(:,"+i+") ";
     else if(i == cantEcua)
        valorSel = valorSel + "xout(:,"+i+")];";
     else
        valorSel = valorSel + "xout(:,"+i+") ";
   }
   jopas.Execute(valorSel);

   Matrix T = jopas.Save("tout");
   Matrix M = jopas.Save("valorSel");

    for(int i=0;i<M.getRows();i++)       //se recorre la matriz y se almacenan los elementos internos en un resultado de simulación listos para ser devueltos
       for(int j=0;j<M.getColumns();j++)
       {
          double valor = M.getRealAt(i, j);
          double tiempo= T.getRealAt(i, 0);
          resultados.add(new Resultado((j+1), tiempo, valor, idsim));
          System.out.println("Equation: " + (j+1) +" Iteration: " + (i+1) + " Result: " + valor + " Time: " + tiempo);
          octavev.GetOutPutConsole().append("Equation: " + (j+1) +" Iteration: " + (i+1) + " Result: " + valor + " Time: " + tiempo +"\n");
       }

  jopas.Execute("plot(tout,valorSel)");
  return this.getResultados(); //se devuelve la simulación
}
```

Todas estas variantes de solución son rutinas que se basan en el método de Runge-Kutta, por tanto son métodos de paso simple, que sólo requieren de los resultados del paso anterior. Esto les brinda la posibilidad de autoiniciarse, 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 originen cambios acotados en la solución.

4.4 Pruebas de Fiabilidad

El correcto funcionamiento de las nuevas formas de simulación que han sido integradas al simulador BioSyS es esencial, una mala simulación puede traer consigo resultados inesperados y defectuosos, tratar de minimizar estos riesgos debe formar parte de la política de cualquier proyecto que desarrolle software. A pesar de que todos los métodos capaces de resolver sistemas de ecuaciones diferenciales que se han usado ya han sido probados a otros niveles y su funcionalidad se garantiza por las organizaciones, empresas o personas que los brindan, un mal uso de ellos puede acarrear soluciones erróneas y es por lo cual se han planificado un conjunto de pruebas que nos permitirán validar su funcionalidad.

4.4.1 Prueba de resultados para los nuevos métodos de simulación implementados

Modelo a Usar:

Presa-Depredador

Descripción del modelo:

Cantidad de variables: 2

Cantidad de parámetros: 4

Variables: x,y

Parámetros: a,b,c,d

```
function [r_r]=Presa_Depredador(varargin)
r_r = zeros(2,1);
y_y = varargin{2};
p_p = varargin{3};
r_r(1)=p_p(1)*y_y(1)-(p_p(2)*y_y(1)*y_y(2));
r_r(2)=-(p_p(3)*y_y(2))+p_p(4)*y_y(1)*y_y(2);
if isequal(isfinite(r_r),true(size(r_r))) == false
error(lastwarn);
end
```

Figura 4.5 Descripción matemática del modelo:(Modelo MatLab).

```
function r_r=Presa_DepredadorLSODE(z,t)
y_y(1)=z(5);
y_y(2)=z(6);
p_p(1)=z(1);
p_p(2)=z(2);
p_p(3)=z(3);
p_p(4)=z(4);
r_r = zeros(2,1);
r_r(1)=p_p(1)*y_y(1)-(p_p(2)*y_y(1)*y_y(2));
r_r(2)=-(p_p(3)*y_y(2))+p_p(4)*y_y(1)*y_y(2);
%completamiento vectorial
r_r(3)=0;
r_r(4)=0;
r_r(5)=0;
r_r(6)=0;
end
```

Figura 4.6 Descripción matemática del modelo:(Modelo Octave).

Variable	Valor
X	1.003
Y	1.254

Tabla 4.1 Valores de simulación para las variables.

Parámetro	Valor
A	0.238
B	1.73
C	0.147
D	1.619

Tabla 4.2 Valores de simulación para los parámetros.

Tiempo inicial	Tiempo final	Valor de integración
1,0	10,0	33

Tabla 4.3 Valores del tiempo.

4.4.2Simulación con Octave Método Lsode

Equation: 1 Iteration: 1 Result: 0.238 Time: 1.0
Equation: 2 Iteration: 1 Result: 1.73 Time: 1.0 Equation: 1 Iteration: 33 Result: -35329.07092404258 Time: 10.0
Equation: 2 Iteration: 33 Result: 18.397808102 Time: 10.0

Tabla 4.4 Resultados de la simulación para cada ecuación.

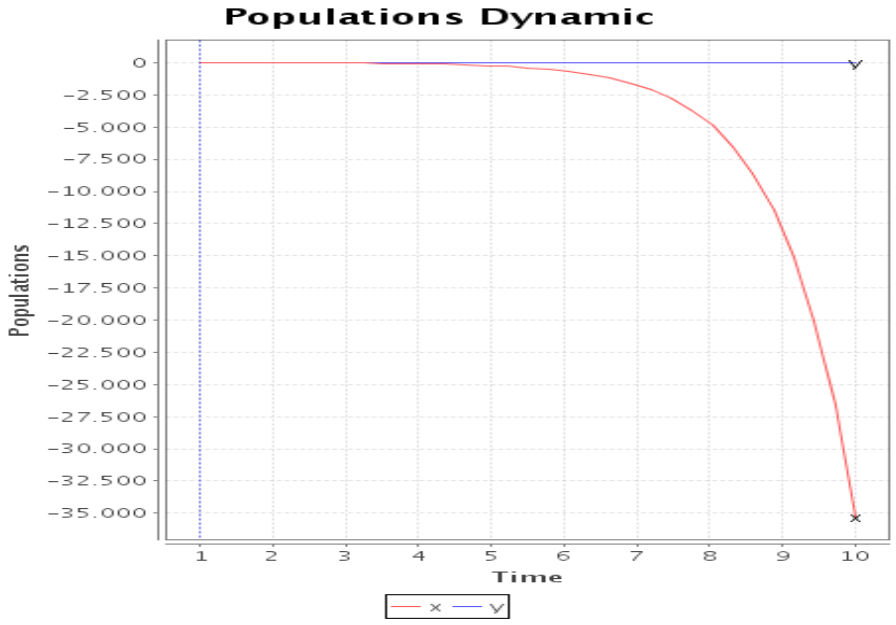


Figura 4.7 Gráficas de resultado para el método Lsode.

4.4.3 Simulación con Octave Método Ode23

Equation: 1 Iteration: 1 Result: 0.238 Time: 1.0 Equation: 2 Iteration: 1 Result: 1.73 Time: 1.0 Equation: 1 Iteration: 75 Result: -35304.90680375171 Time: 10.0
Equation: 2 Iteration: 75 Result: 18.397808102 Time: 10.0

Tabla 4.5 Resultados de la simulación para cada ecuación.

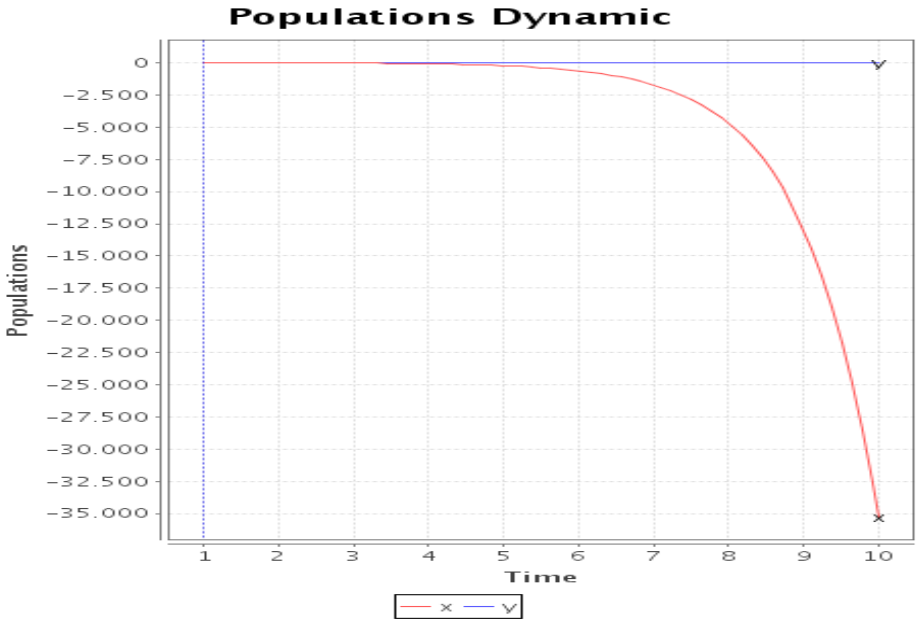


Figura 4.8 Gráficas de resultado para el método Ode23.

4.4.4 Simulación con Octave Método Ode45

.

Equation: 1 Iteration: 1 Result: 0.238 Time: 1.0 Equation: 2 Iteration: 1 Result: 1.73 Time: 1.0
Equation: 1 Iteration: 49 Result: -35329.0514251032 Time: 10.0 Equation: 2 Iteration: 49 Result: 18.397808102 Time: 10.0

Tabla 4.6 Resultados de la simulación para cada ecuación

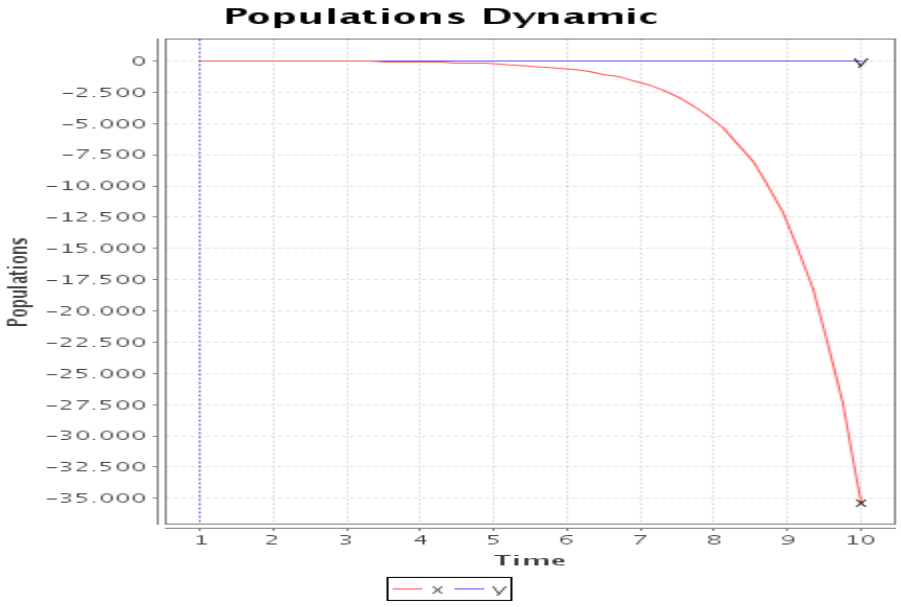


Figura 4.9 Gráficas de resultado para el método Ode45.

4.4.5 Simulación con Octave Método Ode78

Equation: 1 Iteration: 1 Result: 0.238 Time: 1.0
Equation: 2 Iteration: 1 Result: 1.73 Time: 1.0 Equation: 1 Iteration: 15 Result: -35329.02849884686 Time: 10.0 Equation: 2 Iteration: 15 Result: 18.397808102 Time: 10.0

Tabla 4.7 Resultados de la simulación para cada ecuación.

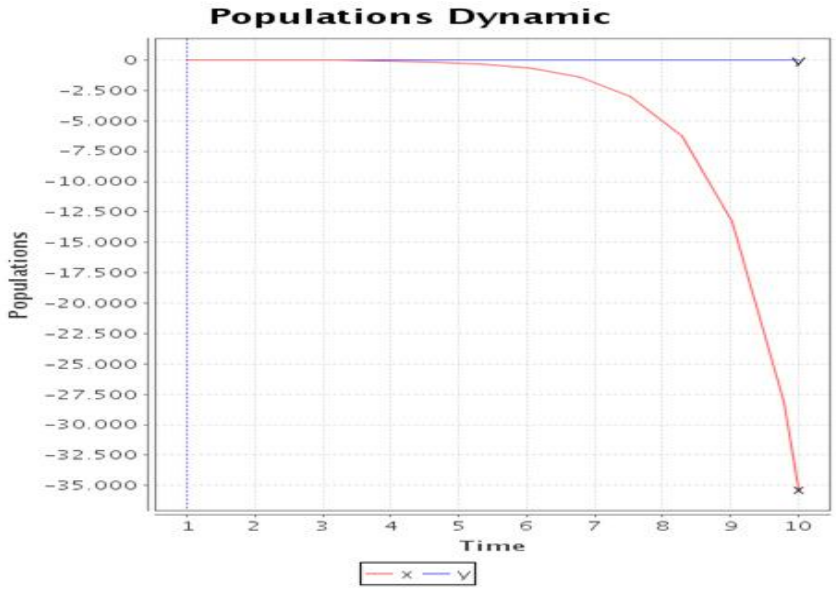


Figura 4.10 Gráfica de resultado para el método Ode78.

4.4.6 Simulación con ODEtoJava Método Runge-Kutta 4to orden ErkTriple

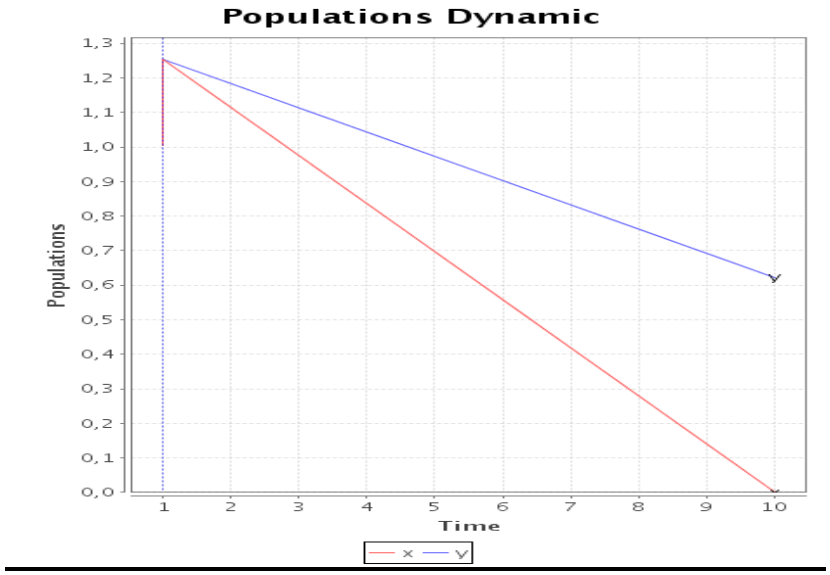


Figura 4.11 Gráfica de resultado para el método ErkTriple.

4.4.7 Simulación con ODEtoJava Método Runge-Kutta 4to-5to-7mo orden Dormand Prince

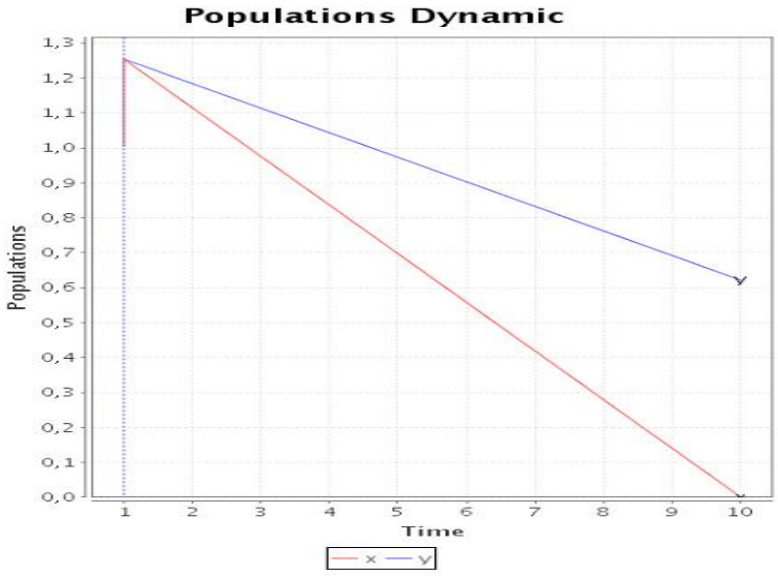


Figura 4.12 Gráfica de resultado para el método Dormand Prince.

4.4.8 Simulación con ODEtoJava Método Runge-Kutta para cualquier ODE (Sin interpolación) Erk

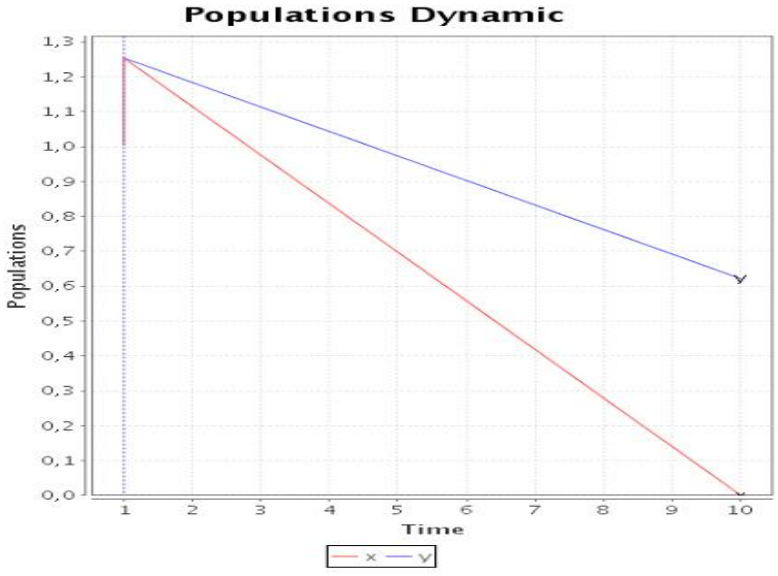


Figura 4.13 Grafica de resultado para el método Erk.

4.4.9 Simulación con ODEtoJava Método Runge-Kutta con paso doble ErkSD

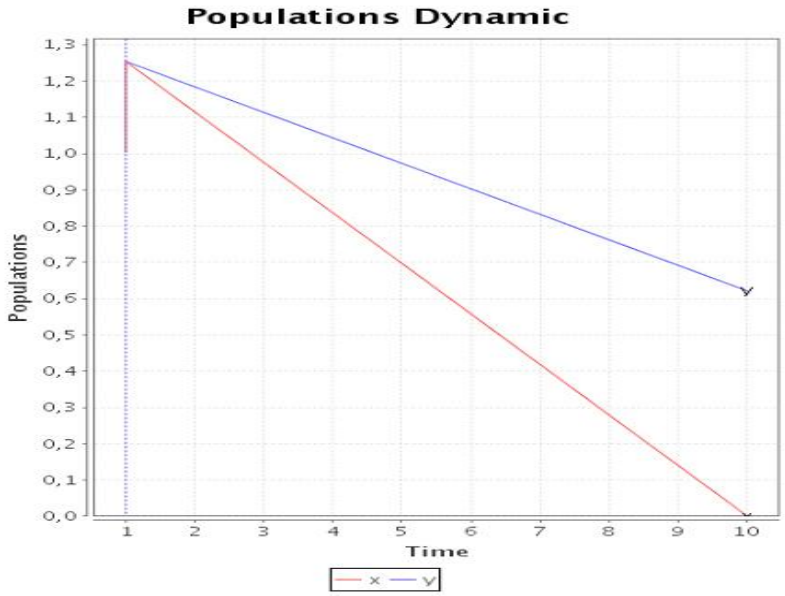


Figura 4.14 Gráfica de resultado para el método ErkSD.

Luego de analizar los resultados obtenidos para el mismo modelo matemático usando herramientas diferentes, podemos concluir que: los resultados brindados por Octave son más amplios y unos métodos necesitan más iteraciones que otros para llegar a la misma conclusión, es el caso de Ode23 que requiere 75 iteraciones y Ode78 solo necesita 15. En este sentido el investigador debe ser capaz de decidir qué método matemático le es más eficaz usar, o si debe sacrificar tiempo por precisión, en general, los resultados brindados por Octave son muy similares y todos tienden al mismo resultado. En el caso de ODEtoJava se apreció un tiempo menor de respuesta en las soluciones y esto está dado por ser la aplicación en sí, la encargada de gestionar las simulaciones al no depender de los resultados de una herramienta externa, como es el caso de las simulaciones que se realizan usando Octave. Los métodos de solución brindados por ODEtoJava obtuvieron a su vez soluciones más específicas y casi idénticas entre sí como se esperaba, pues de otra manera se hubieran identificado divergencias de solución. Sin embargo, de manera general, ambas soluciones tanto las brindadas por Octave y las de ODEtoJava son concluyentes. La dinámica de la población del sistema Presa-Depredador representada por el modelo matemático expuesto anteriormente, decrece en función del tiempo. Lo mismo posibilita determinar que ambas herramientas por vías diferentes son capaces de realizar

simulaciones de manera correcta. Llegado este punto el sistema brinda la posibilidad al investigador de comparar soluciones que fortalezcan una decisión.

4.5 Conclusiones del Capítulo

Este capítulo representa la guía para comprender cómo se ha realizado la implementación de cada uno de los componentes que le han dado vida a las funcionalidades propuestas a desarrollar. Un conjunto de diagramas, así como la explicación detallada de ellos y del código fuente perteneciente a las principales funciones de la aplicación, permitirán al lector relacionarse más con la solución y a futuros desarrolladores brindarles los elementos para mejorar o replicar la aplicación desarrollada.

Conclusiones

Con la realización de este trabajo de diploma se alcanzaron con éxito los objetivos trazados, pues se reimplementó el módulo de simulación de BioSyS de manera que fuera capaz de realizar simulaciones de Sistemas Biológicos, haciendo uso de los modelos matemáticos que definen a estos sistemas. Básicamente se creó una herramienta capaz de interactuar con el asistente matemático Octave, utilizando a este para resolver Sistemas de Ecuaciones Diferenciales, además se rediseñaron las funciones matemáticas inherentes a la librería ODEtoJava, de forma que se pudieran integrar eficientemente al simulador en cuestión. Estas nuevas funcionalidades fueron incorporadas al Simulador de Sistemas Biológicos BioSyS, creando un producto final con calidad, más diverso y eficiente, una herramienta que puede adaptarse al entorno de cualquier ambiente de investigación con éxito.

Recomendaciones

Al concluir este trabajo el equipo de desarrollo define como futuras recomendaciones al simulador:

- Realizar pruebas de caja negra y caja blanca para comprobar y lograr una mayor eficiencia del sistema.
- Unir todas las formas de simulación en un solo paquete de simulación.
- Brindarle al simulador la posibilidad de gestionar las simulaciones, de manera que los valores de cada simulación para un modelo puedan ser accedidos desde la propia herramienta.

Referencias Bibliográficas

1. **Moreno Parra, Rafael Alberto.** *Definición de Simulación.* [En línea] http://docencia.50webs.com/simula01.htm.

2. **Flórez Amaya, Andrés Felipe.** CECIF. *Modelación de Sistemas Biológicos como herramienta en el descubrimiento de nuevos blancos moleculares.* [En línea] http://74.125.47.132/search?q=cache:eWdrioWi0VEJ:www.cecif.org/magazine/index2.php%3Foption%3Dcom_content%26do_pdf%3D1%26id%3D6+Modelaci%C3%B3n+de+Sistemas+Biol%C3%B3gicos&cd=2&hl=es&ct=clnk&gl=cu.

3. **Navarro Bermudez, Mabel y Rodriguez Aldana, Yissel.** *BioSyS: Implementación del Módulo de Simulación.* Ciudad de La Habana, 2008.

4. Telecomunicaciones-29. *Modelo matemático.* [En línea] http://telecomunicaciones-29.blogspot.com/.

5. *Modelo Matemático Clasificaciones de los Modelos.* [En línea] http://www.semcali.gov.co/Biblionet/index.php/W:Modelo_matem%C3%A1tico.

6. **Ibáñez, Juan José.** Weblogs. *Concepto y Tipos de Modelos Científicos .* [En línea] http://weblogs.madrimasd.org/universo/archive/2008/05/10/91441.aspx.

7. **Ramos, Santiago.** Instituto de Zoología Tropical. *SECCIÓN ECOLOGÍA DE COMUNIDADES Y SISTEMAS: Laboratorio de Sistemas de Información y Modelaje Ecológico Ambiental Análisis de Sistemas.* [En línea] http://www.ciens.ucv.ve/instzool/ECSSR.html.

8. Ecuaciones diferenciales: una introducción moderna. *Sistemas de ecuaciones diferenciales.* [En línea] http://www.reverte.com/catalogo/img/pdfs/9788429151626.pdf.

9. *Sistemas de Ecuaciones Diferenciales Lineales.* [En línea] http://www.tecnun.es/asignaturas/metmat/texto/en_web/Sistemas_lineales/Sistemas_lineales.htm.

10. *MÉTODO DE EULER.* [En línea] http://docentes.uacj.mx/gtapia/AN/Unidadse/Euler/euler.htm.

11. Métodos Numéricos Avanzados. *Método de Heun.* [En línea] http://mna.wikidot.com/ode.

12. *Métodos Runge-Kutta.* [En línea] http://mate.uprh.edu/~pnm/notas4061/rungek/rungek.htm.

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

14. Addlink Software Científico. *Matemática.* [En línea] http://www.addlink.es/familias.asp?idfam=3&nomfam=Matematicas.

15. Taringa Inteligencia Colectiva. *Matlab - MATrix LABoratory .* [En línea] http://www.taringa.net/posts/downloads/1149013/Matlab---MATrix-LABoratory.html.

16. Upna Universidad Pública de Navarra. *MatLab Introducción al MatLab.* [En línea] http://www.unavarra.es/si/sisoftes.htm.

17. Software Cómputo Numérico . *GNU OCTAVE .* [En línea] http://eq1sistema.890m.com/1.4%20SOFTWARE%20COMPUTO%20NUMERICO.htm.

18. *Grupo de Usuarios de Software Libre - Perú.* [En línea] http://www.somoslibres.org/modules.php?name=News&file=print&sid=2518.

19. Virtualdag.org. *"EL REGRESO" xD.* [En línea] http://virtualdag.org/tag/cluster/.

20. Semillero de investigación especializado en computación de alto rendimiento. *Grid Computing.* [En línea] http://siecar.upbbga.edu.co/areas.html.

21. *Introduction to OpenUP (Open Unified Process).* [En línea] http://www.eclipse.org/epf/general/OpenUP.pdf.

22. Paraiso Linux Mi mundo Linux. *Herramientas para modelado UML.* [En línea] http://paraisolinux.com.ar/herramientas-para-modelado-uml/.

23. Slideshare. *Metodologia Uml - Presentation Transcript.* [En línea] http://www.slideshare.net/Waleskita/metodologia-uml-presentation.

24. Ciencia y Técnica Administrativa. *Herramientas Case.* [En línea] http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm .

25. *Herramientas Case El mejor soporte para el proceso de desarrollo de software.* [En línea] http://www.innovavirtual.org/moodle/file.php/178/archivos_curso/CAP_12_2006_I_SI905/CAP_12_2006_I_SI905_VA8_M.pdf.

26. Programacion en castellano. *Visual Paradigm for UML.* [En línea] http://www.programacion.com/noticia/1363/.

27. zonaClic. *Java.* [En línea] http://clic.xtec.net/es/jclic/java.htm .

28. Guia Ubuntu. *NetBeans.* [En línea] http://www.guia-ubuntu.org/index.php?title=NetBeans.

29. pyBernate Herramienta ORM para PostgreSQL. *Mapeo objeto-relacional.* [En línea] http://www.bensoft.cl/pybernate.

30. Xeridia. *Curso de Hibernate.* [En línea] http://www.xeridia.com/servicios/cursos/curso_hibernate.

31. Conjunto de roles: OpenUP/Basic Roles. *Rol: Analista.* [En línea] http://epf.eclipse.org/wikis/openupsp/openup_basic/rolesets/openup_basic_roles,_TZIJ0O8NEdmKSqa_gSYthg.html.

32. El Mundo Informático. *Patrones Grasp(Patrones de Software para la asignación General de Responsabilidad) Parte II.* [En línea] http://jorgesaavedra.wordpress.com/category/ingenieria-de-software/.

33. **Oktaba, Hanna.** Introducción a Patrones. *Patrones.* [En línea] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

34. **Dorado, Andrés.** Slideshare. *Categorias de Patrones Patrones de Arquitectura.* [En línea] http://www.slideshare.net/2008PA2Info3/tema-de-revision-2003.

35. educaedu. *Patrones de arquitectura.* [En línea] http://www.educaedu.com.mx/curso-de-arquitectura-de-software-y-patrones-de-diseno-con-uml-cursos-5731.html.

36. Drauta. *Desarrollando aplicaciones web con Modelo-Vista-Controlador.* [En línea] http://www.drauta.com/articulos/desarrollando-aplicaciones-web-con-modelo-vista-controlador/.

37. Informática Nicaragua. *Modelo Vista Controlador .* [En línea] http://informaticanicaragua.net/index.php?action=printpage;topic=84.0.

38. Kioskea. *Patrones de diseño.* [En línea] http://es.kioskea.net/contents/genie-logiciel/design-patterns.php3.

39. **Lagos Torres, Manuel.** El Rincón del Programador. *Introducción al diseño con patrones.* [En línea] http://www.elrincondelprogramador.com/default.asp?id=29&pag=articulos/leer.asp.

40. **Gracia, Joaquin.** IngenieroSoftware. *Diseño de Software Orientado a Objetos.* [En línea] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

41. Expendedora: Modelo de Dominio. *Modelo de Dominio.* [En línea] http://iie.fing.edu.uy/ense/asign/desasoft/practico/hoja8/ejemplos_clase2.pdf.

42. Entorno Virtual de Aprendizaje. *Conferencia 3_ FT Requerimientos Ingeniería de Software 1.* [En línea] http://teleformacion.uci.cu..

43. Entorno Virtual de Aprendizaje. *Conferencia 1 Ingeniería de Software 2.* [En línea] http://teleformacion.uci.cu.

44. Entorno Virtual Aprendizaje. *Interaccion Ingeniería de Software 1.* [En línea] http://teleformacion.uci.cu.

45. Sparx. *El Modelo Físico.* [En línea] http://www.sparxsystems.com.ar/resources/tutorial/physical_models.html.

Bibliografía

Addlink Software Científico. *Matemática.* [En línea] http://www.addlink.es/familias.asp?idfam=3&nomfam=Matematicas.

Alfaro, Victor M. Febrero 2002. Métodos numéricos para la solución de ecuaciones diferenciales ordinarias (EDO).

Booch Grady, Rumbaugh Jim,Jacobson Ivar, El Lenguaje Unificado de Modelado.

Bustamante, Luis Guillermo. Duque Velásquez, Camilo. Zuluaga, Camilo. Correa Zabala, Francisco José. Métodos Numéricos con Octave.

Ciencia y Técnica Administrativa. *Herramientas Case.* [En línea] http://www.cyta.com.ar/biblioteca/bddoc/bdlibros/proyectoinformatico/libro/c5/c5.htm.

Cifuentes, Jimmy. Modelado computacional de sistemas biológicos usando herramientas de computación bioinspirada.

Cifuentes Valiente, José María. Manual de iniciación a GNU Octave.

Conjunto de roles: OpenUP/Basic Roles. *Rol: Analista.* [En línea] http://epf.eclipse.org/wikis/openupsp/openup_basic/rolesets/openup_basic_roles,_TZIJ0O8NEdmKSqa_gSYthg.html.

Dorado, Andrés. Slideshare. *Categorias de Patrones Patrones de Arquitectura.* [En línea] http://www.slideshare.net/2008PA2Info3/tema-de-revision-2003.

Drauta. ***Desarrollando aplicaciones web con Modelo-Vista-Controlador***. [En línea] http://www.drauta.com/articulos/desarrollando-aplicaciones-web-con-modelo-vista-controlador/.

educaedu. ***Patrones de arquitectura.*** [En línea] http://www.educaedu.com.mx/curso-de-arquitectura-de-software-y-patrones-de-diseno-con-uml-cursos-5731.html.

Ecuaciones diferenciales: una introducción moderna. *Sistemas de ecuaciones diferenciales.* [En línea] http://www.reverte.com/catalogo/img/pdfs/9788429151626.pdf.

El Mundo Informático. *Patrones Grasp(Patrones de Software para la asignación General de Responsabilidad) Parte II.* [En línea] http://jorgesaavedra.wordpress.com/category/ingenieria-de-software/.

Entorno Virtual de Aprendizaje. ***Conferencia 1 Ingeniería de Software 2.*** [En línea] http://teleformacion.uci.cu.

Entorno Virtual de Aprendizaje. ***Conferencia 3_ FT Requerimientos Ingeniería de Software 1.*** [En línea] http://teleformacion.uci.cu.

Bibliografía

Entorno Virtual Aprendizaje. ***Interaccion Ingeniería de Software 1.*** [En línea] http://teleformacion.uci.cu.

Expendedora: Modelo de Dominio. ***Modelo de Dominio.*** [En línea] http://iie.fing.edu.uy/ense/asign/desasoft/practico/hoja8/ejemplos_clase2.pdf.

Flórez Amaya, Andrés Felipe. CECIF. *Modelación de Sistemas Biológicos como herramienta en el descubrimiento de nuevos blancos moleculares.* [En línea] http://74.125.47.132/search?q=cache:eWdrioWi0VEJ:www.cecif.org/magazine/index2.php%3Foption%3Dcom_content%26do_pdf%3D1%26id%3D6+Modelaci%C3%B3n+de+Sistemas+Biol%C3%B3gicos&cd=2&hl=es&ct=clnk&gl=cu.

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

Gracia, Joaquin. IngenieroSoftware. *Diseño de Software Orientado a Objetos.* [En línea] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

Grupo de Usuarios de Software Libre *- Perú.* [En línea] http://www.somoslibres.org/modules.php?name=News&file=print&sid=2518.

Guia Ubuntu. *NetBeans.* [En línea] http://www.guia-ubuntu.org/index.php?title=NetBeans.

Hamilton Castro, Alberto F. Grupo de Computadoras y Control. Dpto de Física y Sistemas ULL. 2 diciembre de 2003. Toolbox de Control de Sistema de Octave.

Herramientas Case El mejor soporte para el proceso de desarrollo de software. [En línea] http://www.innovavirtual.org/moodle/file.php/178/archivos_curso/CAP_12_2006_I_SI905/CAP_12_2006_I_SI905_VA8_M.pdf.

Ibáñez, Juan José. Weblogs. *Concepto y Tipos de Modelos Científicos .* [En línea] http://weblogs.madrimasd.org/universo/archive/2008/05/10/91441.aspx.

Informática Nicaragua. ***Modelo Vista Controlador .*** [En línea] http://informaticanicaragua.net/index.php?action=printpage;topic=84.0.

Introduction to OpenUP *(Open Unified Process).* [En línea] http://www.eclipse.org/epf/general/OpenUP.pdf.

Kioskea. ***Patrones de diseño.*** [En línea] http://es.kioskea.net/contents/genie-logiciel/design-patterns.php3.

Lagos Torres, Manuel. El Rincón del Programador. *Introducción al diseño con patrones.* [En línea] http://www.elrincondelprogramador.com/default.asp?id=29&pag=articulos/leer.asp.

Lovelle Cueva, Juan Manuel. Introducción a UML.

Bibliografía

Malumbres Varona, Juan Luis. Métodos Clásicos de Resolución de Ecuaciones Diferenciales Ordinarias.

Modelo Matemático Clasificaciones de los Modelos. [En línea] http://www.semcali.gov.co/Biblionet/index.php/W:Modelo_matem%C3%A1tico.

MÉTODO DE EULER. [En línea] http://docentes.uacj.mx/gtapia/AN/Unidadse/Euler/euler.htm.

Métodos Numéricos Avanzados. *Método de Heun.* [En línea] http://mna.wikidot.com/ode.

Métodos Runge-Kutta. [En línea] http://mate.uprh.edu/~pnm/notas4061/rungek/rungek.htm.

Moreno Lemus, Noel. BioSyS: Software para la simulación y análisis de sistemas biológicos.

Moreno Parra, Rafael Alberto. *Definición de Simulación.* [En línea] http://docencia.50webs.com/simula01.htm.

Navarro Bermudez, Mabel y Rodriguez Aldana, Yissel. *BioSyS: Implementación del Módulo de Simulación.* Ciudad de La Habana, 2008.

Oktaba, Hanna. Introducción a Patrones. *Patrones.* [En línea] http://www.mcc.unam.mx/~cursos/Algoritmos/javaDC99-2/patrones.html.

Paraiso Linux Mi mundo Linux. *Herramientas para modelado UML.* [En línea] http://paraisolinux.com.ar/herramientas-para-modelado-uml/.

Programacion en castellano. *Visual Paradigm for UML.* [En línea] http://www.programacion.com/noticia/1363/.

pyBernate Herramienta ORM para PostgreSQL. *Mapeo objeto-relacional.* [En línea] http://www.bensoft.cl/pybernate.

Ramos, Santiago. Instituto de Zoología Tropical. *SECCIÓN ECOLOGÍA DE COMUNIDADES Y SISTEMAS: Laboratorio de Sistemas de Información y Modelaje Ecológico Ambiental Análisis de Sistemas.* [En línea] http://www.ciens.ucv.ve/instzool/ECSSR.html.

Semillero de investigación especializado en computación de alto rendimiento. *Grid Computing.* [En línea] http://siecar.upbbga.edu.co/areas.html.

Sistemas de Ecuaciones Diferenciales Lineales. [En línea] http://www.tecnun.es/asignaturas/metmat/texto/en_web/Sistemas_lineales/Sistemas_lineales.htm.

Bibliografía

Slideshare. ***Metodologia Uml - Presentation Transcript.*** [En línea] http://www.slideshare.net/Waleskita/metodologia-uml-presentation.

Sparx. ***El Modelo Físico.*** [En línea] http://www.sparxsystems.com.ar/resources/tutorial/physical_models.html.

Storti, M. Introducción a Octave.

Software Cómputo Numérico . ***GNU OCTAVE .*** [En línea] http://eq1sistema.890m.com/1.4%20SOFTWARE%20COMPUTO%20NUMERICO.htm

Sotomayor, Borja. Introducción a la Computación Grid.

Taringa Inteligencia Colectiva. ***Matlab - MATrix LABoratory .*** [En línea] http://www.taringa.net/posts/downloads/1149013/Matlab---MATrix-LABoratory.html.

Telecomunicaciones-29. ***Modelo matemático.*** [En línea] http://telecomunicaciones-29.blogspot.com/.

Upna Universidad Pública de Navarra. ***MatLab Introducción al MatLab.*** [En línea] http://www.unavarra.es/si/sisoftes.htm.

Vega Contreras, Gerardo. Clúster de Alta Disponibilidad.

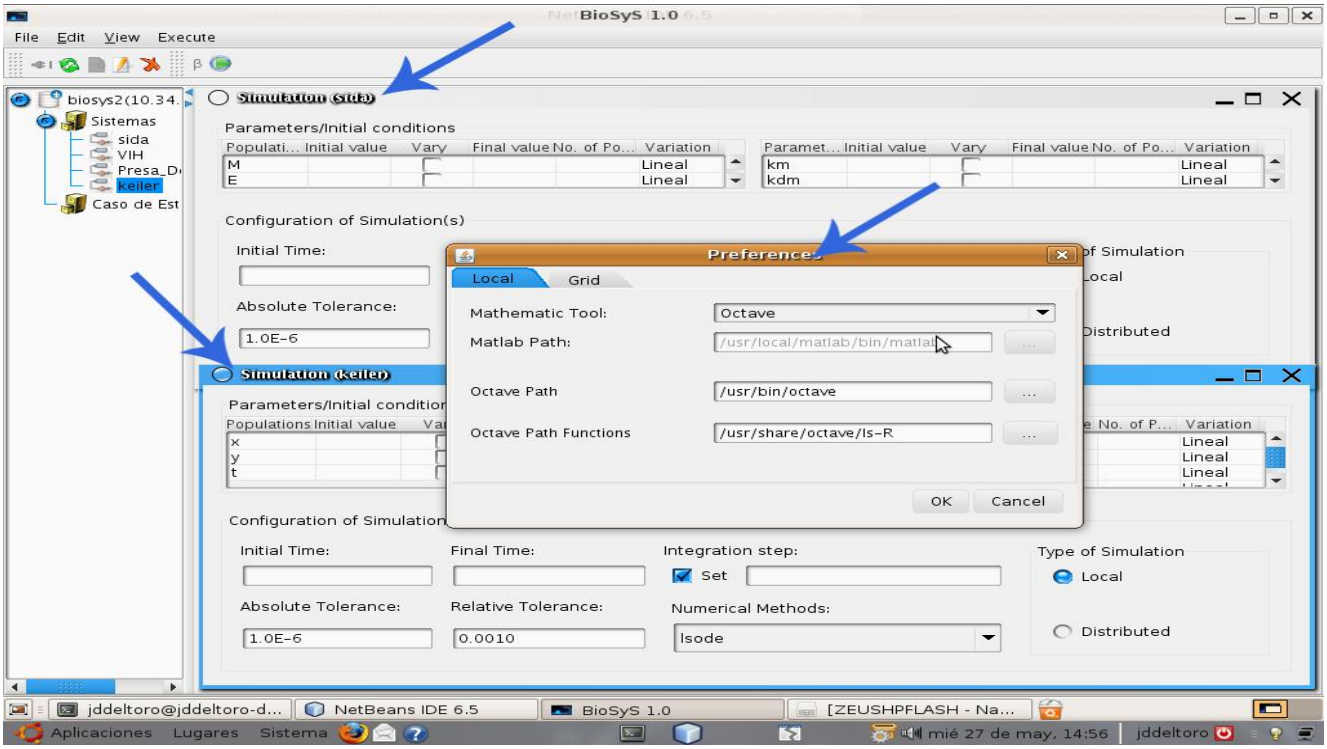
Virtualdag.org. ***"EL REGRESO" .*** [En línea] http://virtualdag.org/tag/cluster/.

Xeridia. ***Curso de Hibernate.*** [En línea] http://www.xeridia.com/servicios/cursos/curso_hibernate.

zonaClic. ***Java.*** [En línea] http://clic.xtec.net/es/jclic/java.htm .

Anexos

Anexo 1 *Patrón Fachada*

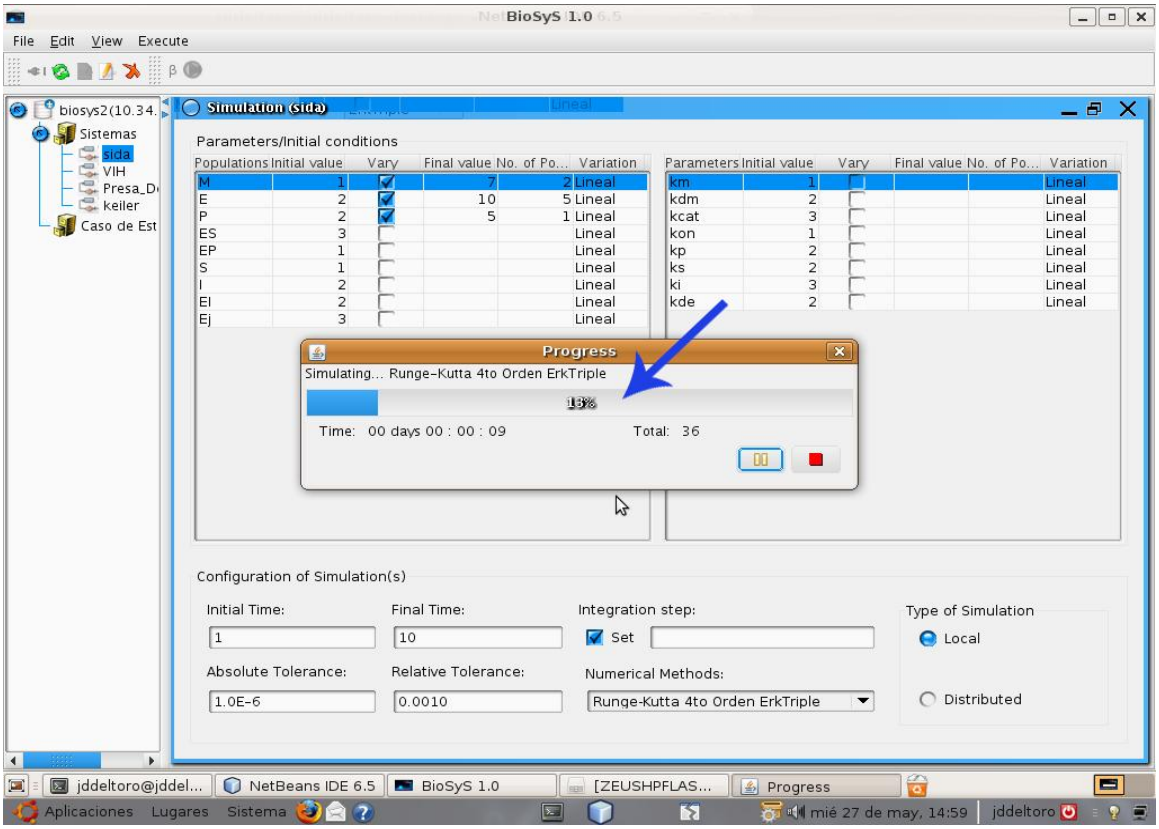


Anexo 2 *Patrón Estratégico*

```
public BSSimular(VistaSimulacion pvista,BSSimulacion bs_ssimulacion, Modelo modelo, BSDAOFactory bsDao, double ti,
                 double tf, double ta, double tr, double paso, String metodoNumerico){
        this.vista = pvista;
        this.vistao = new BSVistaOctaveGraphics();
        this.modelo = modelo;
        this.bs_ssimulacion = bs_ssimulacion;
        this.bsDao = bsDao;
        this.ti = ti;
        this.tf = tf;
        this.ta = ta;
        this.tr = tr;
        this.paso = paso;
        this.metodoNumerico = metodoNumerico;
    }

    public BSSimular(BSGraficadorDinamica pvista,BSSimulacion bs_ssimulacion, Modelo modelo, BSDAOFactory bsDao,
            double ti, double tf, double ta, double tr, double paso, String metodoNumerico){
        this.vistagd = pvista;
        this.modelo = modelo;
        this.bs_ssimulacion = bs_ssimulacion;
        this.bsDao = bsDao;
        this.ti = ti;
        this.tf = tf;
        this.ta = ta;
        this.tr = tr;
        this.paso = paso;
        this.metodoNumerico = metodoNumerico;
        this.tpointLisnpace = 0;
        this.count=0;
        this.odeFcnFormat =0;
        this.trace=0;
    }
```

Anexo 3 *Patrón Observador*



Glosario de Términos

***Biología de Sistemas*:** La Biología de Sistemas es una disciplina académica que pretende integrar diferentes niveles de información, con el fin de entender cómo funcionan los sistemas biológicos.

***Molécula*:** Una molécula es una partícula formada por un conjunto de átomos ligados por enlaces covalentes o metálicos (en el caso del enlace iónico no se consideran moléculas, sino redes cristalinas), de forma que permanecen unidos el tiempo suficiente como para completar un número considerable de vibraciones moleculares.

***Ecosistemas*:** Un ecosistema es una unidad natural que consiste en todas las plantas, animales y micro-organismos (factores bióticos) de un área funcionando junto con todos los factores no vivos (abióticos) del medio ambiente.

***SED*:** Sistema de ecuaciones diferenciales. Conjunto de varias ecuaciones diferenciales con varias funciones incógnitas y un conjunto de condiciones de contorno.

***DNA*:** Acido desoxirribonucleico, frecuentemente abreviado como ADN (y también DNA, del inglés *DeoxyriboNucleic Acid*), es un tipo de ácido nucleico, una macromolécula que forma parte de todas las células.

***Genes*:** Un gen es el conjunto de una secuencia determinada de nucleótidos de uno de los lados de la escalera del cromosoma referenciado.

***Proteínas*:** Las proteínas son macromoléculas formadas por cadenas lineales de aminoácidos. El nombre proteína proviene de la palabra griega *πρώτα* ("prota"), que significa "lo primero" o del dios *Proteo*, por la cantidad de formas que pueden tomar.

***Metabolitos*:** Un metabolito es cualquier molécula utilizada o producida durante el metabolismo.

***API*:** Una interfaz de programación de aplicaciones o API (del inglés **A**pplication **P**rogramming **I**nterface) es el conjunto de **funciones** y **procedimientos** (o métodos, si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.