Universidad de las Ciencias Informáticas
Facultad 6

Título: “BioSyS: Módulo de Modelación gráfica de Sistemas Biológicos.”

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

Autor(es):

Alfredo Martínez Figueredo.

Arisbel Luna Gallardo.

Tutor(es):

Dr. Kalet León Monzón

Msc. Noel Moreno Lemús.

Ing. Yuniesky Armentero Moreno.

.

La Habana, junio 2008

“Año 50 de la Revolución”

"Manantial de vida es el entendimiento al que lo posee;

mas la erudición de los necios es necedad."

Rey Salomón.

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 se firma la presente a los ____ días del mes de _____ del año _____.

Alfredo Martínez Figueredo	Noel Moreno Lemus
_____	_____
Firma del autor	Firma del tutor
Arisbel Luna Gallardo	Yuniesky Armentero Moreno
_____	_____
Firma del autor	Firma del tutor

DATOS DE CONTACTO

Tutores:

Dr. Kalet León Monzón.
Centro de Inmunología Molecular, Habana, Cuba.
Email: kalet@ict.cim.sld.cu

Lic. Noel Moreno Lemus.
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: noel@uci.cu

Ing. Yuniesky Armentero Moreno.
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: yarmentero@uci.cu

AGRADECIMIENTOS

A aquellos que han colaborado con nuestra formación como profesionales, a nuestros familiares y amigos que nos han apoyado a lo largo de la carrera, y en general a todos los que estuvieron a nuestro lado compartiendo las buenas y las malas. Gracias.

DEDICATORIA

A mi esposa Lianet por permanecer en su promesa de amor y ser con su sonrisa la mujer más bella.

A mis padres Isabel y Alfredo por estar tan cerca de mí siempre y ser el complemento de la felicidad de mi vida.

A mis abuelos Isabel, Caridad, René y Damaso Alfredo (en su memoria) eternamente bellos para mí.

A mis hermanas Alis y Yuliet que han crecido conmigo y son mis estrellas.

A mis tíos y tías en cariño entrañable.

A mis amigos con quienes comparto mi historia, llena de momentos inolvidables.

Y por encima de todos a quienes cito y amo, a Aquel que es mi Camino, mi Verdad y mi Vida junto a su gran familia a la cual pertenezco por la eternidad.

Alfredo Martínez Figueredo.

A mi abuela Ignacia por su preocupación y cuidado, por esperarme cada pase con un beso y un abrazo.

A mis padres Arnaldo y Angela por su apoyo incondicional, por estar cuando nadie más estuvo y por su sacrificio de darlo todo con amor y entereza.

A mi hermano Aisbel por su fidelidad y cariño, por confiar en mí y alentarme cuando me encontré cansado.

A mi familia por hacerme sentir seguro y darme su apoyo en todo tiempo.

A mis hermanos y amigos, y a todos los que en su momento estuvieron presente dándome lo mejor de sí para que hoy fuera real este momento.

Y por sobre todas las cosas a Aquel a quien le debo la vida y el aliento, mi mejor Amigo; a Él y a todos los demás. Gracias.

Arisbel Luna Gallardo.

RESUMEN

La Biología de Sistemas es un área de investigación científica interdisciplinaria que se preocupa del estudio funcional de los sistemas biológicos, al adentrarse al estudio los investigadores enfrentan constantes dificultades en el análisis de los sistemas biológicos a diferentes niveles de abstracción, desde el nivel molecular hasta los ecosistemas. Estas dificultades desaparecen o se aminoran a través de la modelación gráfica, la cual se lleva a cabo a través de herramientas computacionales. Actualmente, las herramientas que existen no cubren plenamente estas necesidades.

El presente trabajo tiene como objetivo fundamental darle continuidad a la implementación de un software que permite generar modelos matemáticos a partir de la modelación gráfica de Sistemas Biológicos que puedan ser descritos mediante dinámica de población, lo cual posibilita la modelación de forma sencilla.

PALABRAS CLAVE

Biología de Sistemas.

Sistemas Biológicos.

Modelación gráfica.

Software.

TABLA DE CONTENIDO

INTRODUCCIÓN

Bioinformática es una disciplina científica emergente que utiliza la tecnología de la información para organizar, analizar y distribuir información biológica con la finalidad de responder preguntas complejas en biología. Surge debido a la fusión entre la Biología y la Informática con la utilización directa del ordenador para el análisis, modelado y simulación de las estructuras y fenómenos observados en los seres vivos en sus distintos niveles de organización. **[1]**

Dentro de la Bioinformática surgió un área de investigación científica que se preocupa del estudio de procesos biológicos usando un enfoque sistémico, esta es la biología de sistemas y es uno de los campos más activos dentro de la Bioinformática. Comenzó a desarrollarse en los años sesenta del siglo XX, si bien su institucionalización académica no se produjo hasta el año 2000. Relaciona y se nutre de campos como la Genómica, Proteómica, Bioinformática y la Biología celular con la Computación y las Matemáticas, en ella se han llegado a descubrir principios fundamentales del funcionamiento de la vida celular antes desconocidos. **[1]**

Este campo de ciencia interdisciplinaria pretende integrar diferentes niveles de información con el fin de entender cómo funcionan los sistemas biológicos e intenta crear modelos matemáticos comprensibles de sistemas mediante el estudio de las relaciones y las interacciones entre las diferentes partes del mismo. **[2]**

La ingeniería siempre ha echo una práctica recurrente del modelado gráfico y diseño de sistemas en variadas técnicas. Ahora el resultado de la exploración de sistemas complejos por parte de las ciencias biológicas se ha encontrado con el análisis de problemas que requieren la aplicación de principios, técnicas de ingeniería, algoritmos y conceptos computacionales para resolverlos. Es por esto que la modelación gráfica se ha extendido ahora a la biología como una parte fundamental. Ha estimulado la creación de equipos multidisciplinarios formado por biólogos, ingenieros y otros profesionales que se encuentran ante el reto de descubrir nuevos conocimientos de sus propias disciplinas, además de reunir un conocimiento más integrado al campo científico.

Este trabajo de modelación junto al de la simulación ha atraído y encausado las fuerzas de la comunidad científica al punto de que han surgido empresas bien definidas en estas áreas, las cuales han logrado muy buenos resultados: **[3]**

- **Entelos**: Dedicada a las ciencias de la vida con el objetivo de mejorar la salud humana, utilizando como herramienta fundamental la Biosimulación **[12]**

- **Genomática**: Es la rama de la informática más importante en el estudio de Sistemas Biológicos**[13]**

- **Gene Network Sciences:** Permite simular el rendimiento clínico de drogas y drogas candidatas, a través de modelos computacionales de células y organismos a un alto nivel**. [14]**

Al enfrentarse al estudio los investigadores encuentran dificultades en el análisis de los sistemas biológicos a diferentes niveles de abstracción, desde el nivel molecular hasta los ecosistemas. Estas dificultades desaparecen o se aminoran a través de la modelación gráfica, la cual se lleva a cabo a través de herramientas computacionales y de la que se persigue obtener, o de cierta manera conformar el correspondiente modelo matemático, pudiendo estar éste en diferentes niveles de abstracción. Actualmente, las herramientas que existen, o sea, las conocidas y estudiadas hasta el momento en el presente trabajo no cubren plenamente estas necesidades. Por lo que surge la preocupante de una herramienta que satisfaga plenamente las exigencias de modelación, tanto gráfica como matemática para el estudio de los Sistemas Biológicos.

El presente trabajo de diploma dará continuidad a una versión existente del módulo de modelación de Sistemas Biológicos que actualmente solo permite la modelación gráfica, siendo la modelación matemática el fin esencial en sí mismo de una herramienta con estas peculiaridades. En el sitio Web www.sbml.com se pueden encontrar otras herramientas que aún no satisfacen del todo los requerimientos propios para estos fines.

Para concluir estas aseveraciones se efectuó un estudio sobre las mismas, encontrándose algunas limitantes que impiden una eficiente modelación gráfica y matemática de Sistemas Biológicos, por lo que el presente trabajo de diploma se centra en el siguiente **problema científico**: ¿Cómo generar modelos matemáticos a partir de modelos gráficos de sistemas biológicos realizados por la versión 1.0 de BioSyS?

El **objeto de estudio** de este proyecto es la modelación de Sistemas Biológicos y el **campo de acción** que comprende el mismo es la generación de modelos matemáticos a partir de modelos gráficos de Sistemas Biológicos.

Objetivo general

- Desarrollar nuevas funcionalidades para la generación de modelos matemáticos a partir de modelos gráficos de Sistemas Biológicos para la versión ByoSyS 1.0.

Objetivos específicos

- Realizar análisis de las nuevas funcionalidades.
- Realizar diseño de las nuevas funcionalidades.
- Implementar las nuevas funcionalidades.

Tareas a desarrollar

- Entrevistas con investigadores del centro del Centro de Inmunología Molecular (CIM).
- Estudio de los principales softwares de modelación gráfica de Sistemas Biológicos existentes.
- Revisión bibliográfica de materiales referentes a la modelación de Sistemas Biológicos.
- Obtención, descripción y validación de los requisitos del sistema.
- Construcción de los artefactos propios del análisis e implementación de las funcionalidades.
- Diseño de las clases del sistema.
- Implementación del sistema.
- Validar la implementación del sistema.

Con el desarrollo de esta aplicación se les facilitará a los usuarios el proceso de realizar la modelación gráfica de sistemas biológicos y obtener el modelo matemático correspondiente.

Estructuración por capítulo del contenido de la tesis

Capítulo 1: Fundamentación teórica: Este capítulo contiene una descripción de las técnicas y definiciones generales en torno a la Biología de Sistemas y modelación gráfica de Sistemas Biológicos. Además se realiza un estudio acerca de algunos softwares que en la actualidad hacen modelación gráfica y finalmente se definen las herramientas y tecnologías a utilizar para desarrollar la solución propuesta a la aplicación de software por el presente trabajo de diploma.

Capítulo 2: Características del sistema: En este capítulo se construye y se describe el modelo de dominio correspondiente a un Sistema Biológico. Se define el sistema ha desarrollar a través de sus actores, requisitos funcionales, requisitos no funcionales, casos de uso, descripción textual de casos de uso así como también el diagrama de casos de uso.

Capítulo 3: Diseño del sistema: En este capítulo se ofrece una descripción del estilo arquitectónico utilizado para continuar con el desarrollo de la aplicación y se definen los patrones de diseño empleados en la misma, además se realiza el diagrama de clases del diseño a seguir, así como la realización los diagramas de interacción y el modelo de despliegue definiendo como quedaría la aplicación en general.

Capítulo 4: Implementación del Sistema: En este capítulo se describe como fue implementada la aplicación en términos de componentes, se muestran fragmentos de códigos de las principales clases con sus descripciones y además se exponen los resultados de las pruebas de aceptación desarrolladas utilizando el método de caja negra para garantizar que el mismo cumple con los requisitos funcionales.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 INTRODUCCIÓN

En el presente capítulo se describe de forma general el propósito de la Biología de Sistemas como ciencia interdisciplinaria. Se dan las conclusiones del estudio de las herramientas computacionales existentes que se dedican a modelar gráficamente Sistemas Biológicos, mencionando en cada una las limitantes encontradas. Además, se tratarán las variadas tendencias y tecnologías que fueron utilizadas para el desarrollo de este software.

1.2 ¿CÓMO LLEGAMOS A LA BIOLOGÍA DE SISTEMAS?

La biología de sistemas tiene sus orígenes en:

- La modelación cuantitativa cinético enzimático, la cual floreció entre 1900 y 1970, la misma estudia la velocidad de las reacciones químicas que son catalizadas por las enzimas.
- Simulaciones desarrolladas para estudiar neurofisiología.
- La teoría del control y cibernética.

Uno de los teóricos que puede ser visto como precursor de la biología de sistemas es Ludwing von Bertalanffy por su teoría general de sistemas.

Desde la década del 60 se fueron obteniendo notables avances y aproximaciones a concretos resultados en el estudio de sistemas complejos moleculares encaminándose a la Biología de Sistemas actual, que surge como necesidad de convergencia en la línea de trabajo del análisis de los Sistemas Biológicos.

Cerca del año 2000, cuando los institutos de sistemas en biología estaban siendo establecidos en Seattle y Tokyo, la Biología de Sistemas emergió como un movimiento en su propio derecho. Desde entonces, varios institutos de investigación dedicados a la biología de sistemas han sido desarrollados. Desde el 2006, dada la escasez de gente trabajando en esta área, se han establecidos varios doctorados en varias partes del mundo.

La Biología de Sistemas (BS) es una ciencia emergente que trata de entender el funcionamiento de los sistemas como un todo y no de sus componentes por separado. Para ellos hace uso de herramientas de otras ramas de la ciencia como son: las de la computación, las ciencias de la vida y disciplinas ingenieriles. **[2] (Fig. 1.1)**

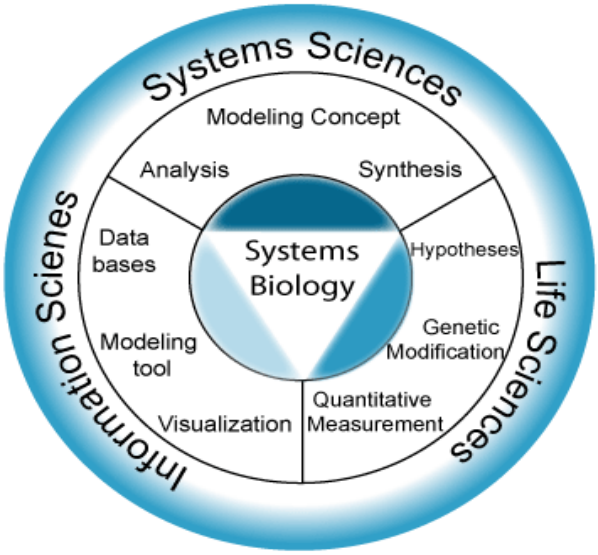


Figura 1.1 Ramas de la ciencia.

En estas ramas se encuentran las ciencias matemáticas, biológicas, físicas, químicas, entre otras como se muestra a continuación. **(Fig 1.2.)**

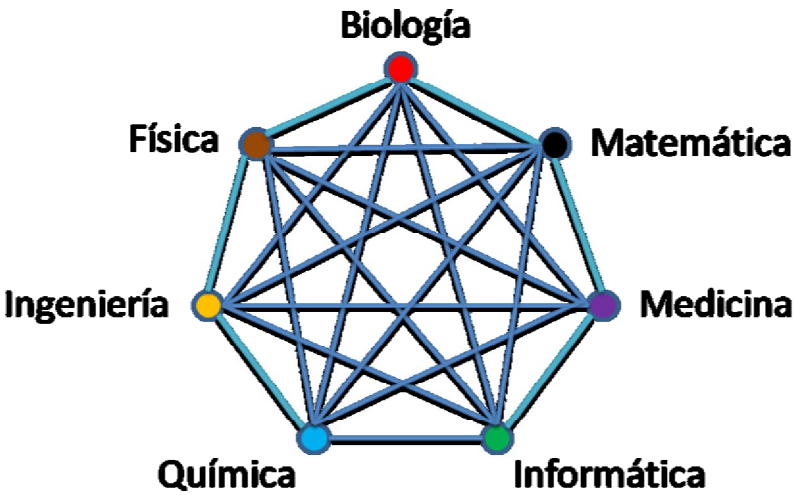


Figura 1.2 Ciencia

La Biología de Sistemas se preocupa del estudio de procesos biológicos usando un enfoque sistémico. La comunidad de investigadores ha propuesto un modelo al que llaman “las 4 Ms” (**Fig1.3**) para el estudio de los SB, dividiéndolo esencialmente en dos áreas, un área experimental y otra computacional. **[2] [4]**

Dentro del área experimental se encuentran la manipulación de datos y las mediciones, y en la computacional la minería y la modelación.

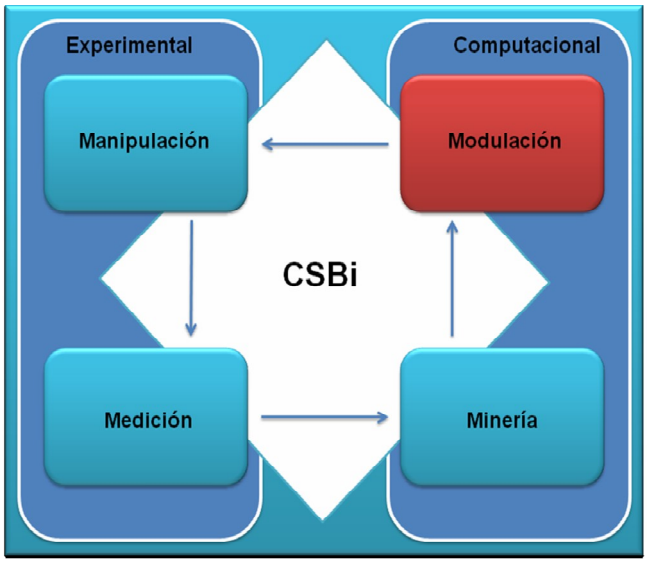


Figura 1.3 Las 4 Ms

Así llegamos a lo que existe hoy como Biología de Sistemas. A diferencia de los métodos clásicos de estudio usados por los biólogos, que se basan en la confirmación o refutación de hipótesis vía resultados experimentales, esta ciencia emplea técnicas de predicción rigurosas. Estas técnicas surgen fundamentalmente del uso de modelos matemáticos que describen el comportamiento del ente en estudio.

1.3 MODELACIÓN GRÁFICA DE SISTEMAS BIOLÓGICOS

Los modelos no son más que esquemas o ecuaciones que explican aquello que se quiere estudiar, lo cual puede excluir características del ente en estudio que se consideren no sean importantes, siendo así tan solo una invención que sirve para explicar una serie de datos que se quiere interpretar. Permiten por tanto, predecir el comportamiento del proceso como un sistema dinámico, generalmente tratado como una red compleja. Para disminuir la complejidad del estudio de los sistemas biológicos es necesaria la creación de un modelo matemático que describa el estudio de los mismos.

Resulta de mayor claridad exponer un concepto que pudiera considerarse de interés por su acertada orientación al término que se está explicando:

"El modelación matemática es una de las herramientas que se utiliza hoy en día para el estudio de problemas en medicina, biología, fisiología, bioquímica y farmacocinética; sus objetivos primordiales son describir, explicar y predecir fenómenos y procesos en dichas áreas. El modelo matemático permite representar un problema médico o biológico de una manera objetiva en que se definen una serie de relaciones matemáticas entre las mediciones cuantitativas (del problema) y sus propiedades." **[4]**

Con el transcurso del tiempo la información que se tiene de los Sistemas Biológicos aumenta progresivamente. Esto provoca dificultades en su comprensión, una de las maneras en que se puede resolver es la modelación, la cual históricamente ha sido la vía de solución más inmediata por el hombre y más concernientemente a los investigadores para el estudio de sistemas complejos.

La presente aplicación de software se basa en la modelación de sistemas biológicos que se describe mediante dinámica de población. Este tipo de modelo permite realizar la modelación de un sistema biológico de una forma sencilla.

Aun existen grandes limitantes relacionadas con los softwares que tienen que ver con esta área de modelación, los sistemas desarrollados no han permitido un fácil manejo al ser complejos. También le han presentado al usuario un evidente grado de esquematicidad, a causa de esto no le permite hacer ciertos cambios óptimos o satisfactorios durante su investigación. Por esto se desarrolla el presente trabajo de diploma que tiene el propósito de dar continuidad a una nueva herramienta que resuelva todas las limitantes que se han detectado, haciendo más flexible la creación de los modelos gráficos de un Sistema Biológico. **[2]**

1.4 SOFTWARE PARA LA MODELACIÓN GRÁFICA DE SISTEMAS BIOLÓGICOS

1.4.1 BIOTAPESTRY

Es una herramienta interactiva por construir, visualizar, y simular las redes reguladoras genéticas. BioTapestry se diseña alrededor del concepto de un modelo de red de desarrollo. Es capaz de representar sistemas que exhiben la complejidad creciente con el tiempo, como por ejemplo la red reguladora genética que controla el desarrollo en los embriones, y los sistemas que exhiben la complejidad creciente con el tiempo en la especificación del tubo neural ventral.

BioTapestry puede captar a cada hora vistas localizadas de la red durante desarrollo basado en mesas de los datos que describen los estados locales y temporales de la red **[5]**.

Desventaja: Las representaciones de las redes abstractas en modelos de 3D del organismo están en vías de desarrollo, por lo que la visibilidad profunda de los resultados no es manuable y por tanto el análisis de los mismos se hace en muchos casos poco aprovechable.

1.4.2 BIONETGEN

Es una herramienta para generar automáticamente modelos matemáticos de sistemas biológicos partiendo de especificadas reglas del usuario que definen las interacciones biomoleculares. Se especifican las reglas en el idioma de BioNetGen que habilita representación precisa, visual, y extensible de interacciones biomoleculares. Un usuario puede indicar las partes de proteínas involucradas en una interacción explícitamente, las condiciones de las que una interacción depende y la conectividad de proteínas en un complejo, donde a partir de la proteína se puede predecir si un nuevo fenotipo, derivado de uno de los nucleótidos, se puede relacionar con una enfermedad genética en seres humanos.

Desventaja: Únicamente modela aquellas reacciones que ocurren en red de genes y lo hace utilizando funciones booleanas **[5].**

1.4.3 CELLWARE

Es una herramienta que no sólo encapsula la topología de la red sino que entiende la expresión entre sus componentes constitutivos. Es capaz de capturar interacciones diversas entre los objetos, aunque requieren varias representaciones matemáticas y las plataformas modeladas. Cellware no sólo se ha diseñado para dirigir modelaciones y simulaciones de las sendas reguladoras y metabólicas del gen sino que también ofrece un ambiente integrado para las representaciones matemáticas diversas, estimación del parámetro y optimización. Además, se proporcionarían un despliegue gráfico de uso fácil y capacidad para ejecutar a los modelos grandes y complejos por defecto. Un rasgo muy especial de Cellware es que sería la primera grid basada en modelación y herramienta de simulación en el campo de Biología de los Sistemas, para un mejor conocimiento.

Desventaja: No incluye una base de datos con la cual realizar consultas de los resultados obtenidos de las simulaciones. Tiene implementado sólo dos herramientas para el análisis: estimación de parámetros y diseño de gráficas **[5]**.

1.4.4 COPASI

Es una aplicación del software para la simulación y análisis de redes bioquímicas. Es libre para el uso comercial. Es estocástico y simula el curso de tiempo determinístico. Entre sus características fundamentales están: el análisis de estado firme, de control metabólico, de modo elemental, de conservación de masa, el cálculo de exponentes de Lyapunov y el scan de parámetros. Además engloba parámetros globales para cambiar las proporciones de cinética múltiples al momento **[5]**.

Desventaja: No es instalable en todas las versiones de Mac Os, además no corre en MS Windows\u2122 95 ni en versiones anteriores, por lo que no es un software del todo multiplataforma.

1.4.5 IMMUNOGRID

Es una herramienta que simula una implementación virtual del sistema inmune humano haciendo uso de la tecnología Grid. Una de sus funcionalidades es su capacidad de simular procesos a escala natural y proporciona herramientas para el diseño de vacunas. Algo novedoso en ella es que logra la conexión de las interacciones a nivel molecular con los modelos a nivel de sistema. Se hace uso de bases de datos para almacenar, manipular y modelar datos inmunológicos, lo que facilita la búsqueda de modelos matemáticos y predictivos. Estas bases de datos son especializadas y utilizan estándares, por lo que facilitarán obtener información más detallada, a un alto nivel de calidad y credibilidad gráfica. **[5]**.

Desventaja: Esta herramienta solo fue creada para estudiar el sistema inmune humano, por lo que solo se puede utilizar en investigaciones relacionadas con el organismo humano.

1.4.6 VCELL

Es una herramienta que posibilita crear modelos biológicos de los cuales se generará un código matemático que podrá ser representado gráficamente en primera, segunda y tercera dimensión; necesario para correr las simulaciones que pueden ser complejas, variando uno o muchos parámetros según una lista especificada de valores posibles o de un rango definido. Las simulaciones son corridas en Internet en un número X de servidores.

Otra de las funcionalidades del software es que permite realizar nuevas representaciones geométricas si el usuario lo desea. Los modelos pueden ser reutilizados, actualizados y publicados para el uso de

otros usuarios, o compartidos privadamente. Los resultados de las simulaciones son guardados en una Base de Datos y se pueden exportar en varios formatos. **[5]**

Desventaja: La herramienta no tiene ninguna funcionalidad definida que lleve a cabo el análisis de los resultados obtenidos.

En el estudio anteriormente presentado se abarcaron los softwares oficialmente registrados hasta el momento y se concluye que existen deficiencias o limitantes en estos software ya que no satisfacen todas las necesidades de los investigadores biológicos para la modelación gráfica y matemática de Sistemas Biológicos. **[5]**

1.5 METODOLOGÍAS, TECNOLOGÍAS Y HERRAMIENTAS UTILIZADAS

Todo desarrollo de software es arriesgado y difícil de controlar, aún se agrava a esto si no media una metodología definida, lo cual lleva al resultado de clientes insatisfechos con el producto y desarrolladores aún más insatisfechos. También es necesario definir qué tecnologías y herramientas se deben utilizar para el desarrollo de una aplicación de esta envergadura. Se presenta una explicación acerca de estos puntos, los cuales describen la conformación de esta aplicación.

1.5.1 METODOLOGÍAS DE DESARROLLO DE SOFTWARE

Los desarrolladores se ven comprometidos en la realización de un software al enfrentarse con la necesidad de desarrollar con calidad y en el menor tiempo posible. Siendo inmediato el conocimiento referente a la organización de las actividades para cada desarrollador por separado y para el equipo de desarrollo. Por tanto, se requiere de una metodología para dirigir las actividades vinculadas al proceso de desarrollo de software a fin de añadir robustez a la producción.

Por una parte tenemos aquellas propuestas más tradicionales que se centran especialmente en el control del proceso, estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir, y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser efectivas y necesarias en un gran número de proyectos, pero también han presentado problemas en otros muchos. Una posible mejora es incluir en los procesos de desarrollo más actividades, más artefactos y más restricciones, basándose en los puntos débiles detectados. Sin embargo, el resultado final sería un proceso de desarrollo más complejo que puede incluso limitar la propia habilidad del equipo para llevar a cabo el proyecto.

Otra aproximación es centrarse en otras dimensiones, como por ejemplo el factor humano o el producto software. Esta es la filosofía de las metodologías ágiles, las cuales dan mayor valor al individuo, a la colaboración con el cliente y al desarrollo incremental del software con iteraciones muy cortas. Este enfoque está mostrando su efectividad en proyectos con requisitos muy cambiantes y cuando se exige reducir drásticamente los tiempos de desarrollo pero manteniendo una alta calidad.

De tantas metodologías con tantos enfoques para dirigir las actividades vinculadas al proceso de desarrollo de software se estudiaron las siguientes:

XP (Extreme Programing o Programación extrema)

Lo fundamental en este tipo de metodología es:

1 La comunicación, entre los usuarios y los desarrolladores.

2 La simplicidad, al desarrollar y codificar los módulos del sistema.

3 La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales**. [6]**

Es una de las metodologías de desarrollo de software más exitosas en la actualidad utilizada para proyectos de corto plazo, corto equipo y cuyo plazo de entrega es sumamente mínimo, esto es una desventaja concreta para desarrollar la presente aplicación, ya que el mismo es conforme a sus consideraciones un proyecto extenso y requiere un plazo de tiempo mayor que el estipulado en esta metodología presentada. La filosofía de XP es satisfacer al completo las necesidades del cliente, por eso lo integra como una parte más del equipo de desarrollo, haciéndolo inaplicable para el desarrollo de la presente aplicación, ya que el cliente no se puede ver de otra forma que como agente externo al equipo de trabajo.

Microsoft Solution Framework (MSF)

MSF es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, los cuales representan un compendio de las mejores prácticas en cuanto a administración de proyectos se refiere. Y tiene las siguientes características:

- Adaptable: es parecido a un compás, usado en cualquier parte como un mapa del cual su uso es limitado a un lugar específico.

- Escalable: puede organizar equipos tan pequeños entre 3 ó 4 personas, así como proyectos que requieren 50 personas o más.
- Flexible: es utilizada en el ambiente de desarrollo de cualquier cliente.
- Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología. **[2]**

Una de las desventajas que se constató en el estudio realizado acerda de MSF, es que en cada fase se debe documentar cada cosa que se haga, esto indudablemente atenta al equipo de producción que quiera agilizar el desarrollo del software. Otra deficiencia que se pudo apreciar es que con frecuencia no ofrece ninguna orientación sobre como llegar a un objetivo determinado, tales como producir un artefacto, realizar alguna tarea específica, entre otras; lo cual implica un riesgo que no le agradaría encontrar a ningún equipo de desarrollo durante la realización de una aplicación de software.

RUP (Proceso unificado de desarrollo de software)

Las cualidades que denotan más méritos en RUP son las siguientes:

- Dirigido por los casos de uso: Parte de la idea que un sistema debe brindar todos los servicios que requiere el usuario. Define caso de uso como el conjunto de acciones que debe realizar el sistema para devolverle al usuario un resultado de valor. Siendo esto la directriz que guía los pasos del desarrollo del software.

- Centrado en la arquitectura: Se toma el diseño completo ignorando los detalles, centrándose en las características más importantes. Se evalúa no solo las necesidades de los usuarios o inversores, sino también todos los aspectos técnicos que rodean y permiten que el sistema funcione con toda eficiencia. Así la arquitectura va evolucionando en su interacción con los casos de uso hasta llegar a un equilibrio entre funcionalidad y características técnicas.

- Iterativo e incremental: La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos. Cada uno de éstos se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. En cada iteración los desarrolladores seleccionan un grupo de casos de uso, los cuales se diseñan, implementan y prueban. La planificación de iteraciones hace

que se reduzcan los riesgos de los costos de un solo incremento, no sacar al mercado un producto en el tiempo previsto, mantener la motivación del equipo pues puede ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos. **[3]**

Estas características van a permitir una planificación ajustada al avance que vaya presentando el producto, además, se irá verificando que las funcionalidades que van siendo implementadas en el software sean exactamente las que desea el usuario. El Proceso Unificado del Rational aporta además un enfoque disciplinado a la asignación y dirección de tareas y responsabilidades, tanto individuales como al equipo de trabajo. Una característica importante es que permite corregir errores en cada iteración y es flexible a cambios en los requerimientos.

¿Podría ser RUP la metodología indicada? la línea de trabajo se identifica fácilmente con las metas que persigue el equipo de trabajo, pero aún queda un proceso de software que se adapta más, OpenUp.

OpenUP

OpenUP conserva las características principales del modelo de desarrollo RUP como una extensión del mismo; incluye el desarrollo iterativo, permite identificar los requisitos operacionales del sistema, prevee las interacciones con los usuarios y los posibles riesgos en el desarrollo del sistema. Plantea que se debe tener una versión ejecutable del proyecto en poco tiempo. En esta metodología solo se debe usar los procesos que sean necesarios y también no se debe usar muchos artefactos, además debe acoplarse a las necesidades del usuario pudiendo ser modificado y extendido. Usa las mismas fases e hitos del RUP pero sin tanta documentación. Se enfoca en el valor del software y disminuye riesgos.

OpenUP es una metodología de desarrollo más ágil y ligera, consiste en equipos a los cuales se les asigna una fase del desarrollo que tienen que complementarse entre sí para obtener un buen producto final, no puede ser una sola persona la que realice todo el trabajo, pues esto podría ocasionar que se pierda de vista ciertas características importantes, por ejemplo para un proyecto pequeño OpenUp constituye equipos de 3 a 6 personas e implica de 3 a 6 meses de esfuerzo del desarrollo. Por estas razones se eligió OpenUp como la metodología de desarrollo para la continuidad de la aplicación, además de que fue una decisión del polo de la facultad de cambiar de metodología.

1.5.2. HERRAMIENTAS CASE

(CASE: Computer Aided Software Engineering o Ingeniería de Software Asistida por Computadoras)

Estas herramientas están elaboradas para aumentar la productividad en el desarrollo de software, además de que reducen el coste del mismo en términos de tiempo y dinero. Pueden guiar en todos los aspectos del ciclo de vida de desarrollo del software, ya sea en el diseño, cálculo de costos, generación de código automático según un diseño dado, compilación automática, documentación o detección de errores, entre otras. Para la selección correcta de la herramienta CASE a aplicar se ha procedido a un estudio de aquellas que han demostrado mayor eficiencia y que a la vez sean ampliamente usadas. Encontrando que las más utilizadas y eficientes apuntan a ser Racional Rose y Visual Paradigm.

La primera herramienta citada mantiene la consistencia de los modelos del sistema de software, genera automáticamente la documentación y el código a partir de modelos; pero se descartó ya que se vio una carencia de integración cuando se incorpora algunas funcionalidades a través de otras aplicaciones, además de que el equipo de desarrollo al implementar directamente sobre la plataforma GNU/LINUX (sistema operativo Ubuntu) no cuenta con la versión de Racional Rose que corra sobre dicha plataforma.

Visual Paradigm es una herramienta CASE potentísima para visualizar y diseñar elementos de software, para ello utiliza el lenguaje UML y proporciona a los desarrolladores una plataforma que les permita diseñar un producto con calidad de una forma rápida. Está disponible en varias ediciones: Enterprise, Professional, Community, Standard, Modeler y Personal. Genera código y realiza ingeniería inversa para diez lenguajes de programación, Java, C++, CORBA IDL, PHP, XML Schema y ADA. Facilita la interoperabilidad con otras herramientas CASE como el Rational Rose y se integra con las siguientes herramientas Java: Eclipse/IBM WebSphere, Jbuilder, NetBeans IDE, Oracle Jdeveloper, BEA Weblogic. Además genera código para C#, Visual Basic.net, Object Definition Lenguage(ODL), Flasch Action Script, Delphi, Perl y Phyton. Se integra con el Visio para importar imágenes del mismo y realizar los diagramas de despliegue. Genera documentación para el proyecto en HTML, MS Word y PDF; además exporta e importa los diagramas en el estándar XML y como imágenes (ya sea con extensiones jpg o png). Es gratis en su edición Community y es multiplataforma.

Convenientemente al sistema operativo sobre el cual el equipo de desarrollo trabaja (distribución de GNU/LINUX: Ubuntu) se decidió utilizar Visual Paradigm como herramienta CASE para visualizar y

diseñar los elementos de software ya que es multiplataforma y utiliza el Lenguaje Unificado de Modelado (UML), además el desempeño sobre su uso es mucho más sencillo y ágil que el del Racional Rose.

1.5.3. LENGUAJE DE MODELADO DEL SOFTWARE

UML (Unified Modeling Languaje o Lenguaje Unificado de Modelado)

Surgió en 1994 como la unificación de todos los métodos de diseño anteriores. Con él se visualiza, especifica y documenta cada una de las partes que comprende el desarrollo de software.

Es posible fijar la serie de requerimientos y estructuras necesarias para determinar un sistema de software previo al proceso de implementación del código. Se presta por su naturaleza para el intercambio entre los usuarios y desarrolladores.

Ha llegado a convertirse casi plenamente en el lenguaje estándar del análisis y diseño de sistemas informáticos dentro de la industria del software. Anteriormente los revisores de algún sistema debían enfrentarse a aprender las semánticas y notaciones de la metodología empleada antes de entender el diseño en sí. UML permite que diseñadores diferentes modelen sistemas diferentes y puedan ampliamente entender cada uno los diseños de los otros.

1.5.4. LENGUAJES DE PROGRAMACIÓN

En el ámbito de las aplicaciones de software que han sido desarrolladas bajo programación orientada a objetos encontramos lenguajes que han sido ampliamente utilizados, tales como el C, C++, C# y Java. Al observar la naturaleza de C y C++ se detecta que a pesar de que tienen características relevantes como su flexibilidad, soporte de plantillas, sobrecarga de operadores, identificación de tipos (RTTI), etc. **–** presentan desfavorablemente un cierto grado de complejidad, siendo los que menos grado de automatismo ofrecen en su desempeño **–** obligan a hacerlo casi todo manualmente - lo cual hace complicado su aprendizaje y utilización.

Por otro lado se tiene la opción entre Java y C#. *C#* es un lenguaje de programación simple pero eficaz, diseñado para escribir aplicaciones empresariales. Es una evolución de los lenguajes C y C++. Utiliza muchas de las características de C++ en las áreas de instrucciones, expresiones y operadores. Presenta considerables mejoras e innovaciones en áreas como seguridad de tipos, control de versiones, eventos y recolección de elementos no utilizados (liberación de memoria) **[11].** Tiene una

alta productividad, alcanzando un óptimo tiempo de desarrollo, incluso mayor al de Java. ¿En qué se hace mas óptimo e indicado aplicar Java y no C#? En la portabilidad. En este punto C# tiene una desventaja y es que todavía están en desarrollo los proyectos (Mono y Portable.Net), cuyos trabajos están enfocados para hacer de C# un lenguaje portable. Java posee un elemento que fielmente le ayuda y es que posee una amplia comunidad de seguidores y desarrolladores por todo el mundo, por lo que existe una gran cantidad de herramientas, aplicaciones, frameworks y librerías libres que apoyan y facilitan el desarrollo sobre este lenguaje.

Java además cumple con todo el tratamiento que encierra un lenguaje de programación orientado a objetos, trabaja sus datos como objetos y con interfaces a esos objetos, soporta la encapsulación, herencia y polimorfismo. Presenta una magnífica riqueza semántica, es robusto - realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución, ayudando a detectar errores lo antes posible en el ciclo de desarrollo y maneja la memoria para eliminar las preocupaciones por parte del programador de la liberación o corrupción de la misma - , de fácil aprendizaje debiéndose en parte a que es interactivo y animado. Es multithreaded (multihilos) - los hilos son básicamente pequeños procesos o piezas independientes de un gran proceso - o sea, permite muchas actividades simultáneas en un programa. Algo muy valioso y característico como lenguaje es que es interpretado, significa esto que el código fuente es compilado a un código intermedio que finalmente será interpretado y ejecutado por una máquina virtual, garantizando esto que un programa pueda ser desarrollado en cualquier plataforma.

Como conclusión del estudio se determina que Java será el lenguaje de programación que se utilizará para desarrollar la aplicación por su adaptabilidad a cualquier equipo de desarrollo, robustez y confiable portabilidad, contrastando con los anteriores lenguajes de programación presentados.

1.5.5. SISTEMA DE CONTROL DE VERSIONES

Una versión de un producto es el estado en que se encuentra en un momento dado de su desarrollo o modificación. Se llama control de versiones a la gestión de los diversos cambios o configuraciones que se realizan sobre los elementos de algún producto.

Los sistemas de control de versiones facilitan la administración de las distintas versiones de cada producto desarrollado, controlando el acceso a ficheros que están bajo su cuidado. El código fuente de un programa necesita controlarse en cuanto a su almacenaje, posibilidad de realizar cambios, registro

histórico de las acciones realizadas y tal vez la generación de informes que traten sobre el estado y los cambios introducidos entre las dos versiones. Todo esto es posible hacerse a través del uso de un sistema de control de versiones. Esto opera generalmente con una copia maestra en un repositorio central, y un programa cliente con el que cada usuario sincroniza su copia local. Esto permite compartir los cambios sobre un mismo conjunto de ficheros y guardar el registro de los cambios realizados por cada usuario, permitiendo así el retorno seguro al estado anterior en caso de necesidad.

Constituye una garantía de seguridad y una gran utilidad contar en el proyecto con un sistema de control de versiones. Ilustrando en la práctica, se verá muy frecuente la necesidad de un sistema control de versiones. Cuando se está escribiendo un programa o un documento muy largo (como un informe, o una tesis), y se quisiera hacer una marca en el proyecto al llegar a un punto estable, éste permite guardar una versión estable del trabajo.

La idea de guardar versiones es que si los nuevos cambios quedan mal, la marca servirá para volver al punto estable y recomenzar desde ahí. Si las cosas salen bien y se alcanza una nueva estabilidad, se pone otra marca. Para el control de versiones de esta herramienta se utilizará Subversion (SVN). El Concurrent Versions System (CVS) es el padre de los controladores de versiones, pero el mismo fue reemplazado por SVN debido a que el primero posee varias deficiencias. Por tanto el SVN es el indicado para un desarrollo con alta competitividad y calidad, destacando en el mismo:

- Es software libre bajo una licencia de tipo Apache/BSD.
- A diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.
- Permite selectivamente el bloqueo de archivos.
- Maneja eficientemente archivos binarios (a diferencia de CVS que los trata internamente como si fueran de texto).
- Se envían sólo las diferencias en ambas direcciones (en CVS siempre se envían al servidor archivos completos).
- La creación de ramas y etiquetas es una operación más eficiente; Tiene costo de complejidad constante (O(1)) y no lineal (O(n)) como en CVS.
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- Se sigue la historia de los archivos y directorios a través de copias y renombrados **[8].**

1.5.6. ENTORNO DE DESARROLLO

Para el lenguaje de programación Java existen varios entornos de desarrollo integrado (IDE del inglés Integrated Development Environment) que permiten su uso, tales como JBuilder, NetBeans y Eclipse. Una de las deficiencias más críticas que desclasifican a JBuilder como IDE para el presente software es que no es multiplataforma, solo corre sobre Window.

NetBeans es un proyecto de código abierto de gran éxito con una gran base de usuarios. Además es un proyecto GNU (libre), tiene un excelente diseñador de interfaces integrado, es muy rápido y fácil de usar, pero el mismo fue descartado ya que la la versión anterior de la aplicación estaba desarrollada en Eclipse y no se había tomado en cuenta en aquel entonces NetBeans porque como IDE no permitía la integración del entorno de desarrollo con un control de versiones Subversion. La actual versión de este IDE (NetBeans 6.0.1) si permite la integración requerida con SVN, pero fue lanzada oficialmente el 4 de marzo del 2008 cuando ya el desarrollo de la aplicación había tomado un curso irreversible para migrar**. [15]**

Eclipse es un IDE que emplea módulos (plug-in) para toda la gama de funcionalidades que posee, a diferencia de otros entornos rígidos donde las mismas aparecen prefijadas, sean de utilidad para el usuario o no. Esto permite también el soporte de otros lenguajes de programación y la introducción de otras aplicaciones auxiliares que pueden resultar útiles durante el proceso de desarrollo como: herramientas UML, editores visuales de interfaces, ayuda en línea para librerías, además de la gran ventaja de ser una herramienta open-source y multiplataforma.

El IDE de desarrollo seleccionado para continuar la implemtación de la aplicación fue Eclipse debido a que posee un potente editor de código y una interfaz amigable para realizar el control de versiones subversión.

1.6 CONCLUSIONES

En el capítulo presentado se explica por qué es utilizado el proceso de desarrollo de software OpenUP para el desarrollo de la herramienta de software, porque la herramienta CASE que se utilizará es Visual Paradigm y UML el lenguaje de modelado del software, y que el lenguaje de programación Java fue seleccionado por ser multiplataforma entre otras. También que para implementar el código se

utilizará Eclipse como el entorno de desarrollo debido a su potente editor de códigos e interfaz amigable para el uso Subversion; que es el sistema de control de versiones escogido para controlar el producto en su desarrollo.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1. INTRODUCCIÓN

En este capítulo se define lo que debe hacer la aplicación a través de los requerimientos funcionales y no funcionales, se realiza el diagrama de caso de uso del sistema, así como las descripciones de los casos de uso del mismo, además se redefinen los casos de uso utilizados en la versión anterior para la continuidad de la aplicación.

2.2. ACTORES DEL SISTEMA

Los actores representan terceros fuera del sistema que interactúan con este. Acorde a la situación en la que se desarrollará la herramienta fue encontrado el siguiente actor:

Nombre del actor	Descripción
Investigador biológico	Es la persona que interactúa con el sistema. Elabora el modelo gráfico del sistema biológico y obtiene el correspondiente modelo matemático, realizando sus estudios particulares.

2.3. DEFINICIÓN DE LOS REQUISITOS FUNCIONALES

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. La aplicación a desarrollar a través de la realización del presente trabajo debe contar con los siguientes requisitos:

R1 - Generar el modelo matemático.
R2 - Guardar el modelo matemático del sistema biológico en formato MathML.
R3 - Guardar el modelo matemático del sistema biológico en formato SBML.

2.4 DEFINICIÓN DE LOS REQUISITOS NO FUNCIONALES

Los requisitos no funcionales son otros requisitos que no forman parte de la funcionalidad principal de la aplicación, como requisitos del entorno de desarrollo o ejecución (sistema operativo, servidores en los que correrá, lenguajes, etc.), restricciones que se aplicarán, prestaciones (tiempo de respuesta

mínimo, alta disponibilidad, etc.), fiabilidad, portabilidad, interfaces externas, seguridad y otros. Se puede decir que responden a aquellas cualidades y características que el producto debe tener para que sea atractivo, confiable, usable y seguro.

Dentro de los requisitos no funcionales para el desarrollo de la aplicación se encuentran:

Funcionalidad: A pesar de que el sistema podrá ser usado en un tiempo breve por aquellos usuarios que se encuentren capacitados para modelar Sistemas Biológicos, se debe realizar un adiestramiento previo con el objetivo de que puedan detectarse errores (en caso de que existan) y posibilitar la familiarización con la herramienta.

Apariencia o Interfaz externa: Se necesita una interfaz amigable, legible, interactiva, fácil de usar, profesional, clara y sencilla.

Usabilidad: El sistema podrá ser usado por aquellos usuarios que posean conocimientos básicos en el campo de la modelación de sistemas biológicos.

Soporte:

- **Mantenimiento**: El sistema debe estar bien documentado de forma tal que el tiempo de mantenimiento sea mínimo en caso de necesitarse.
- **Instalación**: La instalación del sistema debe caracterizarse por su facilidad, claridad y sencillez.
- **Portabilidad**: El sistema debe ser multiplataforma (ser capaz o caracterizarse por poder funcionar o mantener una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas).

Confiabilidad:

- **Tiempo medio de reparación:** La reparación del sistema en caso de surgir fallas en el mismo debe realizarse en el menor tiempo posible, poniendo todos los esfuerzos en función de que no supere las 72 horas.
- **Ayuda y Documentación:** El sistema constará con una ayuda en línea donde esté presente la documentación básica que posibilite comprender su funcionamiento y las funcionalidades generales a tener en cuenta para garantizar la utilización del mismo de manera eficiente.

Software:

- Para el uso del sistema es necesario tener instalado la máquina virtual de java 1,5 o una versión superior.

Hardware:

- Se debe contar con 256 MB de memoria RAM como mínimo, aunque lo ideal serian 512 MB.
- Procesadores Pentium IV.
- Disco duro de 20 GB como mínimo (depende de la cantidad de información a almacenar).

2.5 DIAGRAMA DE CASOS DE USO DEL SISTEMA

Cada forma en que los actores usan un sistema se representa con un caso de uso y éstos son "fragmentos" de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores. De manera más precisa un caso de uso especifica una secuencia de acciones que el sistema debe llevar a cabo, interactuando con sus actores dónde se obtiene un resultado de valor para los actores **[6]**.

Un diagrama de casos de uso del sistema contiene actores, casos de uso del sistema y las relaciones existentes entre los mismos. Ver **Fig 2.1**

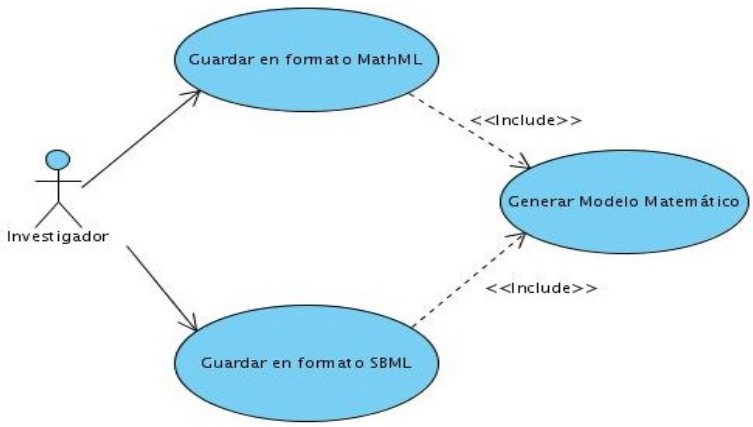


Figura 2.1 Diagrama de casos de uso del sistema

2.6 DESCRIPCIÓN DE CASOS DE USO DEL SISTEMA

R1-Generar el modelo matemático.

Nombre del Caso de Uso	Generar el modelo matemático
Actores	Investigador
Propósito	Permite al investigador generar el modelo matemático a partir del modelo gráfico.
Resumen	El caso de uso es iniciado por el investigador cuando desea guardar el modelo matemático a alguno de los formatos de la herramienta, luego de ser especificado la ruta a guardar el sistema genera el modelo matemático.
Referencias	R1
Precondiciones	R2 ó R3 deben ser ejecutados.
Poscondiciones	Es generado el modelo matemático.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**
1. El investigador solicita guardar el modelo matemático.	2. Muestra una ventana solicitando los datos necesarios para llevar a cabo el procedimiento.
3. Llena la solicitud en dependencia del formato a generar.	4. Genera el modelo matemático.

R2: Guardar el modelo matemático del sistema biológico en formato MathMl.

Nombre del Caso de Uso	Guardar en formato MathMl.
Actores	Investigador
Propósito	Permitirle al investigador guardar el modelo del sistema biológico en formato MathMl.
Resumen	El caso de uso es iniciado por el investigador cuando desea guardar el modelo matemático a partir del modelo gráfico; después de haber seleccionado la ruta donde será guardado, el sistema genera el modelo matemático en el formato MathMl.
Referencias	R2
Precondiciones	El modelo haya sido generado al formato MathML.
Pos condiciones	El modelo es guardado en formato MathML.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**
1. Solicita guardar el modelo matemático.	2. Muestra una ventana solicitando los datos necesarios para llevar a cabo el procedimiento de guardar en formato MathML.
3. Llena la solicitud introduciendo nombre y ruta donde desea guardar el modelo.	4. Verifica si el nombre tiene caracteres no válidos o si es posible guardar en la ruta introducida. 5. En caso de que alguno de los datos introducidos sea denegado el software vuelve a demandar al actor la entrada de datos válidos. 6. En caso de que los datos introducidos sean correctos genera el

	modelo matemático en formato MathML. 7. Guarda el modelo matemático generado en formato MathMl en la ruta especificada.

R3: Guardar el modelo matemático del sistema biológico en formato SBML.

Nombre del Caso de Uso	Guardar en formato SBML
Actores	Investigador
Propósito	Permitirle al investigador guardar la información del sistema biológico en el formato SBML.
Resumen	El caso de uso es iniciado por el investigador cuando desea guardar el modelo matemático a partir del modelo gráfico; después de haber seleccionado la ruta donde será guardado, el sistema genera el modelo matemático en el formato SBML
Referencias	R3
Precondiciones	El modelo haya sido generado al formato SBML.
Pos condiciones	El modelo es guardado en formato SBML.
Curso normal de los eventos	
Acción del actor	**Respuesta del Sistema**
1. Solicita guardar el modelo el modelo matemático.	2. Muestra una ventana solicitando los datos necesarios para llevar a cabo el procedimiento de guardar en formato SBML.
3. Llena la solicitud introduciendo la ruta donde desea guardar el modelo,	4. Verifica si los dos MetaID son diferentes o tiene caracteres no válidos o si es posible guardar en la ruta introducida además de si el dato correspondiente al campo versión fue introducido.

dos MetaID que tienen que ser diferentes y la versión, además de algunos datos opcionales como la notas y anotaciones.	5. En caso de que alguno de los datos introducidos sea denegado el software vuelve a demandar al actor la entrada de datos válidos. 6. En caso de que los datos introducidos sean correctos genera el modelo matemático en formato SBML. 7. Guarda el modelo matemático generado en formato SBML en la ruta especificada.

2.7 CONCLUSIONES

En el capítulo presentado se explica como se elaboraron artefactos requeridos por el OpenUp para el desarrollo del software, funcionalidades que debe realizar el software representada por los requisitos funcionales y las cualidades que él mismo debe tener en los requisitos no funcionales, actor que interactúa con el sistema y el diagrama con una descripción detallada de los casos de uso del sistema.

CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1. INTRODUCCIÓN

Durante el diseño se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos, funcionales y no funcionales que se le suponen. A través del presente capítulo se dará una descripción del estilo arquitectónico utilizado para el desarrollo de la herramienta, se describirán los patrones de diseño empleados así como los diagramas de clases del diseño y el modelo de despliegue.

3.2. ESTILO ARQUITECTÓNICO UTILIZADO

Para el desarrollo de la herramienta se utilizó el estilo arquitectónico Modelo Vista Controlador (Model-View-Controller - MVC), destinado para aplicaciones con sofisticadas interfaces donde la lógica de interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica de negocio, o sea, para el diseño de herramientas que necesitan la habilidad de mantener múltiples vistas para los mismo datos, tales como editores gráficos. **[8]**

Este estilo se basa principalmente en separar en tres capas el diseño de las aplicaciones, el modelo de datos, la presentación de los mismos y las acciones de los usuarios , utilizando la capa que gestiona las acciones (controlador) como administradora de los posibles eventos. Ver **Fig 3.1.**

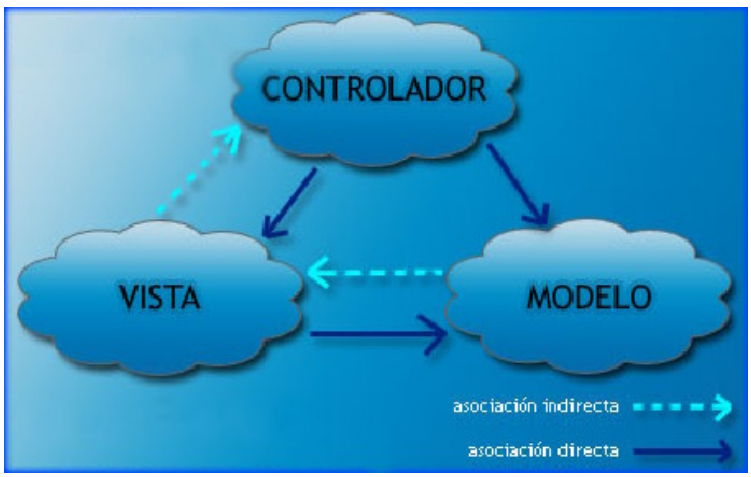


Figura 3.1

Su objetivo es realizar un diseño que desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.

Entre las principales ventajas que presenta este estilo arquitectónico se encuentra un mayor soporte para múltiples vistas, debido a que la vista se separa del modelo y no hay ninguna dependencia directa entre vista y modelo y la interfaz de usuario puede mostrar múltiples vistas de los mismos datos al mismo tiempo. Mayor soporte a los cambios: los requisitos de interfaz tienden a cambiar más rápidamente que las reglas de negocio. Puesto que el modelo no depende de las vistas, la adición de nuevos tipos de vistas al sistema generalmente no afectan al modelo. Por tanto, el ámbito del cambio se limita a la vista en tanto que le modelo permanece intacto. La vista correspondiente a este estilo arquitectónico se encuentra en el **Anexo 5.**

3.3. PRINCIPALES PATRONES DE DISEÑO UTILIZADOS

Los Patrones de Diseño (Design Patterns) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). En el presente epígrafe se describirán algunos de los patrones de diseño utilizados para el modelo de la herramienta así como los objetivos de su uso. **[9]**

3.3.1. PATRÓN GRASP: POLIMORFISMO

El polimorfismo tiene varios significados relacionados. En este contexto significa “asignar el mismo nombre a servicios en diferentes objetos” cuando los servicios son parecidos o están relacionados. Los diferentes tipos de objetos normalmente implementan una interfaz común o están relacionados en una jerarquía de implementación con una superficie común.

Cuando se identifican variaciones en un comportamiento, se le asigna la clase (interfaz) al comportamiento y se utiliza polimorfismo para implementar los comportamientos alternativos. Al implementar comportamientos alternativos con sentencias IF-ELSE, no se hace más que limitar la reutilización y crecimiento de la aplicación.

Si una aplicación muestra mensajes distintos en distintos idiomas, con IF, al aumentar en uno el número de idiomas, obligaría a añadir un nuevo IF, sin embargo al usar polimorfismo se limitaría a crear un objeto de una clase polimórfica nueva lo cual posibilita el bajo acoplamiento, la alta cohesión y un alto potencial de reutilización. Ver **Fig 3.2.**

Beneficios al usarlo:

- Se añaden fácilmente las extensiones necesarias para nuevas variaciones.
- Las nuevas implementaciones se pueden introducir sin afectar a los clientes.

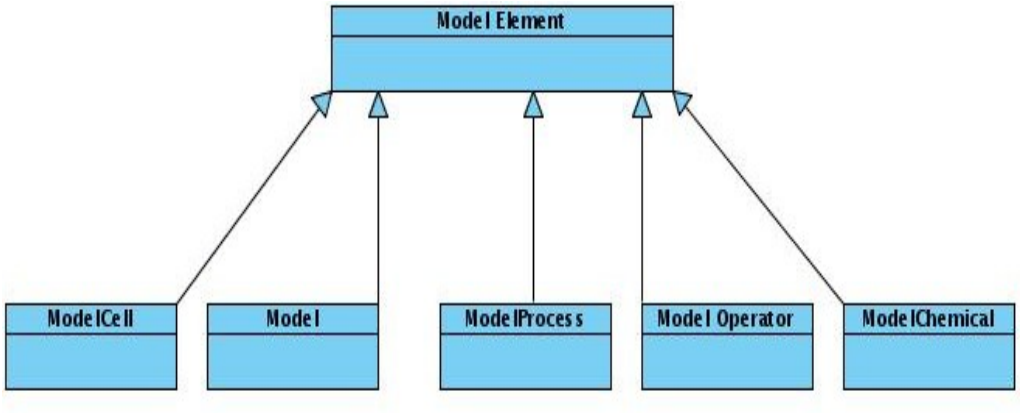


Figura 3.2 Polimorfismo

De esta forma se definen iguales servicios en la clase *ModelElement* que tendrán por el polimorfismo distintos funcionamientos en las clases hijas.

3.3.2. PATRÓN GRASP: EXPERTO

Lo que plantea este patrón es asignar una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con dicha responsabilidad. El problema que resuelve el experto está referido al principio más básico mediante el cual las tareas son asignadas en el diseño orientado a objetos.

El experto se emplea más que cualquier otro patrón en la asignación de responsabilidades, es un principio usado continuamente en el diseño orientado a objetos. Este patrón no significa una obscura

idea, sino que expresa la "intuición" común mediante el cual los objetos hacen cosas relacionadas con la información que ellos tienen, donde muchos expertos parciales colaboran en esa tarea.

El patrón se utiliza en diseños donde un objeto de software hace ciertas operaciones, que también son realizadas en el mundo real por el objeto que representa. Éste, como muchas cosas en la tecnología de objetos, tiene una analogía con el mundo real. Comúnmente se les asignan responsabilidades a individuos que tienen la información necesaria para cumplir con las tareas.

Beneficios al usarlo:

- La encapsulación es mantenida, desde que los objetos usan sus propias informaciones para realizar tareas. Esto permite poco acoplamiento, lo cual conduce a sistemas más robustos y de mantenimiento mucho más fácil.
- El comportamiento está distribuido a lo largo de clases que tienen la información requerida, así se alienta una clase "pesoligero" de definiciones que son fáciles de entender y mantener. La alta cohesión también es soportada.

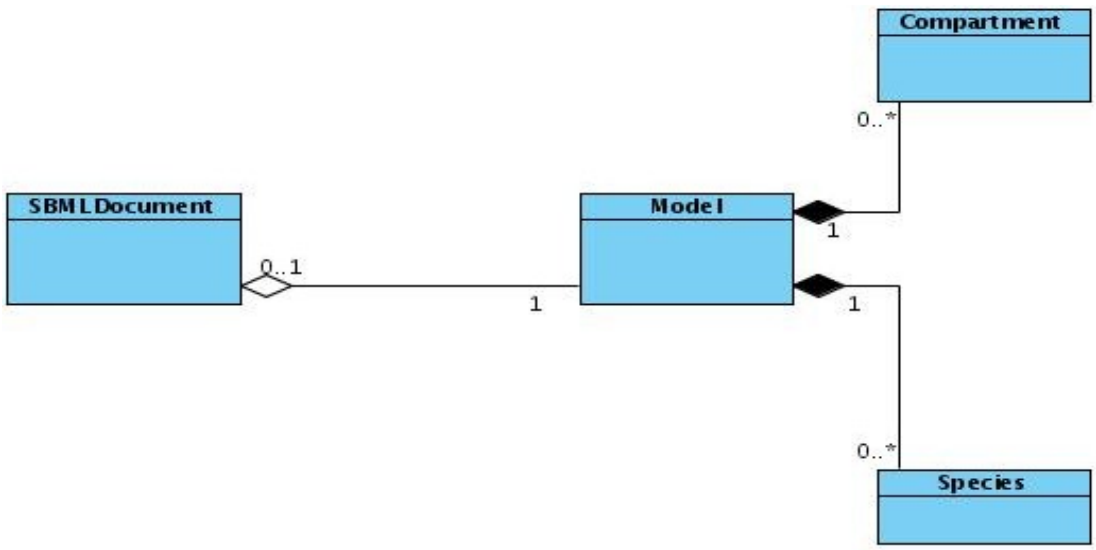


Figura 3.3 Experto

En la **Fig 3.3** se puede ver que *SBMLDocument* contiene una instancia de *Model*, a la que se le deja la responsabilidad de manipular las clases *Compartment* y *Species* en su debido momento al pedirle sus respectivos datos en un formato definido en XML, lo cual será incorporado a la respuesta de un servicio de *SBMLDocument*.

3.3.3. PATRÓN GRASP: CREADOR

Se asigna la responsabilidad de que una clase B cree un Objeto de la clase A solamente en caso de que B sea una agregación (o composición) de A contando que B tiene los datos de inicialización de A (datos que requiere su constructor). La creación de instancias es una de las actividades más comunes en un sistema orientado a objetos. En consecuencia es útil contar con un principio general para la asignación de las responsabilidades de creación. Si se asignan bien el diseño puede soportar un bajo acoplamiento, mayor claridad, encapsulación y reutilización.

El creador guía la asignación de responsabilidades relacionadas a la creación de objetos, una tarea muy común en sistemas orientados a objetos. El intento básico del patrón creador es encontrar un creador que necesite estar conectado al objeto creado en un evento en particular. El **Agregar** agrega partes, **Contener** contiene contenido, **Grabar** graba, todas estas son relaciones muy comunes entre clases en un diagrama de clases. El creador sugiere que el contenedor encerrado o grabado es un buen candidato por la responsabilidad de crear la cosa contenida o grabada.

Beneficios al usarlo:

- El bajo acoplamiento es soportado, lo cual implica bajo mantenimiento y altas oportunidades de reuso.

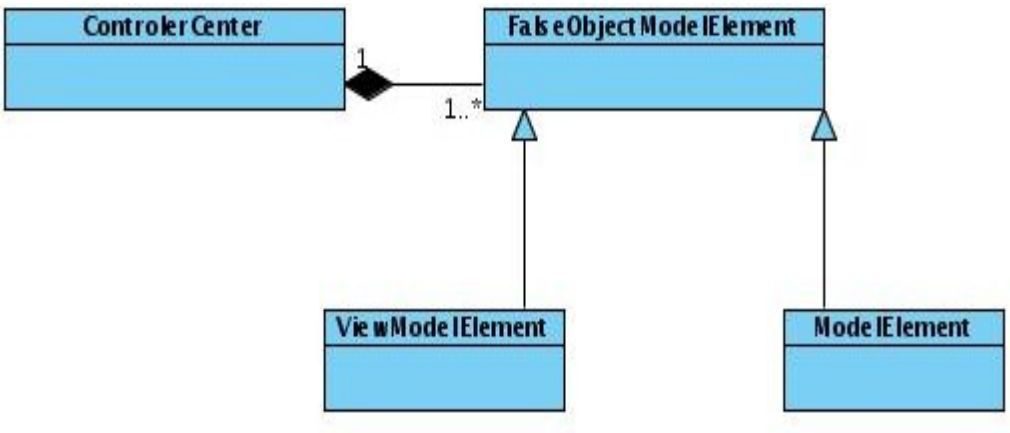


Figura 3.4 Creador

En la **Fig 3.4** se muestra un apoyo para entender como se utiliza este patrón en la herramienta. La clase ControlCenter es la encargada de crear todas las instancias de FalseObjectModelElement. ¿Cómo lo logra? Desde el modelo gráfico el usuario está todo el tiempo haciendo cambios, esta clase ControlCenter se está encargando por debajo del entorno gráfico de trabajo del usuario tanto de actualizar el *ViewModelElement* como de crear la instancia *ModelElement* correspondiente a partir de los datos del *ViewElement* modificado o creado.

3.3.4 PATRÓN GRASP: ALTA COHESIÓN

Lo que sugiere este patrón es que cada elemento del diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable. El uso del mismo se evidencia en la figura 3.3 donde se observa que la clase *Model* es responsable de obtener y conformar la información de las clases *Compartment* y *Species* para posteriormente incorporar sus respectivos datos, en formato SBML y MathMl, al fichero.

3.4 DIAGRAMA DE CLASES DEL DISEÑO

Un diagrama de clases de diseño muestra las especificaciones para las clases software de una aplicación. Incluye la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces con sus operaciones y constantes.
- Métodos.
- Navegabilidad.
- Dependencias.

Este tipo de diagramas muestra definiciones de entidades software más que conceptos del mundo real y se elabora para tener en cuenta los detalles concretos de la implementación del sistema. En los **Anexos 1, 2, 3 y 4** se encuentran los diagramas de clases del diseño de las tres capas, modelo, vista y controlador respectivamente; el **Anexo 6** registra la descripción de los mismos de forma detallada.

3.5 ASPECTO DEL DISEÑO A DESTACAR

Existen varios aspectos que debe caracterizar el diseño de un software:

- Correcto: Conduce a una puesta en práctica satisfactoria.
- Adaptable: Facilita el cambio (el cambio es inevitable).
- Eficiente: Gastar solo los recursos necesarios.
- Simple: Ser tan comprensible como sea posible.

El diseño realizado de la herramienta cumple con las características anteriormente citadas, pero se hará referencia a continuación a su facilidad de cambio por la importancia que para el presente trabajo tiene.

La modelación gráfica ha sido una de las principales vías empleada por el hombre a lo largo de los años para comprender el funcionamiento de sistemas. En la actualidad el hombre modela un conjunto de procesos que son de su interés, procesos químicos, biológicos, industriales, sociales, etc. Por tal razón se desarrolló el diseño de la herramienta enfocándolo en convertirla en un framework para el desarrollo de aplicaciones capaces de modelar gráficamente procesos.

El objetivo anterior fue cumplido debido al uso adecuado de los patrones de diseño descritos en la anterior versión del producto, en la que se hace uso de patrones que se enfocaron a estos fines, aportando así un elevado grado de extensibilidad a la aplicación. A lo largo de su desarrollo se han agrupado los patrones siguientes:

- Patrón Delegation
- Patrón Interfaz
- Patrón Composite
- Patrón Abstract Factory
- Patrón Facade
- Patrón Grasp: Experto
- Patrón Grasp: Creador

- Patrón Grasp: Polimorfismo
- Patrón Grasp: Alta Cohesión

3.4 CONCLUSIONES

En el capítulo presentado fue descrito el estilo arquitectónico que se utilizó en el diseño de la herramienta así como los patrones de diseño utilizados y la fundamentación de su uso. También fueron desarrollados los diagramas correspondientes a las clases del diseño.

CAPÍTULO 4: IMPLEMENTACIÓN DEL SISTEMA

4.1. INTRODUCCIÓN

En el presente capítulo se describe como fue implementada la herramienta haciendo uso de componentes, para esto se muestra el diagrama de componentes. Se ha desarrollado varias pruebas para conocer la aceptación del producto, utilizando los métodos de caja negra en cada caso para asegurar que los requisitos funcionales demandados fueron cumplidos, además de asegurar el correcto funcionamiento de la aplicación.

4.2. IMPLEMENTACIÓN

El flujo de trabajo de implementación comienza a partir de los resultados obtenidos del diseño donde se propone crear un plano del modelo de implementación. Además se describen como los elementos del modelos de diseño se implementan en términos de componentes y como se organizan de acuerdo a los nodos específicos en el modelo de despliegue **[6]**.

A continuación de manera detallada es explicado el código perteneciente a las clases de los diagramas de diseño expuestos en los **Anexos 1, 2, 3, 4;** y descritas minuciosamente en el **Anexo 7.**

4.2.1 DESCRIPCIÓN DEL CÓDIGO EN CLASES CONTROLADORAS

Dentro de la clase controladora *FalseObjectList* que es la que guarda los *FalseObjectModelElement*, los cuales contienen los datos del objeto visual y del modelo **–** se ha creído oportuno citar y explicar brevemente algunos de los métodos principales que introducen las ecuaciones respectivas a los objetos del modelo.

A continuación esta plasmado el método *ObtainModelElementsOfProcessInTarget.* Este método obtiene todos los conectores, ya sean inhibiciones, catálisis, señales simples, degradaciones, diferenciaciones que tienen su objetivo determinado en determinado componente gráfico, aunque no se se exceptúa que este componente pueda ser a su vez un conector, pues como parámetro que se le pasa externamente tenemos una lista de puertos, que son aquellos que el componente gráfico ha notificado que tiene determinada entrada de un conector.

```
private ArrayList<FalseObjectModelElement>
ObtainModelElementsOfProcessInTarget(ArrayList<ViewPortBorder> portsAnd)

        {

                ArrayList<FalseObjectModelElement> retornar=new
ArrayList<FalseObjectModelElement>();


                        for(int i=1;i<size();i++)

                                if(get(i) instanceof FalseObjectComplexProcessModelElement)

                                {

                                        for (int h = 0; h < portsAnd.size(); h++)


        if(((ViewComplexConnector)((FalseObjectComplexProcessModelElement)get(i)).ObtainViewEle
ment()).AnyWayIsPoint(portsAnd.get(h).getConnection()))

                                                        retornar.add(((FalseObjectModelElement)get(i)));

                                }

                        return retornar;

        }
```

Este otro método llamado *ExtractModelElementFromFalseObject* extrae el objeto de modelo de los *FalseObjectModelElement* que le entran por parámetro, para que de forma eficiente se pueda trabajar y manejar las instancias en los métodos más complejos.

```
public ArrayList<ModelElement>
ExtractModelElementFromFalseObject(ArrayList<FalseObjectModelElement> listing)

        {

                ArrayList<ModelElement> retornar=new ArrayList<ModelElement>();
```

```
        for(int i=0;i<listing.size();i++)
            retornar.add(((FalseObjectModelElement)listing.get(i)).ObtainModelElement());
        return retornar;

    }
```

Este método llamado *ExtractViewModelElementFromFalseObject,* a manera del anterior extrae el objeto visual de los *FalseObjectModelElement* que le entran por parámetro, para que de forma eficiente se pueda trabajar y manejar las instancias en los métodos más complejos.

```
    public ArrayList<ViewModelElement>
ExtractViewModelElementFromFalseObject(ArrayList<FalseObjectModelElement> listing)
    {
        ArrayList<ViewModelElement> retornar=new ArrayList<ViewModelElement>();
        for(int i=1;i<listing.size();i++)
            retornar.add(((FalseObjectModelElement)listing.get(i)).ObtainViewElement());
        return retornar;
    }
```

Este método llamado *ObtainFromPort1* busca y retorna el *FalseObjectModelComponent* que contiene el puerto visual que es entrado por parámetro, de esta forma se localiza en medio de la gran gama de componentes gráficos aquel componente que tiene determinada locación dado un puerto visual.

```
    public FalseObjectModelComponent ObtainFromPort1(ViewPortBorder port)
    {
        for (int i = 0; i < size(); i++)
        {
            if(get(i) instanceof FalseObjectModelComponent)
```

```
                {

        if(((ViewSimpleComponent)((FalseObjectModelComponent)get(i)).ObtainViewElement()).KnowIf
PortIsBusy(port.getConnection()));
                        return (FalseObjectModelComponent)get(i);
                }
        }
        return null;
}
```

Este método busca por *FalseObjectList* y extrae las ecuaciones correspondientes a los FalseObjectModelElement cuyos ids son pasados por parámetro. De esta forma con tan solo manejar los ids, se obtienen inmediatamente las ecuaciones de los elementos que realmente existen y han sido tratados previamente con el fin de utilizar sus ecuaciones para modificarlas o ser utilizadas para conformar las ecuaciones de otros componentes gráficos.

```
public ArrayList<Equacion> ObtainEquationOf(ArrayList<String> ids)
{
        ArrayList<Equacion> equations=new ArrayList<Equacion>();
        for (int i = 0; i < size(); i++)
                for (int j = 0; j < ids_connectors.size(); j++)

if(((FalseObjectModelElement)get(i)).ObtainViewElement().getId()==ids_connectors.get(j))
                        {
                                if(((FalseObjectModelElement)get(i)).ObtainModelElement()!=null)
```

```
if(((FalseObjectModelElement)get(i)).ObtainModelElement().getEquation()!=null)

equations.add(((FalseObjectModelElement)get(i)).ObtainModelElement().getEquation());
                        }
        return equations;

}
```

El método *IfConectorGoToComponent* recibe por parámetro un conector visual y un componente visual y retorna verdadero si este conector está de alguna manera relacionado con dicho componente, ya sea porque sea su objetivo o fuente. Retorna falso en caso de que no exista ninguna relación y se descarta la posibilidad de que el componente esté afectado por dicho conector, lo cual modificaría su ecuación inevitablemente.

```
public boolean IfConectorGoToComponent(ViewComplexConnector
conector,ViewSimpleComponent component)
{   ArrayList<ViewPortBorder> ports=component.getListPort();
        for (int i = 0; i < ports.size(); i++)
        if(conector.AnyWayIsPoint(ports.get(i).getConnection()))
                return true;
        return false;   }
```

El método *ActualizeWithEquationOR* recibe por parámetro una determinada ecuación e id y entonces busca por *FalseObjectList* el FalseObjectModelElement y a la ecuación de su instancia de objeto del modelo le agrega un término que sería dicha ecuación entrada por parámetro.

```
public void ActualizeWithEquationOR(String id,Equacion eq)
{
```

```
            for (int i = 0; i < size(); i++)
            {
                    if(((FalseObjectModelElement)get(i)).ObtainViewElement().getId()==id)

((FalseObjectModelElement)get(i)).ObtainModelElement().AddToEquationAs_("OR", eq);
            }
}
```

Este método *ActualizeList* es el más importante de la clase FalseObjectList pues con tan solo su llamada se empiezan a ejecutar métodos internos de la clase que van a ir conformando las ecuaciones en cada uno de los *FalseObjectModelElement* que se encuentran dentro de ella misma. La llamada a 6 procedimientos como se muestra en el cuerpo del método irá creando o actualizando las ecuaciones en cada una de las instancias.El código correspondiente al método *ActualizeList* se encuentra posterior a la tabla de descripción

Para una mejor exposición se explica en la tabla siguiente:

Métodos del cuerpo del método de *ctualizeList()*	Descripción
CreateModelElementInFalseObjectModelComponent()	Crea la ecuación del *FalseObjectModelElement* si éste es de tipo *FalseObjectModelComponent*. Que son principalmente los componentes simples tales como ands, ors, canales iónicos, degradaciones, receptores, canales de Salida.
CreateModelElementInFalseObjectComplexProcessModelElement()	Crea la ecuación del *FalseObjectModelElement* si éste es de tipo *FalseObjectComplexProcessModelElement.* Que son principalmente los conectores tales

	como señales simples, catálisis, inhibiciones, *diferenciaciones, transiciones, transportes.*
CreateModelElementInFalseObjectModelExpandComplexComponent()	Crea la ecuación del *FalseObjectModelElement* si éste es de tipo *FalseObjectModelExpandComplexComponent.* Que son principalmente los componentes complejos tales como los modelos y células.
ActualizeModelElementInFalseObjectModelComponent()	Actualiza aquellas ecuaciones de los componentes simples que creó previamente con el método *CreateModelElementInFalseObjectModelComponent()* dada la posible afectación de algún conector.
ActualizeModelElementInFalseObjectComplexProcessModelElement()	Actualiza aquellas ecuaciones de los conectores que creó previamente con el método *CreateModelElementInFalseObjectComplexProcessModelElement()* dada la posible afectación de algún otro conector a él.
ActualizeModelElementInFalseObjectModelExpandComplexComponent()	Actualiza aquellas ecuaciones de los componentes complejos que creó previamente con el método *CreateModelElementInFalseObjectModelExpandComplexComponent()* dada la posible afectación interna por la presencia de conectores o componentes simples o otros tipos de afectaciones como la de conectores diferenciación de otros componentes complejos que interactúan con él.

```
public void ActualizeList()
```

```
{
        CreateModelElementInFalseObjectModelComponent();
        CreateModelElementInFalseObjectComplexProcessModelElement();
        CreateModelElementInFalseObjectModelExpandComplexComponent();
        ActualizeModelElementInFalseObjectModelComponent();
        ActualizeModelElementInFalseObjectComplexProcessModelElement();
        ActualizeModelElementInFalseObjectModelExpandComplexComponent();
}
```

4.2.2 DESCRIPCIÓN DEL CÓDIGO EN CLASES DEL MODELO

En esta sección se explica las funcionalidades principales de la clase *ModelElement*, clase entidad que es la que guarda la información matemática de los componentes visuales del modelo. De esta clase se halla una instancia dentro de la clase entidad *FalseObjectModelElement* que contiene también la instancia visual. Esta clase es abstracta, heredando de ella las siguientes clases hijas:

Clases Hijas de *ModelElement*	Descripción
ModelCell	Es la que representa a las células.
Model	Es la que representa a los modelos.
ModelOperator	Es la que representa los operadores and y or.
ModelChemical	Es la que representa los químicos tales como citoquinas, receptores, degradaciones, canales de salida, canales iónicos.
ModelProcess	Es la que representa los conectores tales como catálisis, inhibiciones, señales simples, transportes, transición, diferenciaciones.

Esta clase contiene en sus atributos la ecuación y el id correspondiente al componente o conector. Esta ecuación puede ser modificada a través del método *AddToEquationAs,* el cual recibe por parámetro la forma en que se va a insertar la equación – en multiplicación o suma – entrado por parámetro.

```
public void AddToEquationAs_(String type/*type= AND || OR*/,Equacion n)

    {   if(equation.getTerminos().size()!=0)

            {       Sumatorial old=new Sumatorial();

                    for (int i = 0; i < equation.getTerminos().size(); i++)

                    {       old.AgregarEcuacion(new Equacion(equation.getTerminos().get(i)));

                    }

                    Termino actual;

                    if(type=="OR")

                    {       old.AgregarEcuacion(n);

                            actual=old;

                    }

                    else

                            {       actual =new Producto();

                                    actual.AgregarEcuacion(new Equacion(old));

                                    actual.AgregarEcuacion(n);

                            }

                    equation=new Equacion(actual);

            }

            else
```

```
        {       Producto alone=new Producto();
                alone.AgregarEcuacion(n);
                equation.AgregarTermino(alone);
        }
    }
```

El método *SaveToMathMl* retorna una instancia *Element* la cual contiene la ecuación personal en formato MathMl.

```
    public Element SaveToMathMl(String formatMath)
    {
        Fraccion f=new Fraccion(new Equacion(new Independiente("d"+getId())),new
        Equacion(new Independiente("dt")));
        Element eq=new Element("mo");
        eq.addContent("=");
        Element element;
        Element previo;
        if(formatMath=="presentation")
        {
            previo=MathFormat.GetStandardFormatToMathMl();
            previo.addContent(f.GetMathMlPresentation());
            previo.addContent(eq);
            element=equation.GetMathMlPresentation(previo);
        }
```

```
    Else
{               previo=MathFormat.GetStandardFormatToMathMl();
                previo.addContent(f.GetMathMlContent(new Element("apply")));
                previo.addContent(eq);
                element = equation.GetMathMlContent(previo);
                }
        return element;
}
```

Lo que sucedió anteriormente en el método *SaveToMathMl* pudo realizarse dado el funcionamiento de un método dentro de la clase Equacion a través del método *GetMathMlPresentation*, el cual le pide a cada término su respectiva representación en MathML como se muestra:

```
public Element GetMathMlPresentation(Element mathMl)
{       for (int i = 0; i < terminos.size(); i++)
        {   mathMl.addContent(terminos.get(i).GetMathMlPresentation());
            if(i<terminos.size()-1)
            {   Element op=new Element("mo");
                op.addContent("+");
                mathMl.addContent(op);
            }
        }
        return mathMl;
}
```

La clase *Equacion* como se supone contiene una lista de instancias de la clase *Termino*, la cual aparece en el cuerpo del método como termino. La clase *Termino* no es definitiva, es abstracta y de ella se desprenden las siguientes clases hijas con sus códigos respectivos para la implementación del método abstracto *GetMathMlPresentation*:

Independiente extends Termino

```
public Element GetMathMlPresentation()
        {    Element mrow;
                if(variable!="-1")
                        mrow=new Element("mi");
                else mrow=new Element("mn");
                mrow.addContent(variable);
                return mrow;
        }
```

Exponencial extends Termino

```
public Element GetMathMlPresentation()
            {
                  Element msup=new Element("msup");
                Element mrowb=new Element("mrow");
                Element mrowe=new Element("mrow");
                Element b=base.GetMathMlPresentation(mrowb);
                Element e=exponente.GetMathMlPresentation(mrowe);
```

```
            msup.addContent(b);
            msup.addContent(e);
            return msup;
        }
```

Fraccion extends Termino

```
public Element GetMathMlPresentation()
    {
            Element mfrac=new Element("mfrac");
            Element mrownum=new Element("mrow");
            Element mrowden=new Element("mrow");
            Element n=numerador.GetMathMlPresentation(mrownum);
            Element d=denominador.GetMathMlPresentation(mrowden);
            mfrac.addContent(n);
            mfrac.addContent(d);
            return mfrac;
    }
```

Sumatorial extends Termino

```
public Element GetMathMlPresentation()
    {       Element producto=new Element("mrow");
            Element final_producto;
            for (int i = 0; i < integrants.size(); i++)
            {     final_producto=integrants.get(i).GetMathMlPresentation(producto);

                    if(i<integrants.size()-1)
```

```
                {    Element op=new Element("mo");
                        op.addContent("+");
                        final_producto.addContent(op);
                }
                producto=final_producto;
            }
            return producto;
        }
```

Producto extends Termino

```
public Element GetMathMlPresentation()
            {
                Element producto=new Element("mrow");
                Element final_producto;
                Element parentesis_i=new Element("mo");
                parentesis_i.addContent("(");
                if(integrants.size()>1)
                        producto.addContent(parentesis_i);
                Element parentesis_f=new Element("mo");
                parentesis_f.addContent(")");
                for (int i = 0; i < integrants.size(); i++)
                {    final_producto=integrants.get(i).GetMathMlPresentation(producto);
```

```
            if(i<integrants.size()-1)
            {     Element op=new Element("mo");
                  op.addContent("*");
                  final_producto.addContent(op);
            }
            producto=final_producto;
      }
      if(integrants.size()>1)
            producto.addContent(parentesis_f);
      return producto;    }              }
```

4.2.3 DESCRIPCION DEL CÓDIGO EN CLASES INTERFACES

Dentro de las clases interfaces es necesario explicar la clase SBML_Editor, la cual es la más relevante. Dentro de ella se escogió detallar en su responsabilidad y funcionamiento el constructor y los métodos principales. Esta clase es la encargada para interactuar con el usuario a fin de que se pueda generar correctamente el documento SBML que contendrá la información de los FalseObjectModelElement.

El constructor del SBML_Editor creará cuando sea invocada una instancia de la clase SBML_Editor inicializando todos sus componentes y funcionalidades visuales. También recibe por parámetro para crearse una FalseObjectList, ya que éste trabajará eventualmente que sea creado con los FalseObjectModelElement que se han venido conformando a partir de la modelación gráfica. Lo primero que hace con esta lista es extraer de ella la información concerniente al formato SBML del futuro documento que salvará en el fichero respectivo.

Toda esta gestión se hace separándose en dos listas principales; la primera de ellas es para guardar los compartimentos que son las células y modelos, la segunda es de especies donde se recoge todo aquello que sea conector o componente simple.

```
public SBML_Editor(FalseObjectList list)
        { initialize();
                this.list=list;
                if(list!=null)
                {    for (int i = 1; i < list.size(); i++)
                        {    f(list.get(i).ObtainViewElement().getType()==elementType.Model.toString()
list.get(i).ObtainViewElement().getType()==elementType.Modelo.toString()
list.get(i).ObtainViewElement().getType()==elementType.Celula.toString())
                                {   compartment.add(list.get(i));
                                }
                                else species.add(list.get(i));
                        }
                }
        }
```

El método siguiente es el de mayor relevancia dentro de la clase interfaz SBML_Editor. El método va actualizando el SBML_Document, el cual se encuentra en la instancia document. Y luego retorna la ejecución del método SaveToSBML de SBML_Document la cual devuelve una instancia de Element, que es un objeto que contiene una información XML, lleno de etiquetas, las cuales han sido conformadas en equivalencia al formato SBML.

```
        public Element SaveALLToSBML()
        {
```

```
                ////actualizar datos singulares del documento
                 document.setLevel("2");
                 document.setMetaid(textField_4.getText());
                 document.setVersion(textField_2.getText());
                 document.setXmlns(textField_3.getText());

                 ////crear el modelo
                 document.getModel().setId("id_"+list.get(0).getId());
                 document.getModel().setMetaid(textField_5.getText());
                 document.getModel().setNotes(new Notes(textArea.getText()));
                 document.getModel().setName(list.get(0).getType());
                 ////actualizar datos del modelo
                 for (int i = 0; i < compartment.size(); i++)
                 {     double x=compartment.get(i).ObtainViewElement().getMaxX();
                          double y=compartment.get(i).ObtainViewElement().getMaxY();
                          double size=x*y;
              document.getModel().getListOfCompartments().add(new
Compartment("id_"+compartment.get(i).getId(),"mi_"+compartment.get(i).getId(),size+""));      }

                 for (int i = 0; i < species.size(); i++)
                 {
                          document.getModel().getListOfSpecies().add(new
Species("id_"+species.get(i).getId(),"mi_"+species.get(i).getId(),ObtainTypeOfCompartmentThatContai
```

```
nThis(species.get(i).getId()),species.get(i).getType(),new
necessary.Annotation(),species.get(i).getId(),"initialConcentration",true));

                }

                for (int i = 1; i < list.size(); i++)

                {     if(list.get(i).ObtainModelElement()!=null)

                        document.getModel().getListOfFuntiontDefinitions().add(new
FunctionDefinition(list.get(i).getId(),"metaid_"+list.get(i).getId(),list.get(i).getType(),new
MathFormat(list.get(i).ObtainModelElement().getEquation())));

                }

                return document.SaveToSBML("presentation");

        }
```

El siguiente método retorna todos aquellos componentes complejos que están dentro de FalseObjectList. Los cuales son los futuros compartimentos del documento SBML que se quiere guardar en el fichero que es generado por SBML_Editor.

```
        private String ObtainTypeOfCompartmentThatContainThis(String id)

        {    String type=list.get(0).getType();

                for (int j = 1; j < list.size(); j++)

                {      if(list.get(j).ObtainViewElement() instanceof ViewComplexComponent)

                        {

                                ArrayList<ViewSimpleComponent>

listSimplexComponent=((ViewComplexComponent)list.get(j).ObtainViewElement()).getListSimplexCom
ponent();

ArrayList<ViewConnector>
```

```
listConnector=((ViewComplexComponent)list.get(j).ObtainViewElement()).getListConnector();

                        for (int i = 0; i < listSimplexComponent.size(); i++)
                        {
                                if(listSimplexComponent.get(i).getId()==id)
                                        return list.get(j).ObtainViewElement().getId();
                        }
                        for (int i = 0; i < listConnector.size(); i++)
                        {
                                if(listConnector.get(i).getId()==id)
                                        return list.get(j).ObtainViewElement().getId();
                        }

                }
        }
        return type;
    }
```

4.3. DIAGRAMA DE COMPONENTES

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

Los elementos de modelado dentro de un diagrama de componentes serán componentes y paquetes. La **fig 4.1** representa el diagrama de componentes perteneciente a BioSyS 1.0.

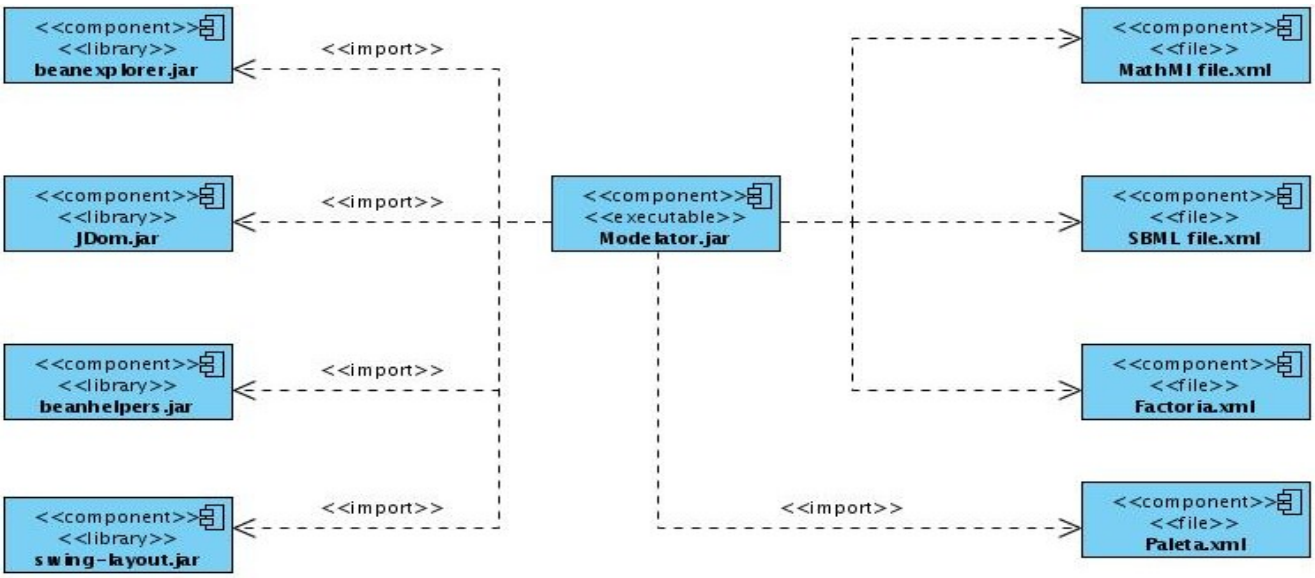


Figura 4.1

4.4. MODELO DE PRUEBA

Como la aplicación desarrollada es un módulo de una plataforma computacional, el tipo de prueba realizado es el de Prueba de Unidades. El objetivo de realizar las pruebas a la aplicación es comprobar que esta cumpla correctamente los requisitos funcionales establecidos ya que se requiere gran esfuerzo mental por parte de los desarrolladores, por lo que la posibilidad de cometer errores es muy alta. Se utilizó el método de caja negra, con lo cual se verificó no sólo que las funciones del software son operativas, sino también que no existieran errores en las interfaces.

Aun así las pruebas no confirman la ausencia de errores en el presente software, solo brindan una medida de cómo responderá el mismo ante algunas situaciones determinadas.

A continuación se muestran los casos de prueba desarrollados para los casos de uso del sistema, viéndose y probándose a través de los métodos de prueba señalados previamente. El caso de uso *Generar Modelo Matemático* es comprobado en los otros dos casos de uso; *Guardar en Formato MathMl* y *Guardar en Formato SBML.*

4.4.1 CASO DE USO: Guardar en formato MathMl

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de Célula
Entrada	Se incorpora al Modelo_BioSys un componente Célula. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar. **(Fig 4.2)**
Resultado Esperado	Se agrega una instancia del componente Célula a Modelo_BioSyS. Y se obtiene la ecuación generada para modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente el componente se obtuvo satisfactoriamente la ecuación diferencial esperada y correspondiente al componente en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 1)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

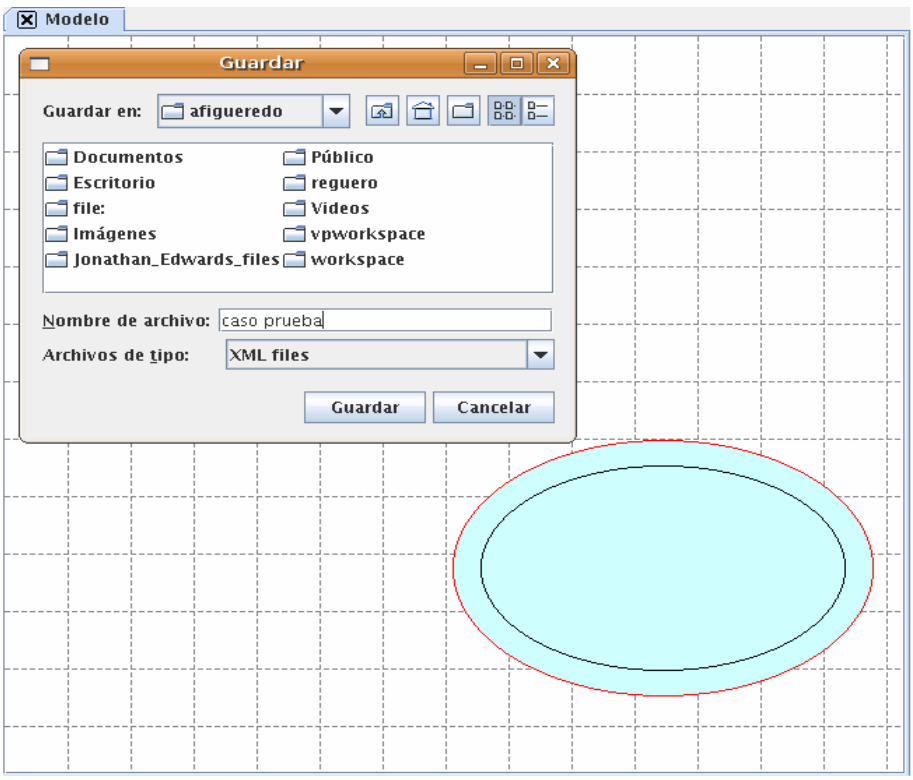


Figura 4.2

Resultado 1

$$\frac{dCelula}{dt} = (KbCelula * ACelula)$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de dos citoquinas interactuando en un AND a un nivel de abstracción extracelular.
Entrada	Se agregan dos instancias del componente Citoquina al Modelo_BioSyS. Se relaciona a través de una relación AND incorporada. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar. **(Fig 4.3)**
Resultado Esperado	Se agrega una instancia del componente Célula a Modelo_BioSyS. Y se obtiene la ecuación generada para modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente los componentes con la debida interacción – AND - se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 2)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

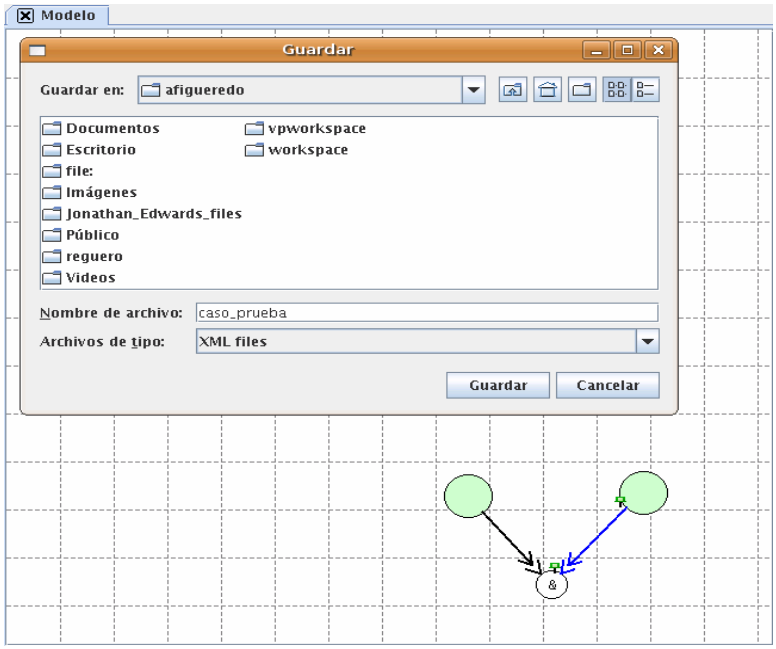


Figura 4.3

Resultado 2

$$\frac{dCitoquina}{dt} = Citoquina + (-1 * (BCitoquina * Citoquina)) + (-1 * \frac{(BCitoquina * Citoquina)^h}{KSeñal + (BCitoquina * Citoquina)^h})$$

$$\frac{dCitoquina1}{dt} = Citoquina1 + (-1 * (BCitoquina1 * Citoquina1)) + (-1 * \frac{(BCitoquina1 * Citoquina1)^h}{KSeñal1 + (BCitoquina1 * Citoquina1)^h})$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de dos citoquinas interactuando en un OR a un nivel de abstracción intracelular.
Entrada	Se agregan un componente Célula y luego dentro de él se incorporan dos componentes Citoquina en dicha relación de OR. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar. **(Fig 4.4)**
Resultado Esperado	Se conforma correctamente el modelo gráfico y se obtiene la ecuación generada para el modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente los componentes con la debida interacción **–** OR - se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 3)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

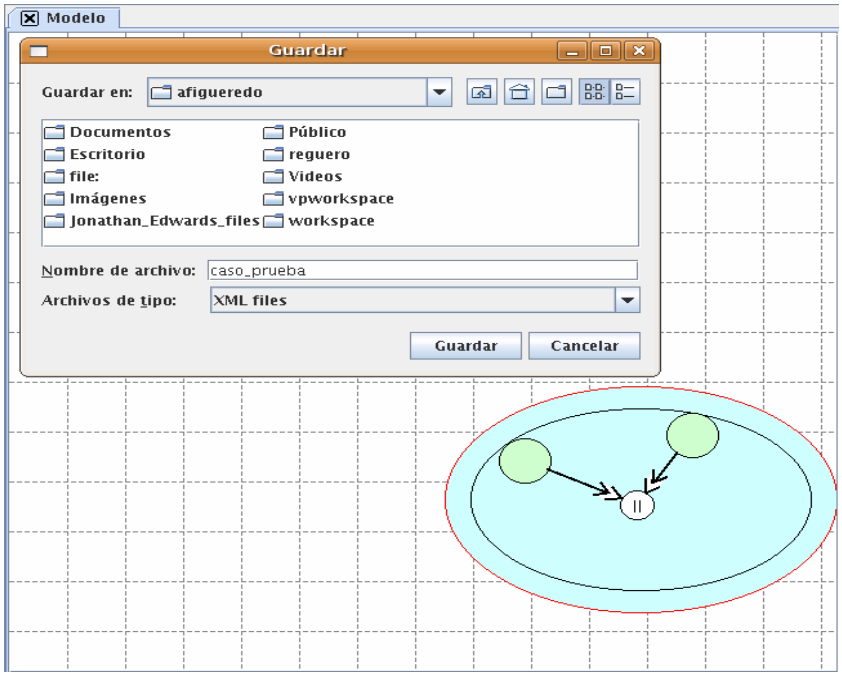


Figura 4.4

Resultado 3:

$$\frac{dCelula}{dt} = (((KbCelula * ACelula) * \frac{(BCitoquina * Citoquina)^h}{KSeñal + (BCitoquina * Citoquina)^h}) * \frac{(BCitoquina1 * Citoquina1)^h}{KSeñal1 + (BCitoquina1 * Citoquina1)^h})$$

$$\frac{dCitoquina}{dt} = Citoquina + (-1 * (BCitoquina * Citoquina)) + (-1 * \frac{(BCitoquina * Citoquina)^h}{KSeñal + (BCitoquina * Citoquina)^h})$$

$$\frac{dCitoquina1}{dt} = Citoquina1 + (-1 * (BCitoquina1 * Citoquina1)) + (-1 * \frac{(BCitoquina1 * Citoquina1)^h}{KSeñal1 + (BCitoquina1 * Citoquina1)^h})$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de la Célula que contiene una Señal Simple desde un receptor a un canal de salida.
Entrada	Se agregan un componente Célula, se incorpora igualmente un componente Receptor y otro CanalDeSalida y luego se relacionan a través de una SeñalSimple. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar. **(Fig 4.5)**

Resultado Esperado	Se conforma correctamente el modelo gráfico y se obtiene la ecuación generada para el modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente los componentes se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción – Célula, Receptor y CanalDeSalida - en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. Ver**(Resultado 4)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

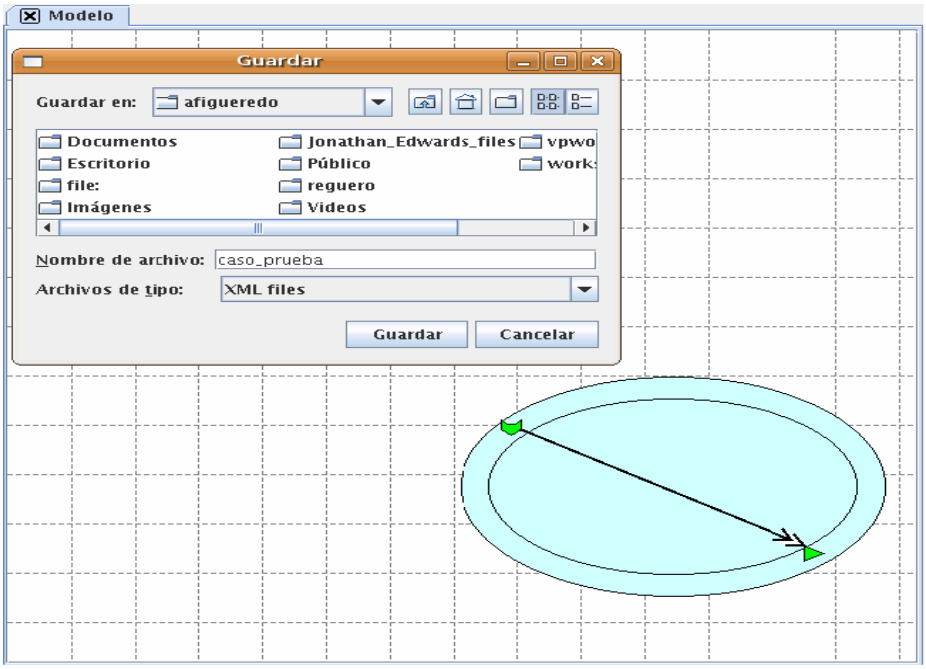


Figura 4.5

Resultado 4:

$$\frac{dCelula}{dt} = ((KbCelula * ACelula) * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dReceptor}{dt} = Receptor + (-1 * (BReceptor * Receptor)) + (-1 * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dCanalDeSalida}{dt} = CanalDeSalida$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de la Célula que contiene una Señal Simple desde un receptor a un canal de salida, la cual está afectada por una Inhibición.
Entrada	Se agregan un componente Célula, se incorpora igualmente un componente Receptor y otro CanalDeSalida y luego se relacionan a través de una SeñalSimple. También se dirige hacia esta una Inhibición proveniente de un receptor de la Célula – incorporado previamente. **(Fig 4.6)** Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar.
Resultado Esperado	Se conforma correctamente el modelo gráfico y se obtiene la ecuación generada para el modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente los componentes se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción – Célula, Receptor, Receptor1 y CanalDeSalida - en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 5)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

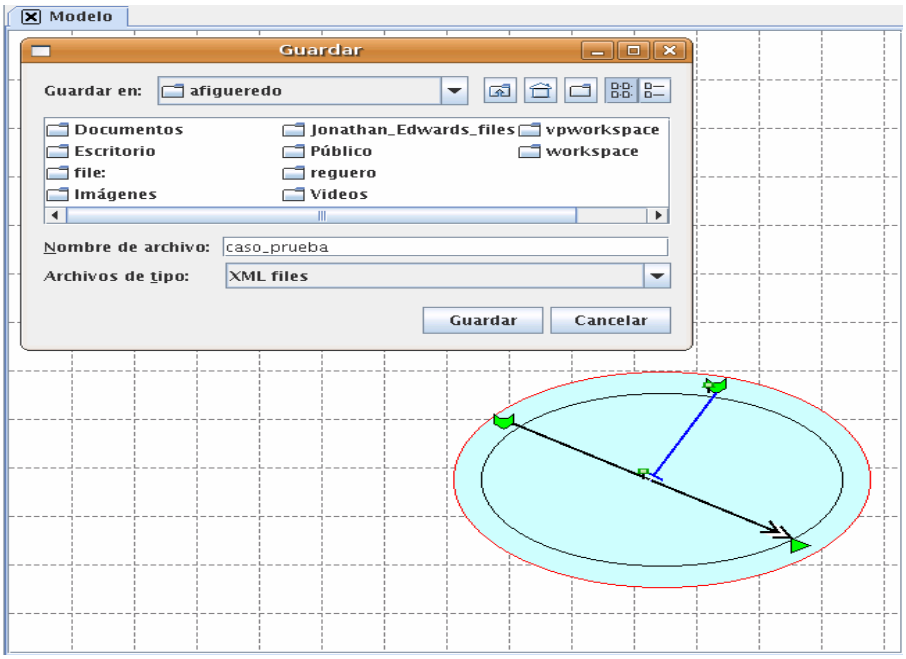


Figura 4.6

Resultado 5:

$$\frac{dCelula}{dt} = (((KbCelula * ACelula) * (\frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h} * \frac{KInhibicion}{KInhibicion + (BReceptor1 * Receptor1)^h})) * \frac{KInhibicion}{KInhibicion + (BReceptor1 * Receptor1)^h})$$

$$\frac{dReceptor}{dt} = Receptor + (-1 * (BReceptor * Receptor)) + (-1 * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dCanalDeSalida}{dt} = CanalDeSalida$$

$$\frac{dReceptor1}{dt} = Receptor1 + (-1 * (BReceptor1 * Receptor1)) + (-1 * \frac{KInhibicion}{KInhibicion + (BReceptor1 * Receptor1)^h})$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de la Célula que contiene una Señal Simple desde un receptor a un canal de salida, la cual está afectada por una Catálisis.

Entrada	Se agregan un componente Célula, se incorpora igualmente un componente Receptor y otro CanalDeSalida y luego se relacionan a través de una SeñalSimple. También se dirige hacia esta una Catálisis proveniente de un receptor de la Célula – incorporado previamente. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar. **(Fig 4.7)**
Resultado Esperado	Se conforma correctamente el modelo gráfico y se obtiene la ecuación generada para el modelo matemático presente.
Resultado de la prueba	Al agregar gráficamente los componentes se obtuvieron satisfactoriamente las ecuación es diferenciales esperadas y correspondientes a los componentes presentes en la interacción – Célula, Receptor, Receptor1 y CanalDeSalida - en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 6)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

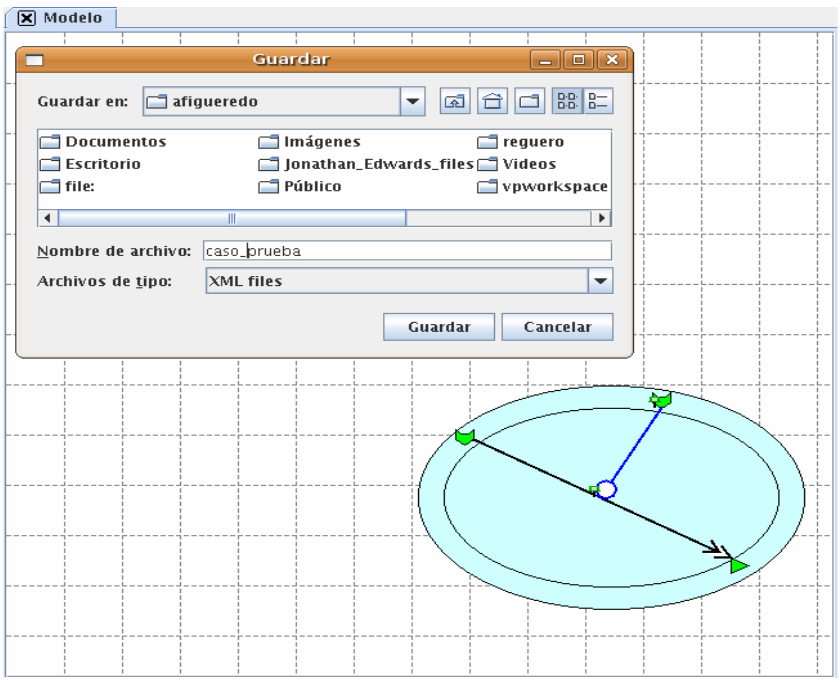


Figura 4.7

Resultado 6:

$$\frac{dCelula}{dt} = (((KbCelula * ACelula) * (\frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h} * \frac{KCatalysis1 + (BReceptor1 * Receptor1)^h}{KCatalysis2 + (BReceptor1 * Receptor1)^h})) * \frac{KCatalysis1 + (BReceptor1 * Receptor1)^h}{KCatalysis2 + (BReceptor1 * Receptor1)^h})$$

$$\frac{dReceptor}{dt} = Receptor + (-1 * (BReceptor * Receptor)) + (-1 * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dCanalDeSalida}{dt} = CanalDeSalida$$

$$\frac{dReceptor1}{dt} = Receptor1 + (-1 * (BReceptor1 * Receptor1)) + (-1 * \frac{KCatalysis1 + (BReceptor1 * Receptor1)^h}{KCatalysis2 + (BReceptor1 * Receptor1)^h})$$

Caso de Uso	Guardar en formato MathML
Caso de Prueba	Obtener modelo matemático de los componentes que quedan – Célula, Citoquinas – luego de quitarle componentes y señales a un modelo matemático complejo.
Entrada	Se agregan componentes y señales tal y como se muestra en la figura. Se obtiene un modelo matemático inicial de este modelo biológico. Se retiran los componentes y señales escogidos. Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en MathMl del modelo matemático. Se especifica la ruta del archivo a guardar.**(Fig 4.8 y Fig 4.9)**
Resultado Esperado	Se obtienen las ecuaciones generadas para el modelo matemático modificado.
Resultado de la prueba	Al retirar gráficamente los componentes y señales se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en el modelo – Célula,

	Citoquinas - en el modelo matemático, en el formato MathMl, el cual fue interpretado por el browser. **(Resultado 7 y Resultado 8)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Debe especificarse una ruta correcta para el fichero MathMl generado.

Entrada:

Entrada Inicial:

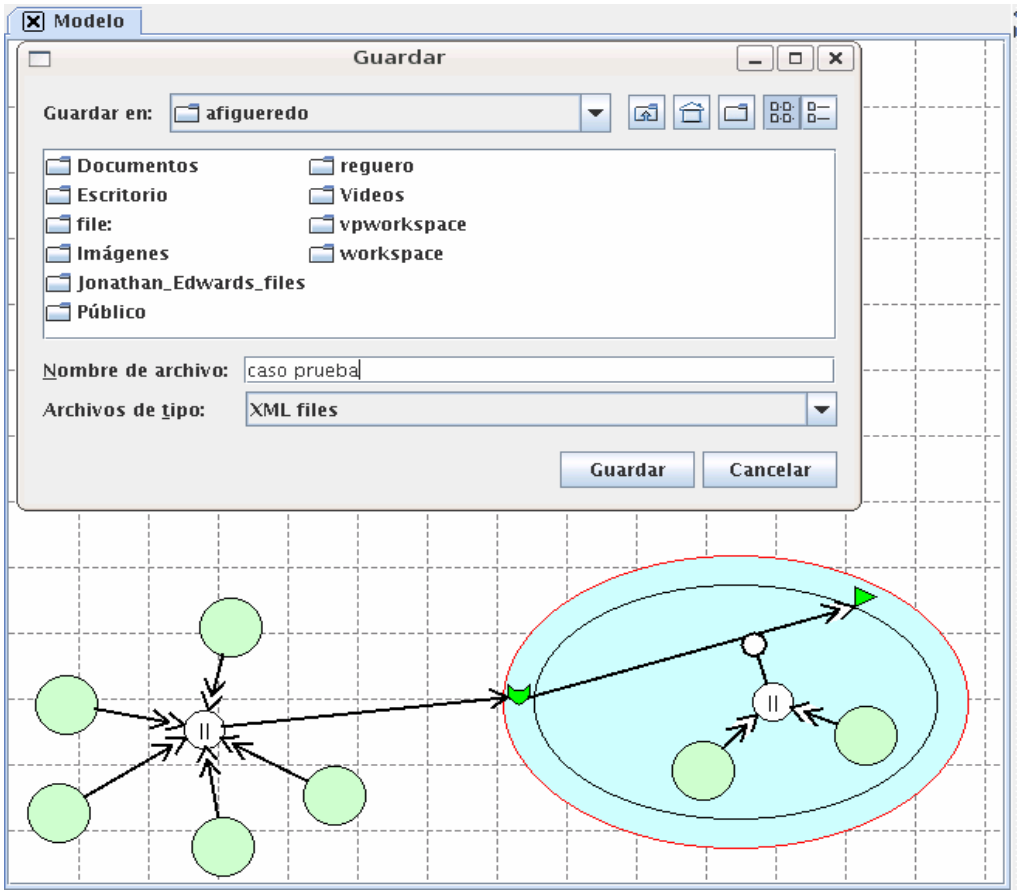


Figura 4.8

- **Resultado 7, de la Entrada Inicial :**

$$\frac{dCelula}{dt} = (((((KbCelula * ACelula) * (\frac{(BReceptor * Receptor)^h}{KSeñal5 + (BReceptor * Receptor)^h} * \frac{KCatalysis1 + Catalysis^h}{KCatalysis2 + Catalysis^h})) * \frac{(BCitoquina5 * Citoquina5)^h}{KSeñal6 + (BCitoquina5 * Citoquina5)^h}) * -$$

$$\frac{dCitoquina}{dt} = Citoquina + (-1 * (BCitoquina * Citoquina)) + (-1 * \frac{(BCitoquina * Citoquina)^h}{KSeñal + (BCitoquina * Citoquina)^h})$$

$$\frac{dCitoquina1}{dt} = Citoquina1 + (-1 * (BCitoquina1 * Citoquina1)) + (-1 * \frac{(BCitoquina1 * Citoquina1)^h}{KSeñal1 + (BCitoquina1 * Citoquina1)^h})$$

$$\frac{dCitoquina2}{dt} = Citoquina2 + (-1 * (BCitoquina2 * Citoquina2)) + (-1 * \frac{(BCitoquina2 * Citoquina2)^h}{KSeñal2 + (BCitoquina2 * Citoquina2)^h})$$

$$\frac{dCitoquina3}{dt} = Citoquina3 + (-1 * (BCitoquina3 * Citoquina3)) + (-1 * \frac{(BCitoquina3 * Citoquina3)^h}{KSeñal3 + (BCitoquina3 * Citoquina3)^h})$$

$$\frac{dCitoquina4}{dt} = Citoquina4 + (-1 * (BCitoquina4 * Citoquina4)) + (-1 * \frac{(BCitoquina4 * Citoquina4)^h}{KSeñal4 + (BCitoquina4 * Citoquina4)^h})$$

$$\frac{dReceptor}{dt} = Receptor + (-1 * (BReceptor * Receptor)) + (-1 * Diferenciacion) + (-1 * \frac{(BReceptor * Receptor)^h}{KSeñal5 + (BReceptor * Receptor)^h})$$

$$\frac{dCanalDeSalida}{dt} = CanalDeSalida$$

$$\frac{dCitoquina5}{dt} = Citoquina5 + (-1 * (BCitoquina5 * Citoquina5)) + (-1 * \frac{(BCitoquina5 * Citoquina5)^h}{KSeñal6 + (BCitoquina5 * Citoquina5)^h})$$

$$\frac{dCitoquina6}{dt} = Citoquina6 + (-1 * (BCitoquina6 * Citoquina6)) + (-1 * \frac{(BCitoquina6 * Citoquina6)^h}{KSeñal7 + (BCitoquina6 * Citoquina6)^h})$$

- **Cambios a Entrada:**

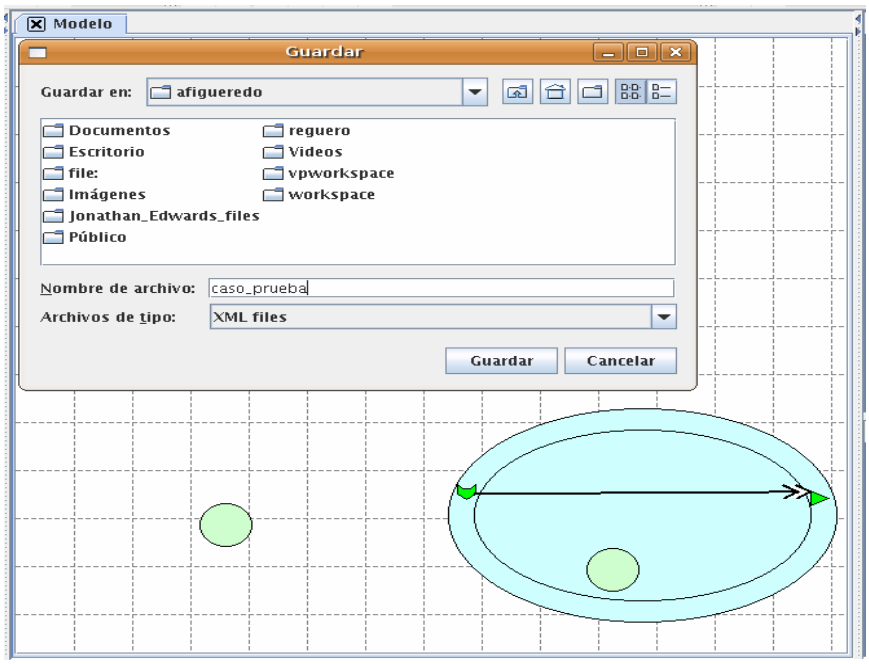


Figura 4.9

- **Resultado 8, de los cambios:**

$$\frac{dCelula}{dt} = ((KbCelula * ACelula) * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dCitoquina}{dt} = Citoquina$$

$$\frac{dCitoquina1}{dt} = Citoquina1$$

$$\frac{dReceptor}{dt} = Receptor + (-1 * (BReceptor * Receptor)) + (-1 * \frac{(BReceptor * Receptor)^h}{KSeñal + (BReceptor * Receptor)^h})$$

$$\frac{dCanalDeSalida}{dt} = CanalDeSalida$$

4.4.2 CASO DE USO: Guardar en formato SBML

Caso de Uso	Guardar en formato SBML
Caso de Prueba	Obtener documento en formato SBML con solamente "*species".*
Entrada	Se agregan componentes de tipo *"species"* - los cuales son los conectores o señales, receptores, operadores AND y OR, canales de salida e iónicos, citoquinas – en relaciones que se definieron arbitrariamente. **(Fig 4.10)** Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en SBML del modelo de sistema biológico. Se llenan los datos requeridos para el documento y el modelo. Se especifica la ruta del archivo a guardar.
Resultado Esperado	Se conforma correctamente el documento SBML, especificándose en su contenido el formato MathMl correspondiente a cada componente visual incorporado al modelo, se colocaron correctamente los datos especificados al documento y por último también se obtuvo satisfactoriamente una lista de las "*species"* presentes sin ningún tipo de *"compartiment".*
Resultado de la prueba	Al agregar gráficamente los componentes visuales se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y

	correspondientes a los componentes presentes en la interacción – dos pares de citoquina y una aislada, los dos operadores OR y AND respectivamente(ver **Fig 4.10**), los seis conectores de señales simples , lo cual fue interpretado por un editor internacional de SBML. **(Resultado Fig 4.11)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Deben entrarse correctamente y llenarse los datos requeridos del documento, estableciendo distintos *"ids"* entre el modelo y el documento. Debe especificarse una ruta correcta para el fichero SBML generado.

Entrada:

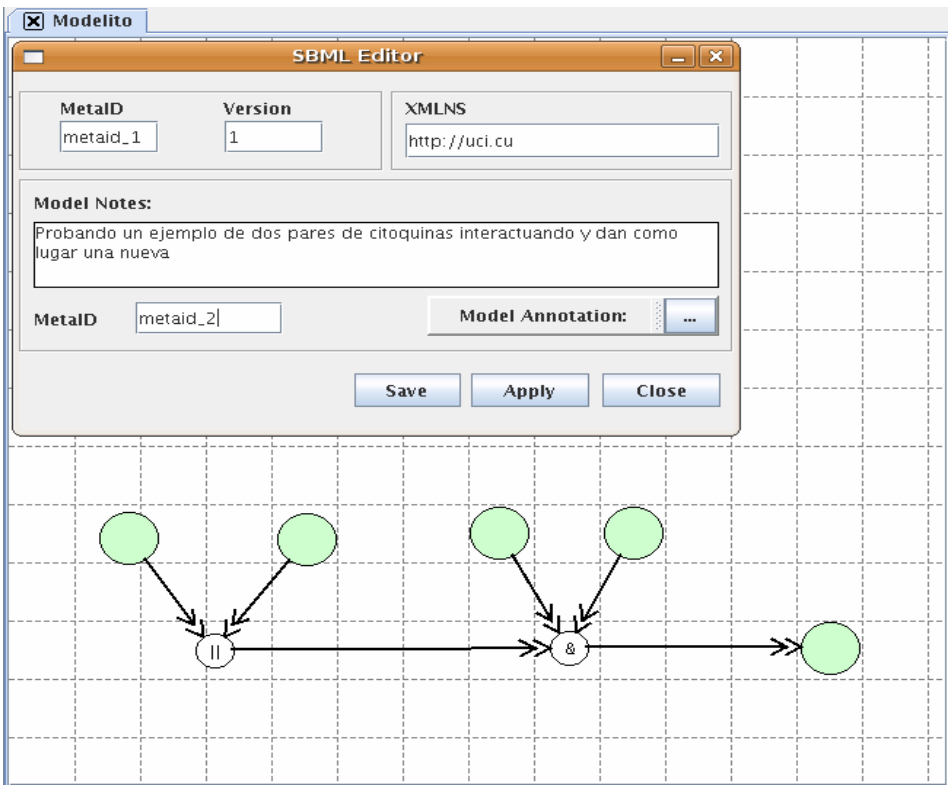


Figura 4.10

Resultado:

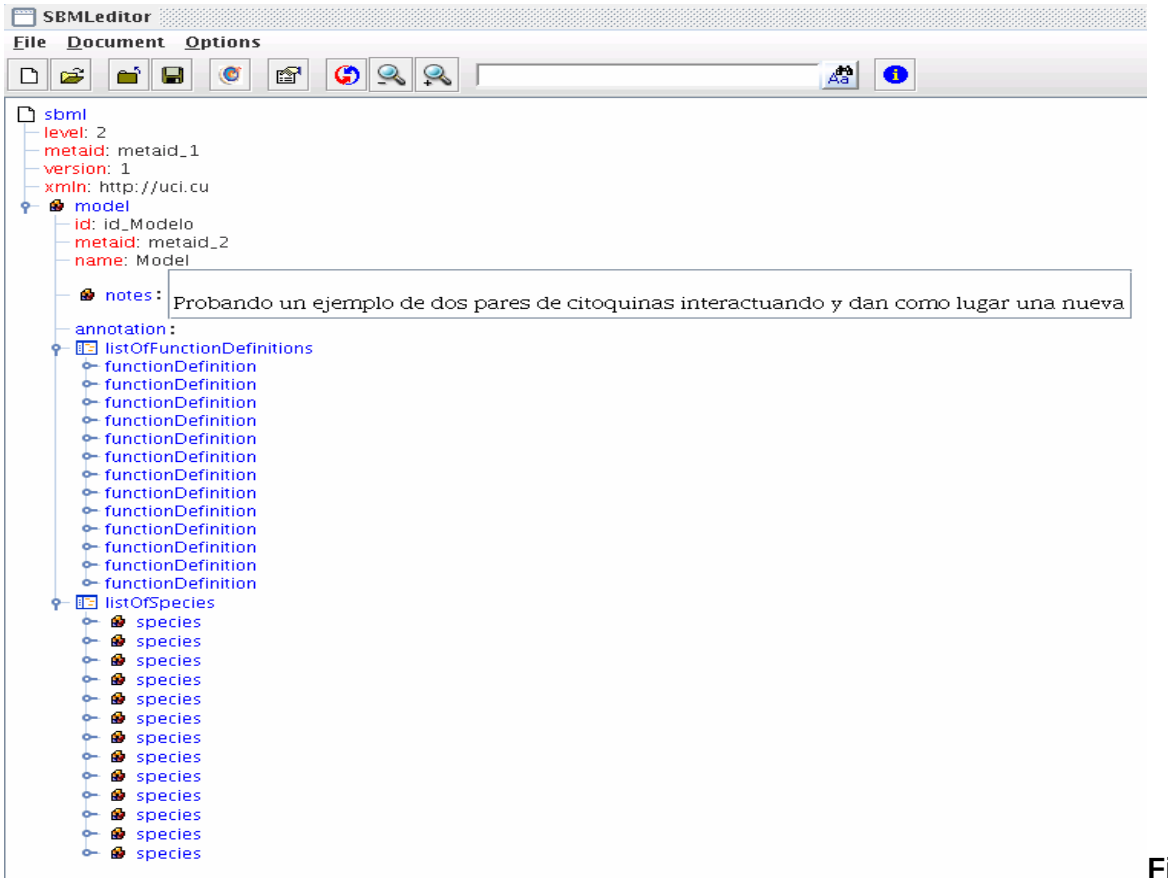


Figura 4.11

Caso de Uso	Guardar en formato SBML
Caso de Prueba	Obtener documento en formato SBML con solamente "*compartments*".
Entrada	Se agregan componentes de tipo *"compartments"* -los cuales son las células– en relaciones que se definieron arbitrariamente. **(Fig 4.12)** Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en SBML del modelo de sistema biológico. Se llenan los datos requeridos para el documento y el modelo. Se especifica la ruta del archivo a guardar.

Resultado Esperado	Se conforma correctamente el documento SBML, especificándose en su contenido el formato MathMl correspondiente a cada componente visual incorporado al modelo, se colocaron correctamente los datos especificados al documento y por último también se obtuvo satisfactoriamente una lista de las "*compartments"* presentes sin ningún tipo de *""species.*
Resultado de la prueba	Al agregar gráficamente los componentes visuales se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción – tres células (ver **Fig 4.12**), los seis conectores de señales simples , lo cual fue interpretado por un editor internacional de SBML. **(Resultado Fig 4.13)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Deben entrarse correctamente y llenarse los datos requeridos del documento, estableciendo distintos "*ids"* entre el modelo y el documento. Debe especificarse una ruta correcta para el fichero SBML generado.

Entrada:

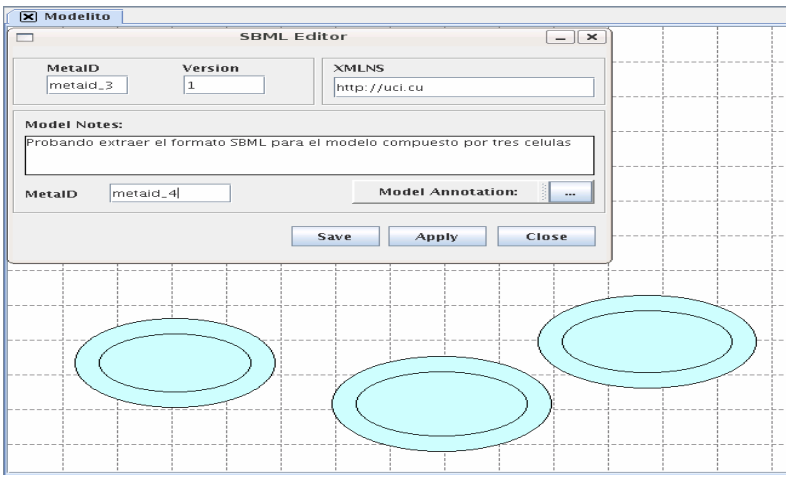


Figura 4.12

Resultado:

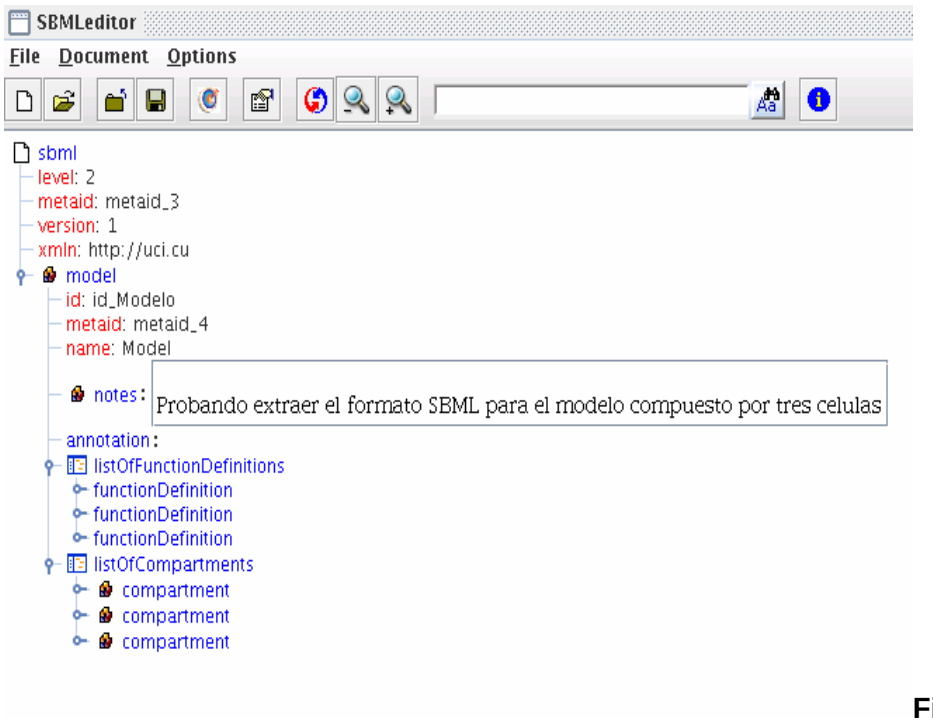


Figura 4.13

Caso de Uso	Guardar en formato SBML
Caso de Prueba	Obtener documento en formato SBML con "*compartiments*" y "species".
Entrada	Se agregan componentes de tipo *"compartiments" y "species"* en relaciones que se definieron arbitrariamente. **(Fig 4.12)** Se hace uso de cualquiera de los menús de la aplicación donde aparezca la opción correspondiente al salvado en SBML del modelo de sistema biológico. Se llenan los datos requeridos para el documento y el modelo.

	Se especifica la ruta del archivo a guardar.
Resultado Esperado	Se conforma correctamente el documento SBML, especificándose en su contenido el formato MathMl correspondiente a cada componente visual incorporado al modelo, se colocaron correctamente los datos especificados al documento y por último también se obtuvo satisfactoriamente una lista de las "*compartments"* y *"species".*
Resultado de la prueba	Al agregar gráficamente los componentes visuales se obtuvieron satisfactoriamente las ecuaciones diferenciales esperadas y correspondientes a los componentes presentes en la interacción – dos células(ver **figura 4.14**) y los restantes componentes simples y conectores, lo cual fue interpretado por un editor internacional de SBML. **(Resultado Fig 4.15)**
Condiciones	Debe existir en el modelo gráfico del Sistema Biológico modelado el componente Modelo_BioSyS y ser un componente del tipo Modelo. Deben entrarse correctamente y llenarse los datos requeridos del documento, estableciendo distintos "*ids"* entre el modelo y el documento. Debe especificarse una ruta correcta para el fichero SBML generado.

Entrada:

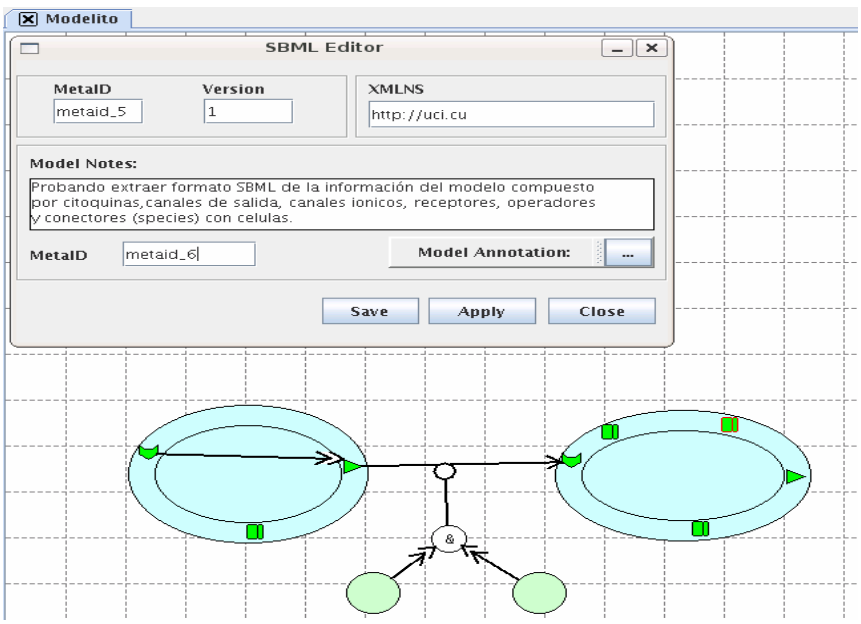
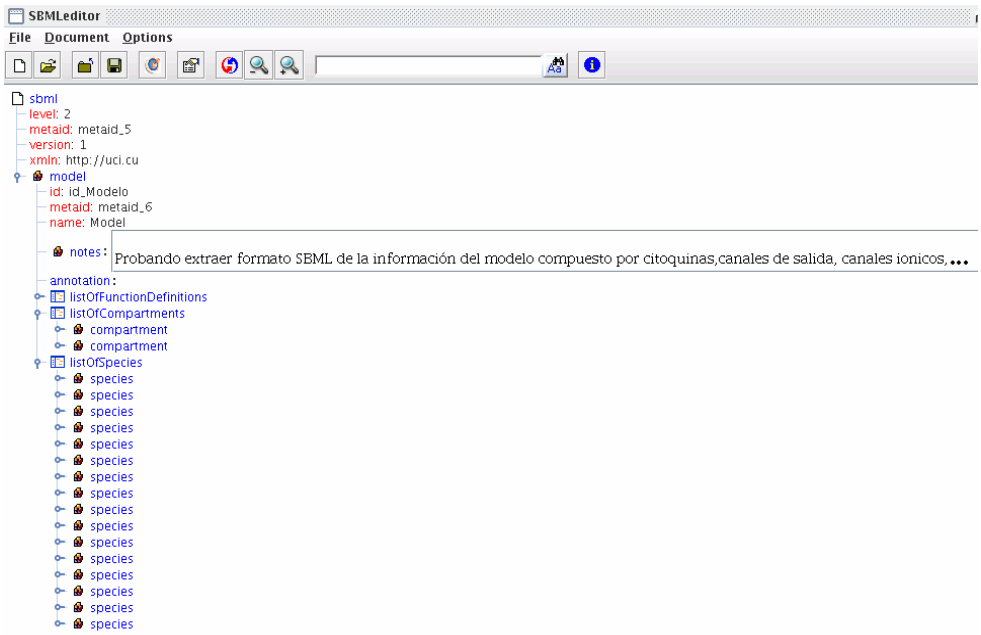


Figura 4.14

Resultado



: **Figura 4.15**

4.5. CONCLUSIONES

En este capítulo se trató la validación del software mediante los casos de prueba de caja negra para el correcto funcionamiento de la aplicación, comprobándose que no existen errores en las funciones operativas del software. Además se definió el diagrama de componentes con el objetivo de brindar una idea de cómo se implementó el software en término de componentes. Se desarrollaron varias pruebas de aceptación usando el método de caja negra y la técnica de particiones de equivalencia para garantizar que los requisitos funcionales fueron cumplidos.

CONCLUSIONES GENERALES

- Se desarrolló una aplicación que permite modelar gráficamente Sistemas Biológicos para generar el modelo matemático correspondiente a través de sistemas de ecuaciones diferenciales y que almacenan los resultados en ficheros, cuyos formatos corresponden a los formatos internacionales SBML y MathML.
- Se realizó el análisis y diseño de una aplicación para modelar gráficamente Sistemas Biológicos y obtener el modelo matemático correspondiente.
- Se implementó una aplicación en Java para modelar gráficamente Sistemas Biológicos que puedan ser descritos mediante dinámica de población, y que permita obtener el modelo matemático correspondiente al modelo gráfico.
- Se comprobó el correcto funcionamiento de la aplicación realizando pruebas de caja.

RECOMENDACIONES

- Incorporarle a la herramienta otras técnicas con el objetivo de profundizar la modelación gráfica permitiendo presentar mayor número de componentes químicos y afectaciones definidas en el curso del tiempo.
- Lograr la integración con el módulo del editor de ecuaciones diferenciales que permitirá editar el sistema de ecuaciones correspondientes al modelo matemático.

BIBLIOGRAFÍA

Referencias:

[1]. BIOINFORMATIC@.es Newsletter. [En línea: 2 de enero 2008] Citado de : http://www.webzinemaker.com/digitalbiology

[2] KITANO "A graphical notation for biochimical networks". Kitano, H. 5, 2003, Biosilico, Vol. 1.

[3] TADMOR,BRIGITTA."Interdisciplinary research and education at the biology–engineering–computer science interface: a perspective". 17, 2005, Biosilico, Vol. 10.

[4] OSORIO, KAREL," Plataforma computacional para el desarrollo de la Biología de Sistema. Facultad de Matemática Computación", Universidad de La Habana. Ciudad de la Habana: 13,s.n., 2004.

[5] SBML.org,"The Systems Biology Markup Languaje" [En línea: 30 de enero del 2008] Citado de : http://sbml.org/Main_Page.

[6] ROBLES, GREGORIO; FERRER, JORGE. "Programación eXtrema y Software Libre. Universidad politécnica de Madrid". Madrid: 46, s.n., 2002

[7] JACOBSON, IVAR; BOOCH, GRADY Y RUMBAUGH, JAMES. "El proceso unificado de software". Primera edición. Pearson Educación, 14,S.A. Año: 2000.

[8] RAMIREZ, ALEJANDRO. "Subversion 2004", [En línea: 3 de febrero 2008] Citado de: http://polaris.dit.upm.es/~rubentb/docs/subversion/TutorialSubversion/index.html

[9] LAGO, RAMIRO. "Patrones de diseño software". [En línea: 5 febrero del 2008]. Citado de: http://www.proactiva-calidad.com/java/patrones/index.html#algunos_patrones.

[10] S. PRESSMAN, ROGER. "Ingeniería de Software: Un enfoque práctico". Tercera Edición, Mc Graw Hill 1993. [En línea: 10 febrero del 2008]. Citado en: http://www.rspa.com/

[11] CLIKEAR.COM, [En línea: 20 de febrero 2008] Citado de : http://www.clikear.com/manuales/csharp/c10.aspx

[12] ENTELOS.com, [En línea: 15 de marzo 2008] Citado de:
http://www.entelos.com/index.php

[13] GENOMATICA.com. [En línea: 15 de marzo 2008] Citado de:
http://www.genomatica.com/index.php

[14] GNS.com [En línea: 16 de marzo 2008] Citado de:
http://www.gnsbiotech.com/static_content/

[15] UPTODOWN.com .[En l'inea: 13 de junio 2008] Citado de:
http://netbeans-ide.uptodown.com/

Bibliografía revisada:

- ✓ BIOPHYSICS, C. B. A. "ByoDyn.A Systems Biology Simulator" Consultado en: http://diana.imim.es/software.php/ByoDyn
- ✓ PARADIGM, V. Documentation [Consultado el: 18 de enero de 2008]. Disponible en: http://www.visual-paradigm.com/
- ✓ PATRICIO LETELIER y PENADÉS, M. C. Métodologías ágiles para el desarrollo de software: eXtreme Programming (XP) Universidad Politécnica de Valencia: [Consultado el: 19 de enero de 2008]. Disponible en: http://www.willydev.net/descargas/masyxp.pdf
- ✓ RO, C. Modelo matemático en medicina y biología. Bases teóricas y fundamentos [Consultado el: 29 de enero de 2008]. (Revista Invest Clin). Disponible en: http://www.imbiomed.com/Innsz/Nnv46n4/espanol/Wnn44-07.html.
- ✓ TERRA. La Biología de Sistemas mejorará el tratamiento de enfermedades Disponible en: http://www.imbiomed.com/Innsz/Nnv46n4/espanol/Wnn44-07.html.
- ✓ MENDOZA SANCHEZ,MARIA. "Metodologías De Desarrollo De Software" Consultado en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf
- ✓ JACOBSON, IVAR; BOOCH, GRADY Y RUMBAUGH, JAMES. "El proceso unificado de software". Primera edición. Pearson Educación, S.A. Año: 2000.
- ✓ WIKIPEDIA. "Subversion". [Consultado en: http://es.wikipedia.org/wiki/Subversion

- ✓ WIKIPEDIA. ”Arquitectura de software”. Consultado en: http://es.wikipedia.org/wiki/Arquitectura_software#Modelos_o_vistas
- ✓ S. PRESSMAN, ROGER. “Ingeniería de Software: Un enfoque práctico ”. Tercera Edición, Mc Graw Hill 1993. [Consultado en : http://www.rspa.com/]
- ✓ S. PRESSMAN, ROGER. “Software Engineering ”.4th Edición, Mc Graw Hill 1998. [Consultado en : http://www.rspa.com/]
- ✓ NORVIG, PETER. “Design patterns en Dinamic Programming ”. Design Patterns: Elements of Reusable Object-Oriented Software. [Consultado en: http://www.norvig.com/design-patterns/ppframe.htm]
- ✓ LARMAN, CRAIG .“Applying UML and Patterns”.Craig Larman. Prentice Hall [Consultado en : http://unjobs.org/authors/craig-larman]
- ✓ GRAND, MARK. “Patterns in Java, Volume 2”. Mark Grand, Wiley Computer Publishing. [Consultado en : http://www.ercb.com/brief/brief.0137.html]
- ✓ LARMAN, CRAIG. “Objects by design”.[Consultado en: http://www.cis.gsu.edu/~cstucke/cis3300/larman_process.pdf]
- ✓ ARCHER, TOM. “C# a Fondo” Editorial McGraw-Hill, 1ª edición Año 2001. PDF, 400 pag.
- ✓ FONSECA, BERNARDO Y GUERRA.” Yudiel. Sistema de servicio comunitario. ISP José Antonio Echevería”, Ciudad Habana: s.n., 2004. Tesis.
- ✓ ALEJANDRO MEDINA SANTIAGO; EVA VALDEZ ALEMÁN, ¨ Simulación de sistemas analógicos para la solución de un grupo de problemas de optimización¨ [Consultado el 9 de abril del 2008]. Disponible en: http://www.uvmnet.edu/investigacion/episteme/numero6-6/reportes/a_simulacion.asp.
- ✓ BIOPHYSICS, C. B. A. ByoDyn [Consultado el: 20 de abril de 2008]. Disponible en: http://diana.imim.es/ByoDyn
- ✓ BRIGITTA TADMOR y TIDOR, B. Interdisciplinary research and education at the biology–engineering–computer science interface: a perspective. marzo, 2008, vol. 10.
- ✓ CELLWARE. Cellware 3.0.2 is available [Consultado el: 5 de mayo de 2008]. Disponible en: http://www.cellware.org/index.html.
- ✓ DORADO, I. C. Simulación de sistemas [Consultado el: 16 de mayo de 2008]. Disponible en: http://www.monografias.com/trabajos20/simulacion-sistemas/simulacion-sistemas.shtml.
- ✓ EXPERIMENTAR. ¿Qué es un modelo? [Consultado el: 16 de mayo de 2008]. Disponible en: http://www.experimentar.gov.ar/newexperi/NOTAS/maquinaviva/modelo.htm

.

- ✓ JAVA, T. D. Características de Java [Consultado el: 19 de mayo del 2008]. Disponible en: http://www.cica.es/formacion/JavaTut/Intro/tabla.html.

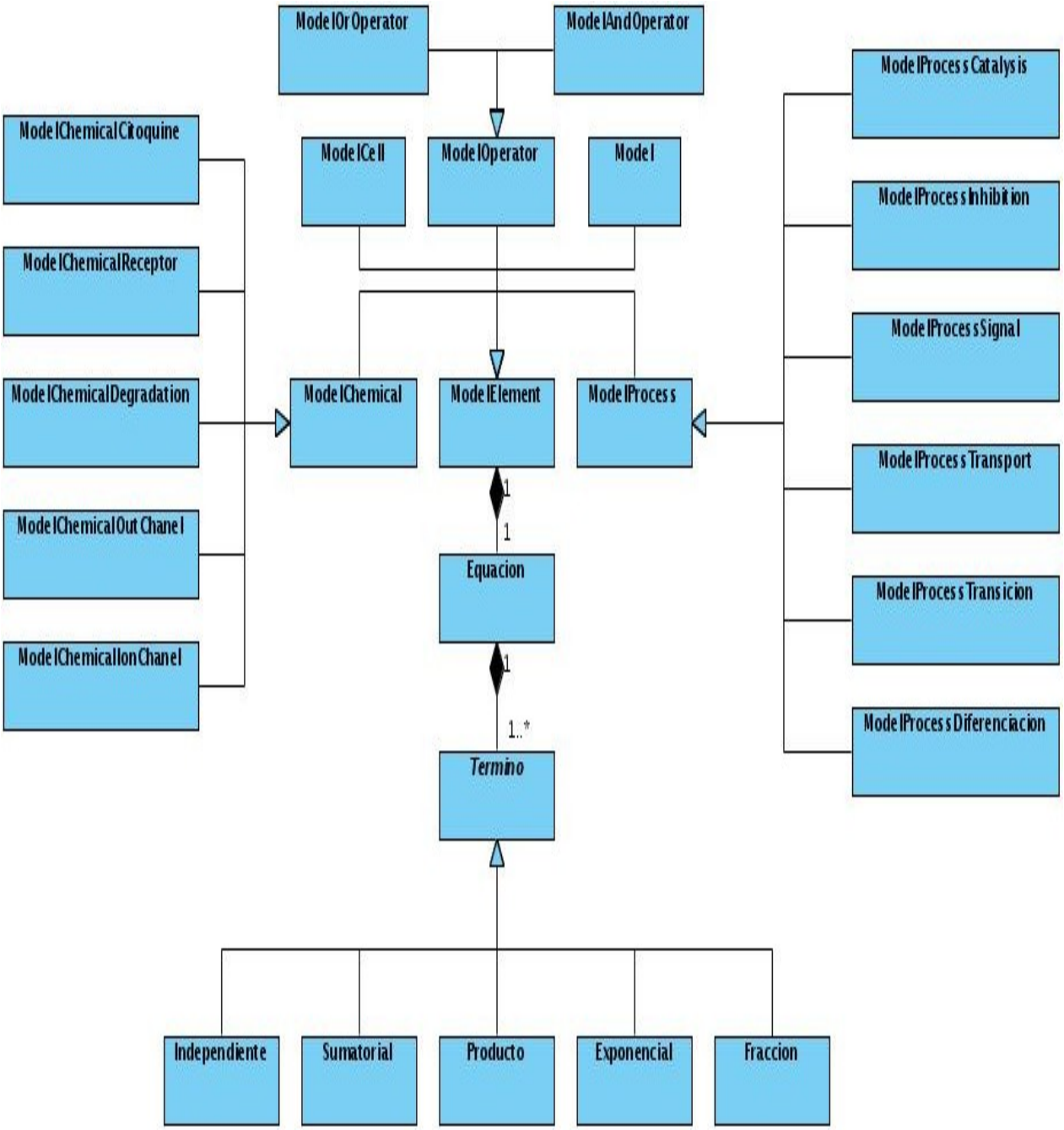
- ✓ LEÓN, A. R. S. Herramienta para la modelación de sistemas biológicos. Universidad de las Ciencias Informáticas, 2006.

- ✓ MARÍA JOSÉ FERRÓN y GONZÁLEZ, J. P. Sistemas de Programas [Consultado el: 23 de mayo de 2008]. Disponible en: http://www.ldc.usb.ve/ldc/principal.php

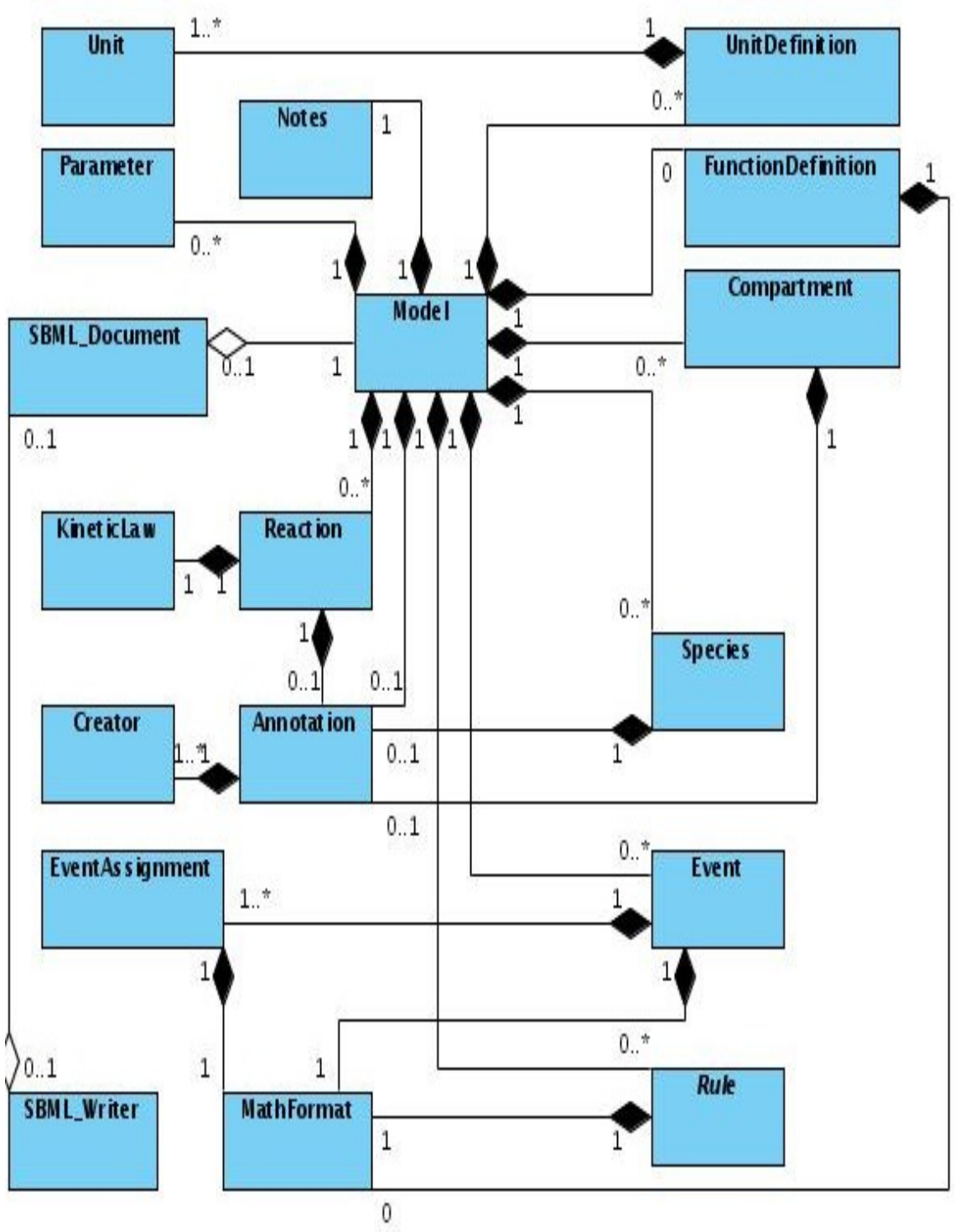
- ✓ MATHWORKS, T. SimBiology 2.1.1 [Consultado el: 24 de mayo de 2008]. Disponible en: http://www.mathworks.com/products/simbiology/description5.html.

ANEXOS

A.1 Diagrama de clases perteneciente a la capa del modelo (MathML)

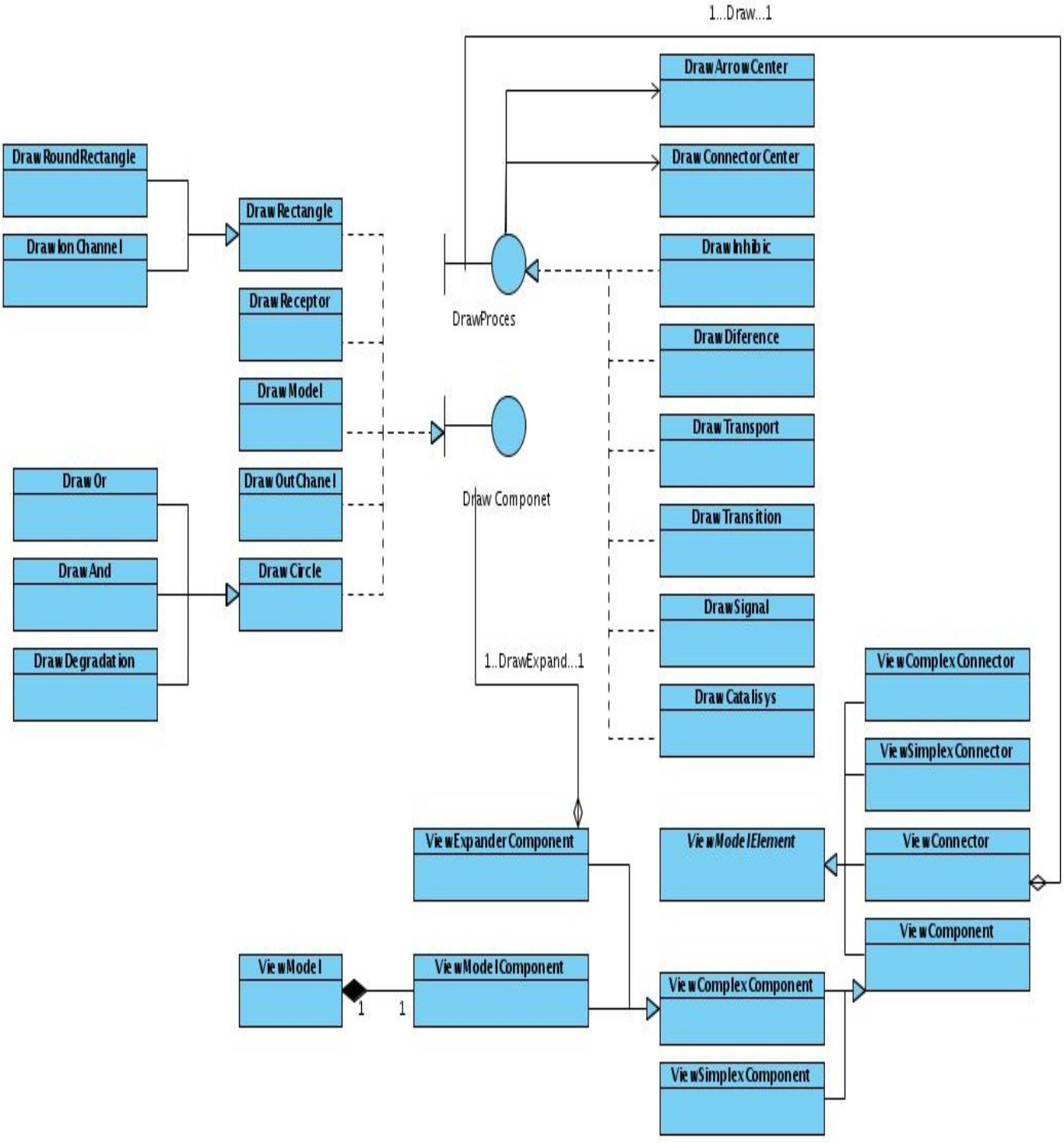


A.2 Diagrama de clases perteneciente a la capa del Modelo (SBML Worker)



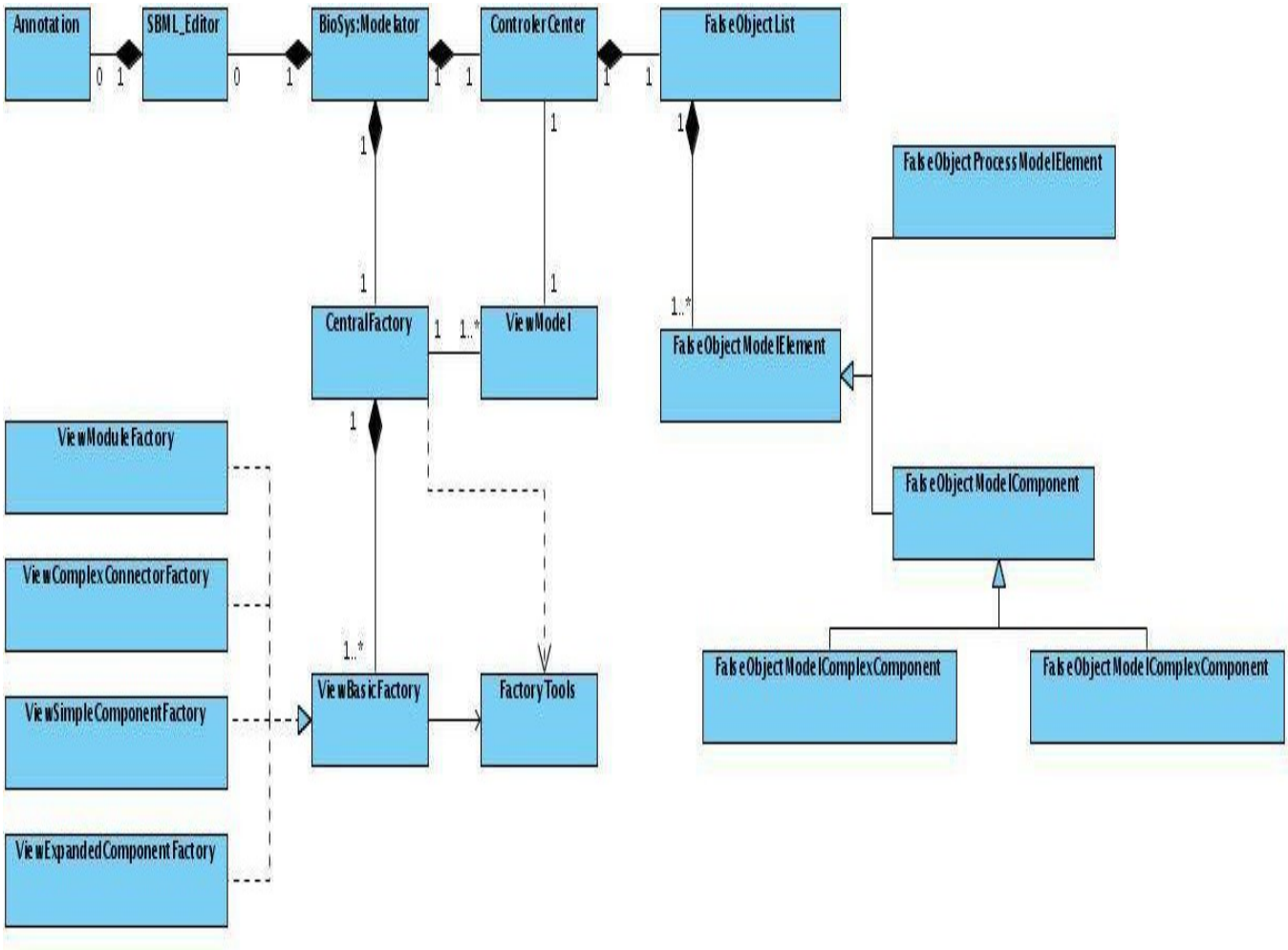
A.3 Diagrama de clases perteneciente a la capa de la vista

[Citada de la tesis responsable de las capas vista y controlador**]**.

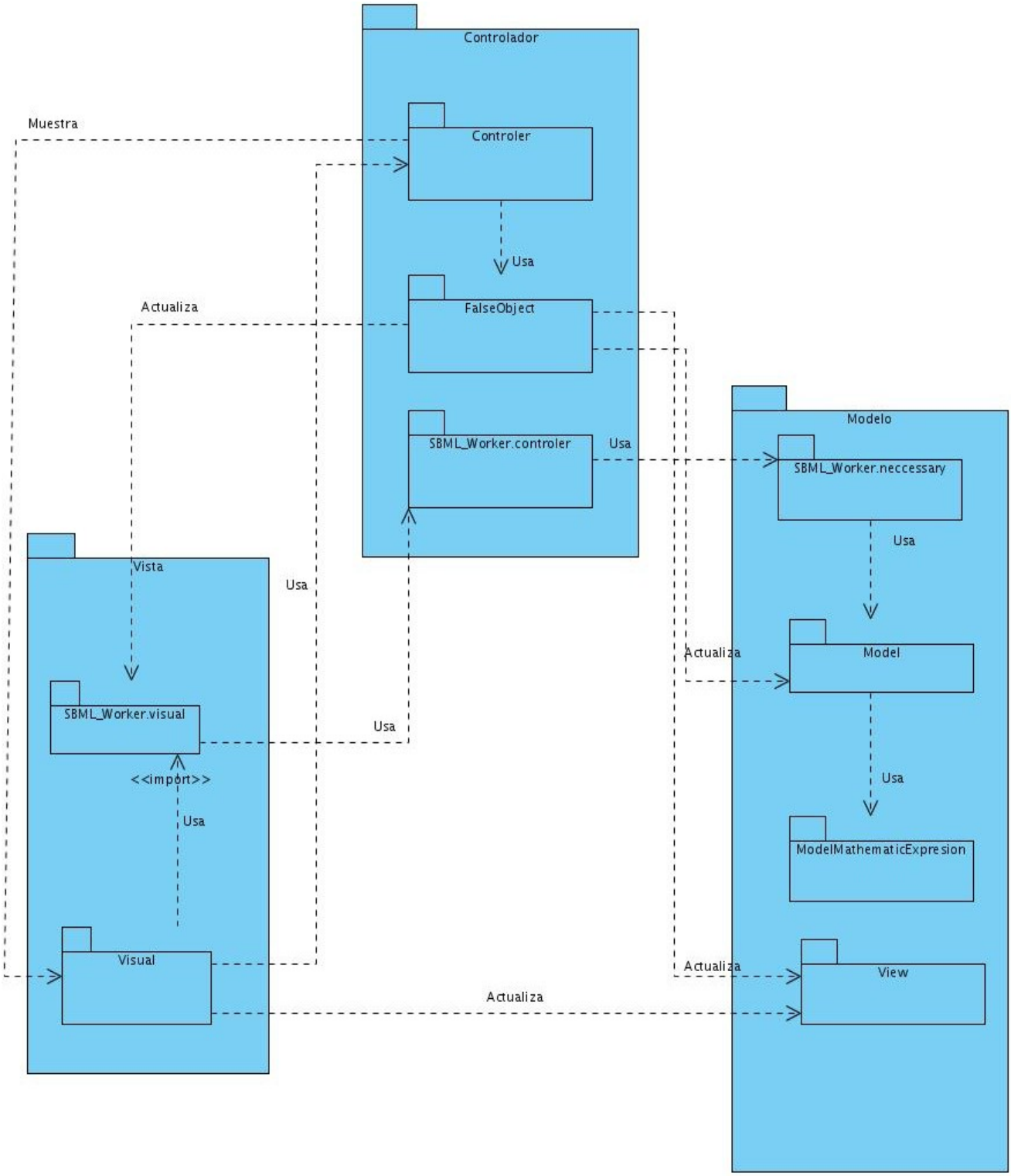


A.4 Diagrama de clases perteneciente a la capa del controlador

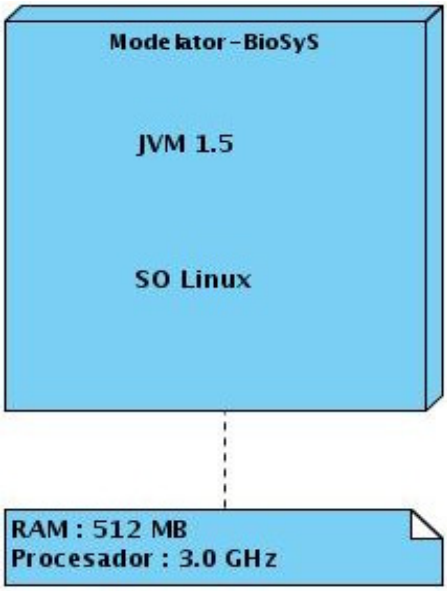
[Citada de la tesis responsable de las capas vista y controlador**]**



A.5 Vista arquitectónica



A.6 Diagrama de Despliegue



A.7 Descripción de de las clases del diseño de forma detallada

1. Clases Visuales

Nombre: SBML_Editor	
Tipo de clase: Interfaz	
Atributo:	**Tipo:**
textField_2	jTexField
textField_3	jTexField
textField_4	jTexField
textField_5	jTexField
textArea	JTextArea
frame	jFrame
list	FalseObjeteList
compartment	ArrayList<FalseObjectModelElement>
species	ArrayList<FalseObjectModelElement>

url	java.io.File
document	src.SBML_Document
Para cada responsabilidad:	
Nombre:	**Descripción:**
void main()	Levanta la aplicación
SBML_Editor()	Constructor de la clase
org.jdom.Element SaveALLToSBML()	Salva la información al formato SBML
ObtainTypeOfCompartmentThatContainThis()	Obtiene el id del tipo de compartimento que contiene al componente visual entrado por parámetro
void saveAsToSBML()	Salva el fichero SBML
void saveToSBML()	Salva el fichero SBML (secundario)
void initialize()	inicializa los componentes visuales del SBML_Editor
src.SBML_Document getDocument()	Propiedad de la clase
void setDocument()	Propiedad de la clase
void AddAnnotation()	Agrega una anotación a la anotación del documento SBML

Nombre: Anotation	
Tipo de clase: Interfaz	
Atributo:	**Tipo:**
list_1	javax.swing.JList
table	javax.swing.JTable
textField	javax.swing.JTextField
textField_2	javax.swing.JTextField
textField_3	javax.swing.JTextField
textField_3	javax.swing.JTextField
frame	javax.swing.JFrame

editor	control.SBML_Editor
annotation	javax.swing.DefaultListModel
Para cada responsabilidad:	
Nombre:	**Descripción:**
void main()	Levanta la aplicación
Annotation()	Constructor de la clase
void initialize()	inicializa los componentes visuales del SBML_Editor

2. Clases Entidad

Nombre: ModelElement	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato SBML

Nombre: Model	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
id	java.lang.String
metaid	java.lang.String
name	java.lang.String
notes	java.lang.String
annotation	necessary.Annotation
listOfUnitDefinitions	java.util.ArrayList
listOfCompartments	java.util.ArrayList
listOfSpecies	java.util.ArrayList
listOfRules	java.util.ArrayList
listOfReactions	java.util.ArrayList
listOfEvents	java.util.ArrayList
Para cada responsabilidad:	
Nombre:	**Descripción:**
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase padre
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar a la ecuación un término, sea en multiplicación, suma o resta
org.jdom.Element SaveToMathMl()	Salvar a formato MathMl
Model()	Constructor de la clase

necessary.Annotation getAnnotation()	Propiedad de la clase
java.lang.Stringvoid getAnnotation()	Propiedad de la clase
java.lang.String getId()	Propiedad de la clase
java.util.ArrayList void setId()	Propiedad de la clase
getListOfCompartments()	Propiedad de la clase
void setListOfCompartments()	Propiedad de la clase
java.util.ArrayList getListOfReactions()	Propiedad de la clase
void setListOfReactions()	Propiedad de la clase
java.util.ArrayList getListOfRules()	Propiedad de la clase
void setListOfRules()	Propiedad de la clase
java.util.ArrayList getListOfSpecies()	Propiedad de la clase
void setListOfSpecies()	Propiedad de la clase
java.util.ArrayList getListOfUnitDefinitions()	Propiedad de la clase
void setListOfUnitDefinitions()	Propiedad de la clase
java.lang.String getMetaid()	Propiedad de la clase
void setMetaid()	Propiedad de la clase
java.lang.String getName()	Propiedad de la clase
void setName()	Propiedad de la clase
java.lang.String getNotes()	Propiedad de la clase
void setNotes()	Propiedad de la clase
Element SaveTo_SBML()	Salvar el modelo en formato SBML

Nombre: ModelCell	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion

id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su padre
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelCell()	Constructor de la clase
void CellState_Born_Death_Diferenciare()	Método para afectar la ecuación con diferenciación
void CellState_Sufer_or_Recieve_Unity	Método para afectar la ecuación con Unión

Nombre: ModelChemical	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase padre

model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Elemen SaveToMathMl()	Salvar su información al formato MathML
ModelChemical()	Constructor de la clase

Nombre: ModelChemicalCitoquine	
Tipo de clase: Entidad	
Atributo	**Tipo**
equation	Equacion
id	String
C	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement Clone()	Método para clonar y devolver una instancia de la clase
Equacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelChemical()	Constructor de la clase Padre

ModelChemicalCitoquine()	Constructor de la clase
String getC()	Propiedad de la clase
void setC()	Propiedad de la clase

Nombre: ModelChemicalReceptor	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
R	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement Clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelChemical()	Constructor de la clase padre
ModelChemicalReceptor()	Constructor de la clase
String getR()	Propiedad de la clase
void setR()	Propiedad de la clase

Nombre: ModelChemicalDegradation	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
D	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
org.jdom.Element void AddToEquationAs()	Agregar término a la ecuación propia
SaveToMathMl()	Salvar su información al formato MathMl
ModelChemical()	Constructor de la clase padre
ModelChemicalDegradation()	Constructor de la clase
String getD()	Propiedad de la clase
void setD()	Propiedad de la clase

Nombre: ModelChemicalOutChanel	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String

OCH	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathML
ModelChemical()	Constructor de la clase padre
ModelChemicalOutChanel()	Constructor de la clase
String getOCH()	Propiedad de la clase
void setOCH()	Propiedad de la clase

Nombre: ModelChemicalIonChanel	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
ICH	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase

ModelElement()	Constructor de su Ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelChemical()	Constructor de la clase padre
ModelChemicalIonChanel()	Constructor de la clase
String getICH()	Propiedad de la clase
void setICH()	Propiedad de la clase

Nombre: ModelProcess	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase padre
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl

ModelProcess()	Constructor de la clase

Nombre: ModelProcessDiferenciacion	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Dif	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelProcess()	Constructor de la clase padre
String getDif()	Propiedad de la clase
void setDif()	Propiedad de la clase
ModelProcessDiferenciacion()	Constructor de la clase

Nombre: ModelProcessCatalysis	
Tipo de clase: Entidad	
Atributo:	**Tipo:**

equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia del la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agrega una ecuación a los términos que tiene incluido
org.jdom.Element SaveToMathMl()	Para obtener la información de este término en formato MathMl
ModelProcess()	Propiedad de la clase padre
ModelProcessCatalysis()	Propiedad de la clase

Nombre: ModelProcessInhibition	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase

Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelProcess()	Constructor de la clase padre
ModelProcessInhibition()	Constructor de la clase

Nombre: ModelProcessSignal	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelProcess()	Constructor de la clase padre
ModelProcessSignal()	Constructor de la clase

Nombre: ModelProcessTransport	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
T	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelProcess()	Constructor de la clase padre
String getT()	Propiedad de la clase
void setT()	Propiedad de la clase
ModelProcessTransport()	Constructor de la clase

Nombre: ModelProcessTransicion	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String

T	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de la clase ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelProcess()	Constructor de la clase padre
String getT()	Propiedad de la clase
void setT()	Propiedad de la clase
ModelProcessTransicion()	Constructor de la clase

Nombre: ModelOperator	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase

ModelElement()	Constructor de la clase padre
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelOperator()	Constructor de la clase

Nombre: ModelOrOperator	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelOperator()	Constructor de la clase padre
ModelOrOperator()	Constructor de la clase

Nombre: ModelAndOperator	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
equation	Equacion
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
ModelElement()	Constructor de su Ancestro
model.ModelElement clone()	Método para clonar y devolver una instancia de la clase
Ecuacion getEquation()	Propiedad de la clase
void setEquation()	Propiedad de la clase
void AddToEquationAs()	Agregar término a la ecuación propia
org.jdom.Element SaveToMathMl()	Salvar su información al formato MathMl
ModelOperator()	Constructor de la clase padre
ModelAndOperator()	Constructor de la clase

3. Clases Entidad: Representación matemática

Nombre: Equacion	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
terminos	ArrayList<Termino>
Para cada responsabilidad:	
Nombre:	**Descripción:**

Equacion()	Constructor de la clase
ArrayList<Termino> getTerminos()	Propiedad de la clase
void setTerminos()	Propiedad de la clase
void AgregarTermino()	Método para agregar un término determinado
void Vaciar()	Método para blanquear la ecuación
org.jdom.Element GetMathMlPresentation	Método para devolver en formato MathML la sumatoria de todos los términos presentes en forma MathMl Presentación
org.jdom.Element GetMathMlContent	Método para devolver en formato MathML la sumatoria de todos los términos presentes en forma MathMl Contenido

Nombre: Termino	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
id	String
Para cada responsabilidad:	
Nombre:	**Descripción:**
void AgregarEquacion()	Método para agregar una ecuación cuando el término es compuesto, se redefine en las clases hijas
Termino()	Constructor de la clase
String getId	Propiedad de la clase
void setId	Propiedad de la clase
org.jdom.Element GetMathMlPresentation	Para obtener la información de este término en formato MathMl

4. Clases Entidad: Representación SBML

Nombre: SBML_Document
Tipo de clase: Entidad

Atributo:	**Tipo:**
level	int
version	int
xmlns	java.lang.String
metaid	java.lang.String
model	src.Model
Para cada responsabilidad:	
Nombre:	**Descripción:**
SBML_Document()	Constructor de la clase
int getLevel()	Propiedad de la clase
void setLevel()	Propiedad de la clase
java.lang.String getMetaid()	Propiedad de la clase
void setMetaid()	Propiedad de la clase
src.Model getModel()	Propiedad de la clase
void setModel()	Propiedad de la clase
int getVersion()	Propiedad de la clase
void setVersion()	Propiedad de la clase
int getXmlns()	Propiedad de la clase
void setXmlns()	Propiedad de la clase

Nombre: Species	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
id	java.lang.String
metaid	java.lang.String
name	java.lang.String

compartment	java.lang.String
annotation	necessary.Annotation
initialAmount	float
initialConcentration	float
Para cada responsabilidad:	
Nombre:	**Descripción:**
java.lang.String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
Species()	Constructor de la clase
float getInitialConcentration()	Propiedad de la clase
void setInitialConcentration	Propiedad de la clase
float getInitialAmount	Propiedad de la clase
void setInitialAmount	Propiedad de la clase
java.util.ArrayList getCompartment	Propiedad de la clase
void setCompartment	Propiedad de la clase
java.lang.String getMetaid()	Propiedad de la clase
void setMetaid()	Propiedad de la clase
java.lang.String getName()	Propiedad de la clase
void setName()	Propiedad de la clase
Element SaveTo_SBML()	Para salvar su información a SBML
necessary.Annotation getAnnotation()	Propiedad de la clase
void setAnnotation	Propiedad de la clase

Nombre: Compartment	
Tipo de clase: Entidad	
Atributo:	**Tipo:**

id	java.lang.String
metaid	java.lang.String
size	java.lang.String
Para cada responsabilidad:	
Nombre:	**Descripción:**
java.lang.String getId()	Propiedad de la clase
void setId()	Propiedad de la clase
java.lang.String getMetaid()	Propiedad de la clase
void setMetaid()	Propiedad de la clase
Compartment()	Constructor de la clase
java.lang.String getSize()	Propiedad de la clase
void setSize()	Propiedad de la clase
Element SaveTo_SBML()	Propiedad de la clase

Nombre: SBML_Writer	
Tipo de clase: Entidad	
Atributo:	**Tipo:**
documento	src.SBML_Document
Para cada responsabilidad:	
Nombre:	**Descripción:**
SBML_Writer()	Constructor de la clase
src.SBML_Document getDocument()	Propiedad de la clase
void setDocument()	Propiedad de la clase
org.jdom.Element SaveAllToMath()	Para salvar su información a SBML

GLOSARIO

- **Bioinformática:** Es la aplicación de los ordenadores y los métodos informáticos en el análisis de datos experimentales y simulación de los sistemas biológicos. Una de las principales aplicaciones de la Bioinformática es la simulación, la minería de datos y el análisis de los datos obtenidos en un estudio.
- **Biología de Sistemas:** Área de investigación científica que se preocupa del estudio de procesos biológicos usando un enfoque sistémico.
- **Metodología:** Define quién hace qué, cómo y cuando.
- **Modelación:** Es un método de obtención del conocimiento, de aplicación en varias ciencias, en el cual se opera con un objeto, no en forma directa sino utilizando cierto sistema intermedio auxiliar conocido como modelo.
- **Modelo:** Representación abstracta de la realidad. Diagramas que representan la estructura de un sistema dado.
- **Patrones:** Unidad de información nombrada, instructiva e intuitiva que captura la esencia de una familia exitosa de soluciones probadas a un problema recurrente dentro de un cierto contexto.
- **Requisitos:** Capacidades o condiciones que se deben cumplir.
- **Sistemas Biológicos**: Sistemas abiertos que operan en condiciones alejadas del equilibrio termodinámico, con muchas y fuertes interacciones no lineales entre sus muchos elementos.
- **Software:** Termino genérico que designa al conjunto de programas que posibilitan realizar una tarea específica en un ordenador.