UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 6

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

Título: **"Nueva funcionalidad para la importación de archivos para el Editor de Ecuaciones de la Plataforma de Simulación de Sistemas Biológicos".**

Autores:
Reynaldo Rodríguez Cruz
Gustavo Cosio Martin

Tutores:
Msc. Gilberto Arias Naranjo.
Ing. Linet Cobo Barreras.

Ciudad de La Habana, Junio del 2009

"Año 50 de la Revolución"

"La mayoría de las ideas fundamentales de la ciencia son esencialmente sencillas y, por regla general pueden ser expresadas en un lenguaje comprensible para todos".

Albert Einstein.

Declaración de Autoría

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

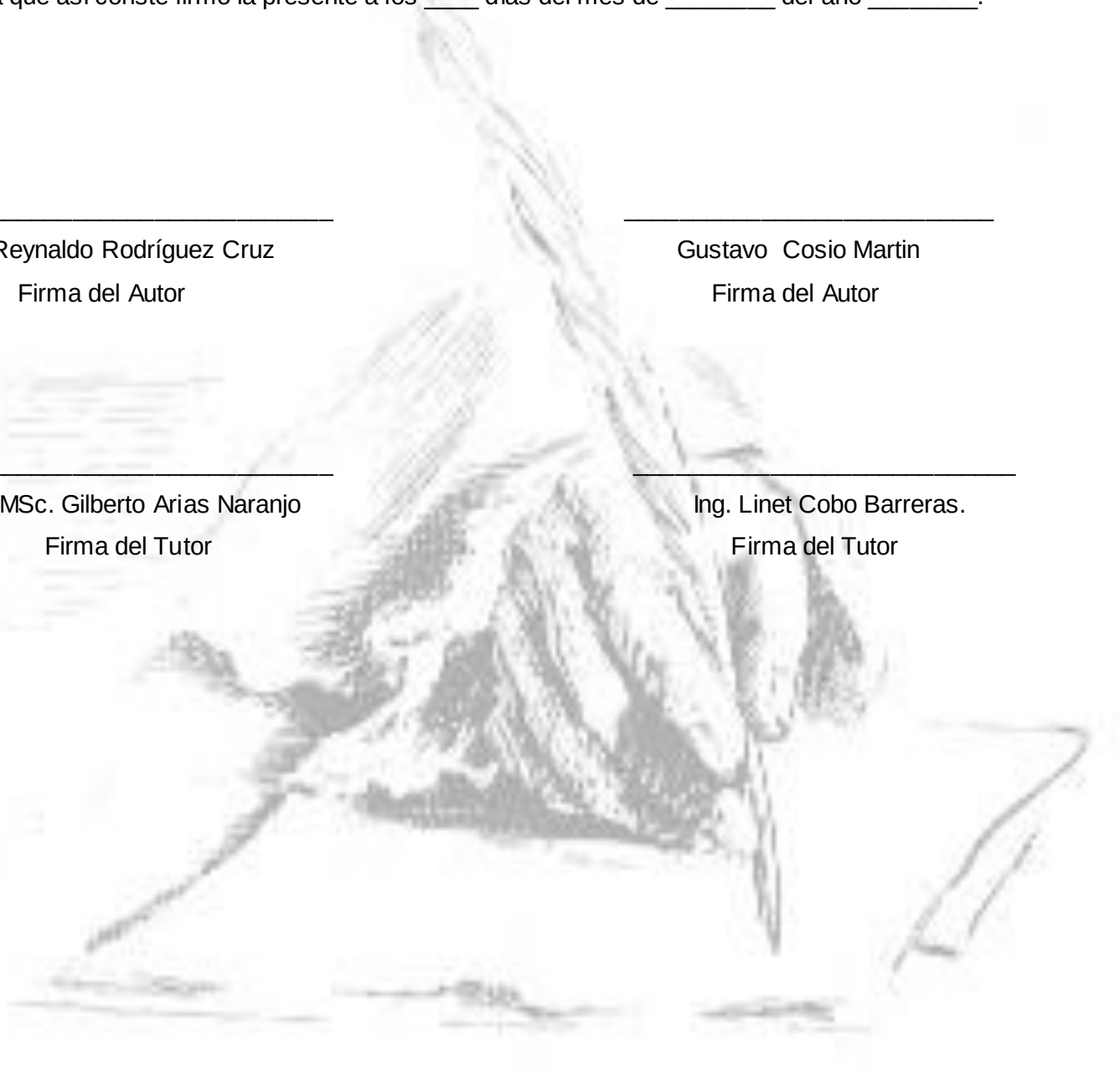
Para que así conste firmo la presente a los ____ días del mes de _____ del año _____.

_____
Reynaldo Rodríguez Cruz
Firma del Autor

_____
Gustavo Cosio Martin
Firma del Autor

_____
MSc. Gilberto Arias Naranjo
Firma del Tutor

_____
Ing. Linet Cobo Barreras.
Firma del Tutor



Datos de Contacto

Gilberto Arias Naranjo: Msc. En Ciencias. Ing. En Ciencias Informáticas.
Correo Electrónico: garias@uci.cu
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Linet Cobo Barreras: Ing. En Ciencias Informáticas.
Correo Electrónico: lcobo@uci.cu
Universidad de las Ciencias Informáticas, Ciudad de la Habana, Cuba.

Agradecimientos

A nuestros tutores Gilberto Arias Naranjo y Linet Cobo Barreras por brindarnos gran ayuda, sus valiosos conocimientos y acertadas observaciones que fueron vitales para el desarrollo de este trabajo.

Muy especial a nuestros padres, que ante todas las dificultades siempre creyeron en nosotros y nos exhortaron a superarnos día a día. Por su inmenso apoyo y guía en cada instante de nuestras vidas principalmente cuando más lo necesitamos. Por su gran dedicación e influencia infinita en nuestra educación, por su confianza y por su amor. Este trabajo también es de ellos pues a ellos les debemos lo que somos.

A nuestra familia, en particular a aquellos que siempre han estado pendientes de nosotros.

A la Revolución Cubana por habernos permitido estudiar en esta universidad tan maravillosa.

A la Universidad de las Ciencias Informáticas por enseñarnos a crecer, a ser mejores personas y a contribuir a convertirnos en profesionales.

Por último queremos agradecer a todas las personas con las cuales hemos tenido la dicha de compartir estos maravillosos años que nunca olvidaremos y vivirán para siempre en nosotros.

Dedicatoria

De Reynaldo Rodríguez Cruz.

A mis padres Magalis y Reynaldo por ser ambos mi principal fuente de inspiración, por haberme brindado tanto apoyo, infinito cariño y sobretodo porque gracias a sus esfuerzos, todo lo que soy hoy como persona se los debo a ellos.

A mi hermanito Rolando por ser sencillamente la persona más especial de mi vida.

A mis abuelas Melba, Caridad y Ramona por ser siempre estar ahí para cuando las necesité.

A mi abuelo Alcides por enseñarme que es el respeto hacia los demás y por haberme enseñado tanto en la vida.

A mis tías Xiomara, Sonia, Narda, Aracilia y María Elena por siempre confiar en mí y por haberme ayudado tanto en los momentos más difíciles.

A mis tíos William, Julio, Jorge, José Ramón, Frank, Alcides y a mis primas Yadira, Yaima, Yaimí y Idalmis por creer en mí.

A todos mis primos y amigos por enseñarme cada día cómo ser mejor persona y por ayudarme en los momentos más duros.

A todas las personas que de una forma u otra han contribuido a convertirme en lo que soy hoy, gracias por estar ahí para mí.

Dedicatoria

De Gustavo Cosio Martin.

A mis queridísimos padres por haberme guiado siempre por el mejor camino, por ser cada día una fuerza más para lograr todo lo que logrado y triunfar en las peores situaciones, gracias por confiar en mi, gracias por estar siempre juntos apoyándome, gracias por ser mis padres.

A mis adorados y nunca olvidados abuelos, Mima y Pupú, por haber sido siempre mis defensores y mas complacientes abuelos, su cariño nunca se desprenderá de mis sentimiento, gracias por haber existidos.

A mi abuelo Carlos por haberme permitido escuchar sus historias, aprender de sus diversas formas de trabajar, de prepararse para cada tarea que se nos imponga en el trabajo profesional y de ahorrar en tiempos difíciles, también a mi Abuela Elida por mostrar siempre su mejor carácter y su vida tan llena de fuerzas para disfrutar cada instante que la vida le regala.

A una persona en especial que me ha acompañado en esta gran batalla.

A muchos compañeros y amigos que nunca me dijeron que no cuando realmente me hacia falta su apoyo, en fin gracias por su apoyo incondicional y por confiar en mi como una gran amigo.

Resumen

Hoy en día constituye una necesidad de primer orden para las instituciones científicas disponer de aplicaciones computacionales que faciliten el proceso de edición y resolución de textos matemáticos. En este contexto, como parte del proceso de informatización de nuestro país, de conjunto con el Polo de Bioinformática de la Universidad de las Ciencias Informáticas (UCI) y el Centro de Inmunología Molecular (CIM) se creó la Plataforma de Simulación y Análisis de Sistemas Biológicos (BioSyS) la cual posee un editor de ecuaciones para facilitar las tareas de edición de los modelos matemáticos a los investigadores de estas instituciones. Una característica indispensable para este tipo de herramienta computacional es que deben contar con funcionalidades que permitan procesar textos matemáticos codificados en los diferentes estándares de representación matemática. Actualmente el Editor de Ecuaciones de la Plataforma BioSyS sólo dispone de funcionalidades para la exportación de archivos en formato MathML Presentation, MathML Content e interno por lo que el presente trabajo tiene como objetivo la implementación de funcionalidades que permitan llevar a cabo el proceso de importación de archivos con los formatos MathML Presentation, MathML Content e Interno.

Palabras Claves: Bioinformática, modelo matemático, sistemas biológicos, editor de ecuaciones.

Abstract

Today, computing applications are a first order need for scientific institutions; taking into consideration they facilitate the edition process and the resolution of mathematical texts. So, as part of our country informatization process, together with the Informatics Science University Bioinformatics Pole and the Molecular Immunology Center, a Simulation and Analysis of Biological Systems Platform (BioSyS) was created. This has an equations editor to facilitate the edition of mathematical models for these institutions investigators. One of the most important features of this computing tool is to have functionalities that allow processing codified mathematical texts in the different standards of mathematical representation. Today the Equations Editor of BioSyS Platform only has functionalities to export files on MathML Presentation, MathML Content and Intern format; that is why our paper is addressed to the implementation of functionalities which allow importing files on MathML Presentation, MathML Content and Intern format.

Tabla de Contenidos

Índice de figuras

Índice de tablas

Introducción

El ser humano siempre ha necesitado medios para hacer cálculos y procesar la información. La complejidad de estos, subordinada a los progresos de la tecnología, se fue acrecentando en el transcurso del tiempo conforme surgían nuevas necesidades; lo que propició la creación de lo que hoy conocemos como ordenador moderno y con ello el surgimiento de una nueva ciencia interdisciplinaria "La Informática", que comprende el estudio de las ciencias de la computación y el manejo y procesamiento de la información. [1]

Actualmente la informática es la tecnología central y ciertamente la más importante que a lo largo del pasado siglo, revolucionó modos de acción y pensamiento de sociedades, haciendo posible lo que hasta hace poco era impensable.

La utilización de la informática se va volviendo algo cada vez más usual e indispensable en el mundo actual, es prácticamente imposible concebir una actividad humana en la cual no esté presente; en una u otra medida se ha convertido en parte habitual de la vida, ya sea cuando se trabaja, se aprende, se juega o se descansa. [2]

Las matemáticas constituyen un pilar indispensable para el desarrollo de la Informática como ciencia, pues diversos fundamentos matemáticos sustentan muchas áreas de sus campos de aplicación y resulta evidente que no se puede dejar de mencionar disciplinas como la criptografía, teoría de grafos, lógica computacional, algoritmización y estructuras de datos, inteligencia artificial, teoría de los tipos, procesamiento de lenguajes y gráficos por computadoras; de fuerte contenido matemático y en las que los principios matemáticos constituyen sus bases existenciales.

Es importante el papel que juegan las matemáticas en el modelado de sistemas, ya sean biológicos o físicos, para la simulación de los mismos utilizando dispositivos de cómputo. La adecuada descripción de un fenómeno científico del cual se tiene abundante información experimental en un marco matemático permite el uso de poderosas herramientas informáticas para la construcción de algoritmos efectivos para su caracterización, análisis y su predicción; además los modelos matemáticos permiten realizar experimentos virtuales cuyos análogos reales serían caros, peligrosos o imposibles.
En la actualidad es necesidad de primer orden para las comunidades científicas e instituciones investigativas disponer de potentes y capaces herramientas informáticas que puedan llevar a cabo el

proceso de escribir o editar textos matemáticos de forma entendible con el objetivo de su posterior análisis, resolución o simulación. En este contexto se crearon los editores matemáticos como aplicaciones computacionales encargadas de formatear y alinear correctamente los símbolos de las fórmulas matemáticas. Actualmente son muchas las herramientas de edición de fórmulas matemáticas, pero no se puede dejar de mencionar a software como MathType, WebEQ, OpenOffice o Mathematica ampliamente difundidos y estandarizados para la representación de textos matemáticos.

Para contribuir al desarrollo que viene experimentando la industria del software en nuestro país, elevar el nivel científico de nuestras instituciones investigativas y ahorrar recursos económicos por conceptos de compra de productos informáticos en mercados extranjeros; de conjunto con el Centro de Inmunología Molecular(CIM) y el polo de Bioinformática de la Universidad de la Ciencias Informáticas (UCI), se dispuso la creación de la Plataforma para la Simulación y Análisis de Sistemas Biológicos (BioSyS) para dotar a los investigadores de estos centros científicos de una herramienta computacional para el análisis y estudio de los sistemas biológicos. Actualmente esta Plataforma cuenta con un editor de ecuaciones creado por parte de su equipo de desarrollo con el objetivo de facilitar al usuario, en este caso al investigador, el proceso de edición de los modelos matemáticos que desea ingresar para su posterior simulación.

Una característica que es imprescindible para este tipo de herramienta computacional es que deben disponer de funcionalidades tanto para exportar como para importar textos matemáticos codificados en los diferentes estándares de representación matemática existentes como TeX, LaTeX o MathML. Actualmente el Editor de Ecuaciones de la Plataforma **BioSyS** solo cuenta con funcionalidades para exportar los modelos matemáticos editados en formato MathML Presentation, MathML Content e Interno, lo que nos lleva a plantear el siguiente **problema científico** a resolver por el presente trabajo:

El editor de ecuaciones de la Plataforma de Simulación y Análisis de Sistemas Biológicos no cuenta con funcionalidades para la importación de archivos en formato MathML Presentation, MathML Content e Interno.

Se ha identificado como **objeto de estudio:**
Proceso de importación de archivos.

Y se ha delimitado dentro de esta área el **campo de acción** a**:**

Proceso de importación de archivos en formato MathML Presentation, MathML Content y Formato Interno.

Para dar cumplimiento al problema planteado se tiene como **objetivo general:**
Implementar funcionalidades para la importación de archivos en formato MathML Presentation, MathML Content e Interno para el Editor de Ecuaciones de la Plataforma de Simulación y Análisis de Sistemas Biológicos **BioSyS**.

Del objetivo general se han desglosado los siguientes **objetivos específicos:**

- Diseñar funcionalidad para la importación de archivos en formato MathML Presentation.
- Diseñar funcionalidad para la importación de archivos en formato MathML Content.
- Diseñar funcionalidad para la importación de archivos en formato Interno.
- Implementar funcionalidad para la importación de archivos en formato MathML Presentation.
- Implementar funcionalidad para la importación de archivos en formato MathML Content.
- Implementar funcionalidad para la importación de archivos en formato interno.
- Probar funcionalidad para la importación de archivos en formato MathML Presentation.
- Probar funcionalidad para la importación de archivos en formato MathML Content.
- Probar funcionalidad para la importación de archivos en Formato Interno.

Para lograr las metas planteadas se le darán cumplimiento a las siguientes **tareas:**

- Búsqueda bibliográfica sobre intérpretes y compiladores descendentes recursivos.
- Búsqueda y estudio de los estándares de representación matemática MathML Presentation, MathML Content y formato interno.
- Búsqueda y estudio de herramientas existentes que realicen importación de archivos en formato MathML Presentation y MathML Content.
- Confección de la gramática para intérprete de MathML Presentación.
- Confección de la gramática para intérprete de MathML Content.
- Confección de la gramática para intérprete de formato interno.
- Implementación de los analizadores léxicos y sintácticos para el lenguaje MathML Presentación.
- Implementación de los analizadores léxicos y sintácticos para el lenguaje MathML Content.
- Implementación de los analizadores léxicos y sintácticos para el lenguaje Formato Interno.

- Generación de las expresiones matemáticas a partir del árbol de sintaxis abstracta generado por los analizadores sintácticos.
- Diseño de los casos de prueba.
- Ejecución de los casos de prueba.
- Evaluación de los resultados de los casos de pruebas diseñados.

El presente trabajo de diploma está estructurado en cuatro capítulos y varios anexos. A continuación se da un breve resumen de los capítulos.

Capítulo 1 “Fundamentación teórica”: Donde se brinda una descripción general del análisis de algunos software existentes que permiten llevar a cabo el proceso de importación de archivos en formato MathML, se describen los tipos de traductores a utilizar, las características de las gramáticas; así como las herramientas, lenguajes de programación y entorno de desarrollo a utilizar.

Capítulo 2 “Características de la funcionalidad”: Donde se da una breve descripción de la funcionalidad a desarrollar, se describen los requisitos funcionales y no funcionales de la funcionalidad, se definen los actores del sistema, el diagrama de caso de uso del sistema y se hace una descripción detallada de los casos de usos del sistema.

Capítulo 3 “Diseño de la funcionalidad”: Donde se da una descripción del estilo arquitectónico utilizado para el desarrollo de la funcionalidad, se describirán los patrones de diseño empleados así como los diagramas de clases del diseño.

Capítulo 4 “Implementación y pruebas”: Donde se describe como fue implementada la herramienta en términos de componentes. Se desarrollarán varias pruebas de aceptación usando el método de caja negra para garantizar que se han cumplido los requisitos funcionales.

Capítulo 1. Fundamentación teórica

En este capítulo se realizará un análisis de las principales herramientas que realizan el proceso de importación de código MathML, además se describirán las características de este estándar de representación matemática. En esta sección se presta especial atención a la descripción del tipo de traductor a utilizar para desarrollar la funcionalidad de importación, así como el tipo de gramática que estos implementarán para efectuar el correcto procesamiento de los lenguajes; también se describen las tecnologías y tendencias actuales consideradas para implementar la funcionalidad.

1.1 Codificación de la Información Matemática

La forma de codificar la información matemática ha evolucionado notablemente hasta nuestros días. En los primeros tiempos el hombre sólo disponía de sistemas numéricos rudimentarios los que resultaban muy difíciles de entender cuando se quería transmitir alguna idea relacionada con planteamientos matemáticos. Con el paso del tiempo, las comunidades científicas de todo el mundo vieron la necesidad de la creación de una forma de representar los textos matemáticos de manera que pudieran ser entendibles por todos, es en este contexto que fueron creados los estándares de representación matemática.

Inicialmente la información matemática se escribía en formato **ASCII**, pero este formato es demasiado limitado. Posteriormente, en 1986, apareció **TeX**, un lenguaje desarrollado por Donald Knuth que se volvió muy popular y es usado ampliamente hasta la actualidad. **TeX** fue la influencia más grande para el surgimiento de otros estándares de representación matemática como **LaTeX** y **MathML**. [3]

1.1.1 MathML

MathML (Mathematical Markup Language), es un lenguaje de marcado para describir expresiones matemáticas y fue lanzado originalmente como recomendación por la W3C el 7 de abril de 1998 para superar las limitaciones que en este sentido poseía HTML. Actualmente se encuentra en su versión 2.0 y su objetivo principal es permitir que las matemáticas sean enviadas, recibidas y procesadas en la Web al igual que lo es el texto HTML.

MathML es una extensión de XML y puede ser considerado un módulo de XHTML. Tal como sucede con los editores de páginas Web, actualmente existen editores de expresiones que permiten crear el código MathML sin que el usuario tenga que involucrarse con este. Algunos de los editores de MathML son Amaya, EzMath, Maple, Mathematica, OpenOffice, MathType y WebEQ. [3]

MathML fue creado para el uso de la comunidad científica, que lo utilizaría en la difusión de sus pensamientos y teorías. Entre sus metas podemos citar:

- Codificar material matemático útil para la enseñanza y la comunicación científica a todo nivel.
- Codificar tanto notación matemática como significado matemático.
- Facilitar conversión desde y hacia otros formatos de presentación (por ejemplo: **TeX**).
- Conveniente para la interacción con software externo. Esto se refiere en particular a generadores de código o posibles intérpretes e inclusive evaluadores de expresiones.
- Ser extensible.

Una de las características más importantes de este lenguaje de marcado es que permite codificar la información matemática de dos formas, la primera corresponde a los elementos de presentación que describe la notación del objeto matemático y la segunda corresponde a elementos de contenido que describe la estructura matemática del objeto matemático. A continuación se detallan cada una de estas formas de representación.

1.1.1.1 Elementos de presentación (MathML Presentation)

Corresponden a "constructores" de la notación matemática tradicional, es decir, los tipos básicos de símbolos y estructuras para la construcción de expresiones, a partir de los cuales cualquier parte de la notación tradicional de la matemática puede generarse. La descripción de las estructuras notacionales que los elementos representan se dan generalmente de una manera visualmente orientada. Los elementos de presentación sólo sugieren (no requieren) maneras específicas de representar, de forma de permitir una representación independiente del medio y para preferencias individuales de estilo.

Los símbolos matemáticos deben ser representados por medio de elementos de MathML. Los principales (<mi>x</mi>) para identificadores, números (<mn>98</mn> y operadores (<mo>+</mo>). Dentro de estos tenemos por ejemplo a <mrow>...</mrow>, que permite que se despliegue en una fila la información dentro de los tags y la <mfrac>...</mfrac>, que permite denotar fracciones.

Es importante notar que en presentación el orden de los esquemas hijos si importa, pero no está forzado a nivel de la **DTD** de MathML. Por ejemplo para la expresión **'a - db'** sería representado como: (Ver anexo 1).

1.1.1.2 Elementos de Contenido (MathML Content)

La intención fundamental de la codificación de contenido en MathML es proveer una codificación específica de la estructura matemática subyacente de una expresión, más allá de cualquier

representación particular para la expresión. La principal razón para proveer esta codificación es que aún con el uso sistemático de tags o etiquetas de presentación no se puede capturar la información semántica entregada por este sistema. Al codificar la información de esta forma se puede asegurar un intercambio de información mucho más preciso, incluso permitiendo la evaluación de las expresiones de una forma simple.

El objetivo básico de la codificación de contenido es el establecimiento de relaciones explicitas entre las estructuras matemáticas y sus significados matemáticos. Cada estructura tiene una semántica predeterminadamente asociada y existe un mecanismo para asociar nuevos significados matemáticos con nuevas construcciones.

La codificación de contenido de MathML esta basada en el concepto de árbol de expresión. En este árbol las hojas corresponden a objetos matemáticos básicos como son números (<cn>63</cn>), variables (<ci>p</ci>), <apply>...</apply> es quizás uno de los más importantes, ya que es el que se usa para efectivamente aplicar la función a sus argumentos o para agrupar elementos, etc. Los nodos intermedios generalmente representan algún tipo de función (<power/>) u otra construcción matemática que crea un objeto compuesto.

Por ejemplo para la expresión **'a - db'** sería representado según esta notación como: **(Ver anexo 2)**.

1.1.2 Lenguaje de descripción Formato Interno

Este lenguaje de descripción de expresiones matemática fue creado por parte del equipo de desarrollo del editor de ecuaciones bajo la conducción principal del Máster en ciencias Gilberto Arias Naranjo. Entre sus características más notables resalta el uso de operadores semánticos para la descripción de las operaciones matemáticas. La ventaja fundamental de este formato es que sólo describe aquellas expresiones con que cuenta el editor de ecuaciones, esto lo convierte en algo sencillo y más fácil de interpretar y usar que el estándar MathML. El objetivo principal de este formato es facilitar la comunicación entre los componentes de la Plataforma BioSyS cuando estos requieran del el uso de las fórmulas matemáticas y además facilitar el proceso de almacenamiento de los modelos matemáticos en la base de datos. Para consultar algunos ejemplos de la forma en que se codifican las expresiones matemáticas con este lenguaje de descripción matemático consulte el anexo 3.

1.2 Herramientas para la importación de código MathML

A continuación se describen algunas de las principales características de las herramientas existentes que llevan a cabo el proceso de importación de archivos con código MathML y se resaltan sus ventajas y el por qué de sus limitaciones de uso por parte del equipo que desarrolla el trabajo.

1.2.1 Abiword-plugin-abimathview

Es un plugin cuya última versión es la 2.6.4, agregable al procesador de texto abiword para la distribución Mandrake de GNU/Linux. Esta herramienta está diseñada para ofrecer la capacidad al abiword de importar y editar documentos o ficheros de código MathML. Es una herramienta libre y posee un repositorio amplio de versiones liberadas que se pueden descargar fácilmente de la siguiente dirección:

http://www.filewatcher.com/abiword-plugin-abimathview.html.

Posee la limitante de que sólo maneja código de elementos de MathML Presentation. Su principal inconveniente es que no es un plugin integrable, o sea, sólo puede ser utilizado como complemento del abiword para brindarle una nueva funcionalidad, además su proceso de compilación reporta numerosos errores, pues sus bibliotecas entran en conflicto con entornos que no sean Mandrake, como distribución de Linux.

1.2.2 GtkMathView

Es una herramienta libre, desarrollada con C++, para la visualización de documentos o ficheros MathML. Proporciona una vista interactiva en la que el usuario puede conocer e interactuar en todo momento con el código MathML de la expresión matemática que está siendo editada. Sus versiones pueden ser descargadas del sitio oficial: **http://helm.cs.unibo.it/mml-widget.html**

Necesita de las siguientes bibliotecas para su correcto funcionamiento glib, cuya versión instalada sea superior a la 2.2; libxml2, versión instalada superior a la 2.0.0 y gmetadom, versión instalada superior a la 0.1.8.

Posee la limitante de que sólo maneja código MathML Presentation y no integrable a la plataforma de trabajo BioSyS porque su arquitectura no es compatible con la arquitectura del editor de ecuaciones.

1.2.3 Formulator v3.8

Aplicación perteneciente a la organización Hermitech Laboratory, desarrollado por Andriy Kovalchuk, Vyacheslav Levitsky, Igor Samolyuk, Valentyn Yanchuk. Sus versiones pueden ser descargadas de su sitio oficial:

http://www. Hermitech.ic.zt.ua/projects/Formulator/index.html.

Es un editor de ecuaciones matemáticas dinámico, diseñado para ejecutarse en ordenadores personales que dispongan de la plataforma de Microsoft Windows. Esta aplicación permite a los usuarios crear cualquier tipo de ecuación de forma muy simple. Permite convertir las ecuaciones

editadas a formato MathML, salvarlas en modo gráfico o cualquier otro formato, listos para ser importados posteriormente.

Formulator es una aplicación ampliamente extensible pues no solo soporta formato MathML sino que hace uso para exportar e importar ficheros de código matemáticos estandarizados y aprobados por la W3C. La aplicación provee un grupo de etiquetas y cajas de textos que representan a los elementos básicos de modelación matemática en la que los usuarios con solo pulsar en la etiqueta deseada pueden ir agregando e insertando los símbolos a las ecuaciones que estén editando. Su principal inconveniente es que es un software propietario y aunque realiza el proceso de importación de archivos con formato MathML Presentation, no se dispone del código fuente de esta funcionalidad. Esta herramienta no es integrable a la plataforma de BioSyS por las características antes mencionadas.

1.2.4 MathML.NET Control

Es el primer y único producto comercial disponible del framework .NET para la edición de ecuaciones matemáticas con el objetivo de exportarlas e importarlas en formato MathML para que puedan ser visualizadas en los navegadores web como el Internet Explorer. Está al alcance de todos los usuarios que van desde estudiantes y profesores hasta científicos altamente calificados. **MathML.NET Control** proporciona una interfaz muy fácil de usuario que permite crear todo tipo imaginable de expresiones matemáticas con tan solo pulsar un click en los diferentes elementos de la gran variedad de paletas de componentes que posee para la edición (ver figura 1). Cada fórmula puede ser guardada como una imagen JPEG o exportada a un mapa de bits con alto poder de la resolución de archivo.

Fig. 1: Barras de símbolos matemáticos de MathML.NET Control.

Escrito el 100% con código C #, brinda a los desarrolladores una forma rápida y sencilla de exportar e importar las ecuaciones editadas en formato MathML, en este caso en archivos de extensión mml o xml. Esta herramienta posee la limitante de que es comercializable y por lo tanto se necesita pagar la licencia para su uso, además no se puede integrar a entornos Unix, en nuestro caso a la Plataforma BioSyS. No es multiplataforma, sólo trabaja con elementos de MathML Presentation, no da soporte a otro tipo de codificación como MathML Content y aunque dispone de la funcionalidad para la importación de archivos con código MathML Presentation no se dispone del código fuente de la misma.

1.3 Tipo de traductor a desarrollar

Un traductor se define como un programa que traduce o convierte desde un texto o programa escrito en un lenguaje fuente hasta un texto o programa equivalente escrito en un lenguaje destino produciendo, si cabe, mensajes de error. Los traductores engloban tanto a los compiladores como a los intérpretes.

Un compilador es un programa que permite traducir el código fuente de un programa en lenguaje de alto nivel, a otro lenguaje de nivel inferior (típicamente lenguaje máquina), es decir, es un programa que proporciona un fichero objeto en lugar del ejecutable final. Un intérprete, a diferencia de un compilador, reconoce al programa de entrada y lo ejecuta a la vez, es decir, no produce un resultado físico (código máquina) sino lógico (una ejecución). [4]

Luego de establecer las principales diferencias entre estos tipos de traductores se acordó que el más idóneo para resolver el problema que tiene planteado el presente trabajo se corresponde al prototipo de un intérprete pues a medida que se vaya procesando el código fuente de los archivos es necesario decidir que acción ejecutar, o sea, lo que se lleva a cabo es una ejecución de las instrucciones contenidas en los archivos.

1.4 Tipo de analizador sintáctico a desarrollar

El análisis sintáctico (parser en lengua inglesa) toma los **tokens** que le envía el analizador léxico y comprueba si con ellos se puede formar alguna sentencia válida del lenguaje, auxiliándose para ello generalmente de gramáticas libres de contextos que contienen un grupo de reglas definidas en notación **EBNF** para describir las sentencias válidas de dicho lenguaje. Existen tres tipos generales de analizadores sintácticos:

- **Analizadores Universales.**

Procesan cualquier tipo de gramáticas libres de contextos utilizando el algoritmo CYK introducido en 1963. Este algoritmo no se considera demasiado eficiente para ser utilizado de forma generalizada ya que su coste es exponencial (dependen de la longitud de la cadena de entrada). [4]

- **Analizadores Descendentes (top-down).**

Construyen el árbol sintáctico de la sentencia a reconocer desde el símbolo inicial (raíz), hasta llegar a los símbolos terminales (hojas) que forman la sentencia, usando derivaciones más a la izquierda.
Los principales problemas que se plantean son dos, el retroceso (backtraking) y la recursividad a izquierdas. Para solventar este inconveniente, se opera con gramáticas LL (k), que pueden realizar un análisis sin retroceso en forma descendente. [4]

- **Analizadores Ascendentes (bottom-up).**

Construyen el árbol sintáctico de la sentencia a reconocer desde las hojas hasta llegar a la raíz. Son analizadores del tipo reducción-desplazamiento (shift-reduce), parten de los distintos **tokens** de la sentencia a analizar y por medio de reducciones llegan al símbolo inicial de la gramática. El principal problema que se plantea en el análisis ascendente es el retroceso. Para solventar este inconveniente, se definieron distintos tipos de gramáticas entre las cuales las más utilizadas son las **LR (k)** ya que realizan eficientemente el análisis ascendente sin retroceso. [4]

Después de estudiar las características de cada tipo de analizador se acordó desarrollar un analizador sintáctico descendente por las siguientes razones:

- ✓ **Sencillez de implementación**: los analizadores descendentes son sencillos de implementar manualmente pues hacen uso de gramáticas de tipo (LL); los analizadores ascendentes son complejos y requieren el uso de herramientas especiales de procesamientos como autómatas; además hacen uso de gramáticas (LR).
- ✓ **Sencillez de funcionamiento (depuración):** El proceso descendente (derivaciones por la izquierda) es más intuitivo y fácil de depurar que el ascendente (en orden inverso, y con derivaciones por la derecha).
- ✓ **Reglas semánticas:** Los esquemas de traducción, gramáticas atribuidas y definiciones dirigidas por sintaxis de gramáticas (LR) que emplean los analizadores ascendentes emplean atributos sintetizados; las que emplean (LL) en este caso los analizadores descendentes ofrecen tanto sintetizados como heredados.

- ✓ **Recuperación de errores:** Los analizadores (LL) poseen información de las construcciones que se están reconociendo, mientas que los (LR) poseen información de aquéllas que ya están reconocidas; la información de los analizadores (LL) es más útil para recuperar los posibles errores.
- ✓ **Tamaño de las tablas de análisis:** los compiladores (LR) requieren tablas de aproximadamente el doble tamaño que los compiladores (LL). [4]

Unido a las características anteriores puede agregarse que los analizadores sintácticos descendentes o también llamados predictivos o orientados hacia un fin operan con gramáticas LL (k), que pueden realizar un análisis sin retroceso en forma descendente y permiten elegir correctamente la producción correspondiente a cada no terminal que se expande.

Es decir, el análisis descendente es determinista, pues sólo deja tomar una opción en la expansión de cada no terminal. Además este tipo de análisis da lugar a los analizadores descendentes recursivos los cuales ejecutan un conjunto de procedimientos recursivos para procesar la entrada, donde cada no terminal en la gramática tiene asociado un procedimiento recursivo y para cada **token** (lexema en español) en la cadena de entrada solo es posible reducir por una regla. Solo es válido aclarar que la condición necesaria para que un analizador recursivo descendente opere correctamente es que la gramática del lenguaje fuente sea LL (1). Es por ello que nuestro principal objetivo es desarrollar un analizador de este tipo por las características antes descritas.

1.5 Tipo de gramática seleccionada

La gramática es un ente o modelo matemático que permite especificar un lenguaje, es decir, es el conjunto de reglas capaces de generar todas las posibilidades combinatorias de ese lenguaje, y sólo las de dicho lenguaje, ya sea éste un lenguaje formal o un lenguaje natural. [4]

Una gramática define a un lenguaje en forma recursiva y generalmente es un cuádruplo G = {N, ∑, P, S} donde:

- N: Es el conjunto de símbolos no terminales los cuales permiten representar combinaciones de símbolos.
- ∑: Es el conjunto de símbolos terminales que se corresponde con el concepto de alfabeto.
- P: Es el conjunto de Reglas de Producción.
- S: Axioma o símbolo distinguido.

Como se mencionó en el epígrafe anterior el tipo de analizador a desarrollar exige como única condición que el lenguaje fuente sea generado por gramáticas de tipo LL (k), es por ello que este tipo de gramática fue la seleccionada para generar los lenguajes. A continuación en el siguiente epígrafe se describen las características de este tipo de gramática.

1.5.1 Gramáticas LL (K)

Las gramáticas LL (k) son un subconjunto de las gramáticas libres de contexto. Permiten un análisis descendente determinista (o sin retroceso), por medio del reconocimiento de la cadena de entrada de izquierda a derecha ("Left to right") y que va tomando las derivaciones más hacia la izquierda ("Leftmost") con sólo mirar los (k) tokens situados a continuación, además las gramáticas LL (k) dan lugar a analizadores descendentes, mientras que los analizadores que utilizan la técnica (LR) dan lugar a analizadores ascendentes. Si k=1 se habla de gramáticas LL (1). [4]

La condición necesaria y suficiente para que una gramática sea LL (1), es que los símbolos directores correspondientes a las diferentes expansiones de cada símbolo no terminal sean conjuntos disjuntos. Dicho de otra manera una gramática es LL (1) si, para cualquier par de producciones **(A → α y A → β)**, se cumple que:

Primeros (α· siguientes(A)) ∩ Primeros (β· siguientes(A)) =Ø

Es decir, que no existen dos axiomas distintos que comiencen con el mismo símbolo. En general, podemos decir que una gramática LL (1) es aquélla en la que es suficiente con examinar sólo un símbolo a la entrada, para saber qué regla aplicar, partiendo del axioma inicial y realizando derivaciones a la izquierda. [4]

Esta condición garantiza que un símbolo no puede aparecer en dos conjuntos de símbolos directores, pues sino el analizador sintáctico descendente no puede decidir (sin recurrir a información posterior) que expansión aplicar; además la condición es suficiente, puesto que el analizador siempre puede escoger una expansión como un símbolo dado, y esta alternativa será siempre correcta.

1.6 Tendencia y Técnica de los Roles

A continuación se mencionan los roles desempeñados por el equipo de desarrollo, los artefactos que se generaron y los distintos tipos de patrones utilizados.

1.6.1 Roles

Los roles desempeñados según los que establece la metodología **OpenUP** (**ver anexo 4**), fueron:

- Analista.

- Desarrollador.
- Arquitecto.

1.6.2 Artefactos

La metodología de desarrollo **OpenUP** propone una lista de artefactos desarrollados por los roles anteriormente mencionados **(Anexo 4)**, pero no todos serán utilizados en el presente trabajo, a continuación se mencionarán los artefactos que serán desarrollados por los roles y lo que realmente son de nuestro interés.

Analista realiza:

- Glosario.
- Requerimientos de Soporte.
- Modelo de Casos de Uso.
- Casos de Uso.

Desarrollador realiza:

- Build.
- Diseño.
- Implementación.
- Prueba de desarrollador.

Arquitecto realiza:

- Documento de Arquitectura.

1.6.3 Patrones

Buschman (1996) define Patrón como una regla que consta de tres partes, la cual expresa una relación entre un Contexto, un Problema y la Solución. Un Patrón sigue el siguiente esquema:

- Contexto: Es una situación de diseño en la que aparece un problema de diseño.
- Problema: Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- Solución: Es una configuración que equilibra la estructura con componentes y relaciones con el comportamiento a tiempo de ejecución. [5]

1.6.3.1 Patrones de caso de uso

Los patrones de casos de uso se presentan a modo de herramientas que permiten resolver los problemas que se les planteen a los desarrolladores de una forma ágil y sistemática. Estos patrones se enfocan hacia el diseño y las técnicas utilizadas en modelos de alta calidad, y no en cómo modelar usos específicos, además permiten con más precisión reflejar los requisitos reales, haciendo más fácil el trabajo con los sistemas, y mucho más simple su mantenimiento. [6]

A continuación se mencionan los patrones de caso de usos existentes:

- Reglas de negocio
- Concordancia (Commonality)
- Componente jerárquico
- Extensión concreta o Inclusión
- CRUD (Creating, Reading, Updating, Deleting)
- Caso de uso grande (Large Use case)
- Sistema de Capas
- Múltiples actores
- Servicio opcional
- Vistas ortogonales
- Secuencia de casos de uso

De ellos, el equipo de desarrollo determinó sólo utilizar el patrón **concordancia.**

1.6.3.2 Patrones de Arquitectura

La selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo de un sistema de software. Existen muchos patrones de arquitectura, pero aquí sólo mencionaremos algunos de los más interesantes. Los sistemas empresariales distribuidos pueden agrupar los siguientes estilos arquitectónicos:

- Modelo-Vista-Controlador (MVC)
- Arquitecturas en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes
- Arquitecturas Orientadas a Servicios

Para la implementación de nuestra aplicación, teniendo en cuenta las características de estos patrones, se utilizará de los antes mencionados solamente, el **Modelo-Vista-Controlador**, que se explicará más adelante.

1.6.3.3 Patrones de Diseño

Los patrones de diseño “Son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan”. [7]

En el desarrollo de nuestra aplicación se emplean los patrones de diseño siguientes:

- **Patrones GRASP(Patrones de los Principios Generales para Asignar Responsabilidades)**
  - Experto
  - Creador
  - Bajo acoplamiento
  - Alta cohesión

- **Patrones GoF,** derivados en tres tipos de patrones dentro de los cuales especificamos los patrones utilizados en la siguiente tabla.

Estructurales	De Comportamiento
Mediador	Observador
	Intérprete

Tabla 1: Patrones (GoF) utilizados según la clasificación.

1.7 Tecnologías y herramientas a utilizar

Uno de los aspectos fundamentales en la elaboración de un software es seleccionar las tecnologías y herramientas que mayores beneficios aporten. Para llevar a cabo la implementación de este trabajo, se realizó un estudio de las mismas, teniéndose en cuenta las tendencias actuales, novedades y facilidades de cada una de ellas para llegar a la selección de las que se enuncian a continuación.

1.7.1 Metodología de desarrollo a utilizar

Una metodología de desarrollo de software ofrece un conjunto de técnicas y procedimientos que permiten conocer la organización de los elementos necesarios para definir un proyecto de software y se debe seleccionar cuál metodología utilizar, teniendo en cuenta, la que se adapta más al medio en que se desarrolla y a los objetivos finales que queremos cumplir.

Se decidió aplicar el **Proceso Unificado Abierto** (**OpenUP** por sus siglas en ingles) como metodología de desarrollo para este proyecto porque con **OpenUP** se logra una mayor productividad por parte del

equipo de trabajo, ya que se definen claramente actividades, roles y sus responsabilidades. **OpenUP** sostiene las mismas características del **Proceso Unificado Racional** (**RUP** por sus siglas en inglés), el desarrollo es iterativo e incremental, es guiado por casos de uso, permite gestionar riesgos, y su enfoque es centrado en la arquitectura.

La metodología plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo corto; utiliza solo los procesos que sean necesarios y para ello necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no es necesario para que el proyecto no tenga errores; propone no utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo ser este modificado, mejorado y extendido. [8]

Por estas razones se eligió **OpenUP** como la metodología de desarrollo para el presente trabajo, además los desarrolladores de la Plataforma de Simulación de Sistemas Biológicos (**BioSyS**) del Polo de Bioinformática tienen establecido entre sus políticas de desarrollo la adopción de esta metodología.

1.7.2 Lenguaje Unificado de Modelado (UML)

El **Lenguaje Unificado de Modelado (UML)** es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema. **UML** "se usa para entender, diseñar, hojear, configurar, mantener, y controlar la información sobre tales sistemas". [9]

UML está compuesto por diversos elementos gráficos que se combinan para conformar diagramas. Debido a que el **UML** es un lenguaje, cuenta con reglas para combinar tales elementos. Es importante recalcar que **UML** no es una guía para realizar el análisis y diseño orientado a objetos, es decir, no es un proceso. **UML** es un lenguaje que permite la modelación de sistemas con tecnología orientada a objetos. En lo que corresponde al desarrollo de programas, posee elementos gráficos para soportar la captura de requisitos, el análisis, el diseño, la implementación, y las pruebas. **UML** describe lo que supuestamente hará un sistema, pero no dice cómo implementarlo, sus objetivos son ofrecer un lenguaje simple y legible que permitiera modelar aplicaciones en cualquier dominio, y generar de manera automática código fuente. **UML** permite que diseñadores diferentes modelen sistemas diferentes y puedan ampliamente entender cada uno los diseños de los otros. Pretende dar apoyo a la mayoría de los procesos de desarrollo orientados a objetos. [9]

UML es un lenguaje capaz de abstraer cualquier tipo de sistema (informático o no), mediante los diagramas, esto es, mediante representaciones gráficas que contienen toda la información relevante del sistema. Para lograrlo, utiliza distintos tipos de diagramas, como los diagramas de implementación,

diagramas de comportamiento o interacción, diagramas de casos de uso y diagramas de clases. Las principales funciones de **UML** son:

- **Visualizar: UML** permite expresar de una forma gráfica un sistema de forma que otra persona lo puede entender.
- **Especificar: UML** permite especificar cuáles son las características de un sistema antes de su construcción.
- **Construir:** A partir de los modelos especificados se puede construir los sistemas diseñados.
- **Documentar:** Los propios elementos gráficos son parte de la documentación del sistema desarrollado, sirviendo para su futura revisión.

UML representa para los desarrolladores de aplicaciones y sistemas una serie de ventajas, al igual que para las organizaciones, entre estos beneficios se destacan:

- Produce un aumento en la calidad del desarrollo.
- Mejora en un 50% o más los tiempos totales de desarrollo.
- Permite especificar la estructura y el comportamiento del sistema.
- Permite dimensionar mejor los riesgos de un proyecto, tener un mejor rendimiento antes de construir el sistema.
- Facilita la documentación de las decisiones de la arquitectura del proyecto.
- Ofrece un mejor soporte a la planificación y control del proyecto.
- Ofrece mayor rigurosidad en la especificación.
- Permite realizar una verificación y validación del modelo realizado.
- Se pueden automatizar determinados procesos y permite generar código a partir de los modelos y viceversa.
- Soportado por muchas herramientas.

1.7.3 Herramienta CASE

Debido a la popularidad que ha ido adquiriendo UML en los últimos años, existen una gran variedad de herramientas CASE (Computer-Aided Software Engineering / Ingeniería de Software asistida por computadoras) que facilitan al ingeniero el modelado de cualquier proceso de desarrollo. 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.

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.

1.7.4 Lenguaje de programación (Java)

Un lenguaje de programación puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora. Se clasifican según el nivel de abstracción (bajo, medio y alto), según la forma de ejecución (compilado e interpretado) y según el paradigma de programación (imperativo, funcional, lógico, orientado a objeto) que poseen cada uno de ellos.

Para desarrollar el presente trabajo se ha decidido utilizar el lenguaje de programación **Java** siendo este uno de los lenguajes de programación de alto nivel y uno de los más populares del mundo, creado por la compañía SunMicrosystems en 1991. Algunas de las características fundamentales de **Java** son:

- ✓ **Distribuido:** desde su comienzo en 1982, la visión de SunMicrosystems ha sido "la red es el ordenador". La idea inicial era lograr un lenguaje que reconociera la red y admitiera el acceso a objetos distribuidos de forma simple. Actualmente **Java** puede invocar métodos en un equipo remoto de manera sencilla y transparente, utilizando protocolos comunes como CORBA (Arquitectura intermediaria para solicitar objetos comunes) y RMI (Invocación de métodos remotos), además de los servicios Web, como Servlets, Portlets, etc.
- ✓ **Interpretado:** Los programas en Java son interpretados. En vez de ser compilados a código de máquina nativo, el código de Java es traducido a código de bytes (o bytecode, en inglés). Esto le permite que el código sea ejecutable en cualquier plataforma que tenga una Máquina Virtual de Java (JVM)* sin necesidad de ser recompilado. Otra ventaja del es que contiene información adicional que le permite optimizar la ejecución en tiempo de ejecución, basándose en elementos que no pueden ser considerados en tiempo de compilación.
- ✓ **Robusto:** La robustez es la medida de cuán fiable son los programas. Java contiene varias funcionalidades que le permiten mejorar la fiabilidad de las aplicaciones:
    - No tiene punteros, lo que evita que un usuario pueda manipular segmentos de la memoria y accidentalmente corromper datos esenciales para la ejecución de otros

programas o incluso el sistema operativo.

- Tiene un recolector de basura. Esta herramienta elimina de la memoria, de manera automática, elementos que no están siendo utilizados, evitando que los programadores olviden accidentalmente liberar la memoria o tengan que preocuparse por saber qué espacio de memoria liberar.

✓ **De arquitectura neutral:** Se dice que Java es arquitectónicamente neutral por ser su código independiente de cualquier plataforma en la que se esté ejecutando. Como ya se explicó el código de Java es traducido a bytecode, luego, la plataforma que necesite ejecutar programas desarrollados en Java sólo necesita tener una versión de la JVM, constituyendo esto una ventaja, pues contando solamente con una JVM para una arquitectura determinada garantizamos que un programa desarrollado en cualquier otra arquitectura sea funcional en esta. [10]

1.7.5 Entorno de desarrollo

Los entornos de desarrollo integrado (IDE por sus siglas en Ingles) son softwares que integran un conjunto de facilidades para el desarrollo de aplicaciones computacionales. Entre los más utilizados y populares para la programación en Java, por las facilidades que brindan y por ser de código abierto pueden mencionarse: Eclipse y NetBeans. Se seleccionó trabajar con el Eclipse en su versión 3.2 por los siguientes aspectos que se mencionan a continuación.

Eclipse

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

Por las características que a continuación son mencionadas se decidió utilizar Eclipse como entorno de desarrollo:

- Soporta Subversion como sistema de control de versiones.
- Dispone de un potente editor de código que presenta una interfaz amigable.
- Es una herramienta open-source y multiplataforma.
- Es ampliamente extensible por la capacitad que tiene de incorporar plugins al ambiente de trabajo.

1.7.6 Aplicaciones de soporte a la herramienta

Manteniendo la idea de reutilizar código o herramientas existentes y para hacer más eficiente nuestro trabajo hemos hecho uso de la herramienta **ANTLR** con el objetivo de generar los analizadores de los lenguajes que se utilizarán para procesar los archivos a importar.

ANTLR

La herramienta **ANTLR** (ANother Tool for Language Recognition) fue desarrollada por Terrance Parr, destacado profesor de la Universidad de San Francisco, y es un generador de analizadores que está dedicado al desarrollo de intérpretes. **ANTLR** es capaz de generar un analizador léxico, sintáctico o semántico en varios lenguajes(java, C++ y C#) a partir de unos ficheros escritos en un lenguaje propio. Dicho lenguaje es básicamente una serie de reglas **EBNF** y un conjunto de construcciones auxiliares. Los ficheros con los que trabaja **ANTLR** tienen extensión ***.g**, y en adelante los llamaremos **ficheros de especificación de gramáticas** o, directamente, **ficheros de gramáticas**.

Un fichero de gramática contiene la definición de uno o varios analizadores. Cada uno de estos analizadores se traducirá a código nativo (java, C++ o C#, dependiendo de ciertas opciones) en forma de clases. Es decir, por cada analizador descrito en el fichero de gramáticas se generará una clase en código nativo especificado, para nuestro caso utilizaremos la opción de código nativo java por ser el lenguaje de programación con el que trabaja nuestro proyecto.

Fig. 2: **Generación de un analizador utilizando la herramienta AntLR.**

ANTLR genera analizadores **pred-LL(k)**, y él mismo utiliza un analizador **pred-LL(k)** para leer los ficheros en los que están escritas las reglas **EBNF**. **ANTLR** admite acciones en sus reglas, además de otras prestaciones como paso de parámetros, devolución de valores o herencia de gramáticas.

ANTLR proporciona un apoyo excelente para la construcción del árbol de sintaxis abstracta, la traducción, la recuperación de errores, y el informe de errores. Siendo AntLR un programa que está

escrito en java, por lo que se necesita alguna máquina virtual de java para poder ejecutarlo. Es software libre, lo que quiere decir que al descargarlo de la página oficial (http://www.antlr.org) obtendremos tanto los ficheros compilados *.class como el código fuente en forma de ficheros *.java y su diseño ofrece gran extensibilidad y tiene muchas aplicaciones.

1.7.7 Sistema de control de versiones

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.

En la actualidad, es muy importante contar con un sistema de control de versiones en el proyecto pues esto garantiza la seguridad del mismo. Cuando se está escribiendo un programa o un documento muy largo (como un informe, o una tesis), y se quiere 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 la funcionalidad a implementar 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 del mismo:

- ✓ Es libre y está bajo licencia de tipo Apache/BSD.
- ✓ A diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente.
- ✓ 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. [11]

1.8 Conclusiones

Al concluir el presente capítulo se descubrió la necesidad de implementar la funcionalidad que permita al editor de ecuaciones de la Plataforma BioSyS llevar a cabo el proceso de importación de archivos de una manera transparente al usuario por la importancia que revierte el formato estándar **MathML** para guardar la información matemática de los sistemas biológicos. Se seleccionó **OpenUP** como metodología de desarrollo del software, al lenguaje de modelado y programación **UML** y **Java** respectivamente. Las herramientas de desarrollo seleccionadas fueron **Visual Paradigm** para **UML**, el entorno de desarrollo **Eclipse** y **AntLR** como herramienta para el procesamiento de los lenguajes.

Capítulo 2. Características del Sistema

En este capítulo se realiza la descripción de las principales definiciones asociadas al dominio del problema, se especifican los requisitos funcionales y no funcionales, se identifica mediante el diagrama de casos de uso del sistema las relaciones entre los actores y casos de uso del sistema; así como las descripciones textuales de cada caso de uso.

2.1 Modelo de dominio

El modelo de dominio permite mostrar de manera visual los principales conceptos que se manejan en el dominio del software en desarrollo para utilizar un vocabulario común que ayude tanto a usuarios, clientes, desarrolladores e interesados de manera general, a entender la lógica de funcionamiento de los procesos relacionados con el negocio, logrando una captura correcta y consensuada de requisitos.

"Una de las primeras actividades centrales de un ciclo de desarrollo consiste en crear un modelo conceptual para los casos de uso, en el cual se explican a sus creadores los conceptos significativos en un dominio del problema". [12]

Tomando esta experiencia, el modelo del dominio define un modelo de clases común para todos los implicados en el desarrollo, sirve como interlocutor entre clientes y desarrolladores. El propósito fundamental de este modelo es generar una terminología común y sentar las bases del entendimiento del desarrollo y no para definir el sistema completo.

2.1.1 ¿Por qué Modelo de Dominio?

Teniendo en cuenta que la metodología de desarrollo de software **OpenUP** no define entre sus principales flujos de trabajo un modelado del negocio, se determinó definir un modelo de dominio para el proyecto en curso con el objetivo de identificar cuáles son los principales conceptos relacionados con el objeto de estudio y cómo están relacionados.

2.1.2 Descripción de los principales conceptos y entidades identificadas

Investigador: Es el usuario que interactúa con el editor de ecuaciones, básicamente esta interacción se produce cuando ordena al editor de ecuaciones ejecutar alguna de las funcionalidades con que cuenta, estas pueden ser: editar un modelo matemático, realizar el análisis dimensional del mismo o exportar los modelos matemáticos en formato MathML.

Modelo Matemático: “Un modelo matemático es una representación para un objeto o proceso, en el cual quedan plasmadas sus principales características, a partir de conjeturas o suposiciones iníciales, estableciendo un compromiso entre complejidad y exactitud; en esta idealización es posible utilizar entes, estructuras y leyes de la matemática; permitiendo interpretar los resultados del modelo en términos del objeto o proceso estudiado”. [13]
Puede definirse también como un modelo matemático a una descripción desde el punto de vista de las matemáticas de un hecho o fenómeno del mundo real, con el objetivo de entender ampliamente el fenómeno y poder predecir su comportamiento en el futuro.

Sistema Biológico: Representa a un conjunto de individuos, población, elementos, organismos, o compuestos químicos que interactúan entre ellos. Es un sistema abierto que opera en condiciones alejadas del equilibrio termodinámico, con muchas y fuertes interacciones no lineales entre sus elementos.

Archivo: Un archivo, también denominado fichero, es una unidad de datos o información almacenada en algún medio. Cada archivo se diferencia del resto debido a que tiene un nombre propio y una extensión, además estos se diferencian entre si por su contenido que puede ser textos matemáticos codificados con uno de los siguientes estándares de representación matemática:

- MathML Presentation
- MathML Content
- Formato Interno

Editor de Ecuaciones: Representa a la herramienta computacional integrada a la Plataforma de Simulación y Análisis de Sistemas Biológicos (**BioSyS**) con el objetivo de proveer a los investigadores una aplicación informática en la que puedan editar los sistemas de ecuaciones que describen los modelos matemáticos que desean estudiar, además garantiza que la estructura sintáctica de las ecuaciones sea correcta y brinda mecanismos que permiten realizar un análisis de homogeneidad dimensional de las mismas.

Estándar de Codificación Matemática: Representa a las formas de codificar información de textos matemáticos con el objetivo de que su entendimiento sea universal.

Estándar MathML Content: Representa al estándar de codificación matemática MathML Content.

Estándar MathML Presentation: Representa al estándar de codificación matemática MathML Presentation.

Formato Interno del Editor de Ecuaciones: Representa al lenguaje de descripción de las expresiones matemáticas que pueden ser editadas por el Editor de Ecuaciones.

2.1.3 Representación del Modelo del Dominio

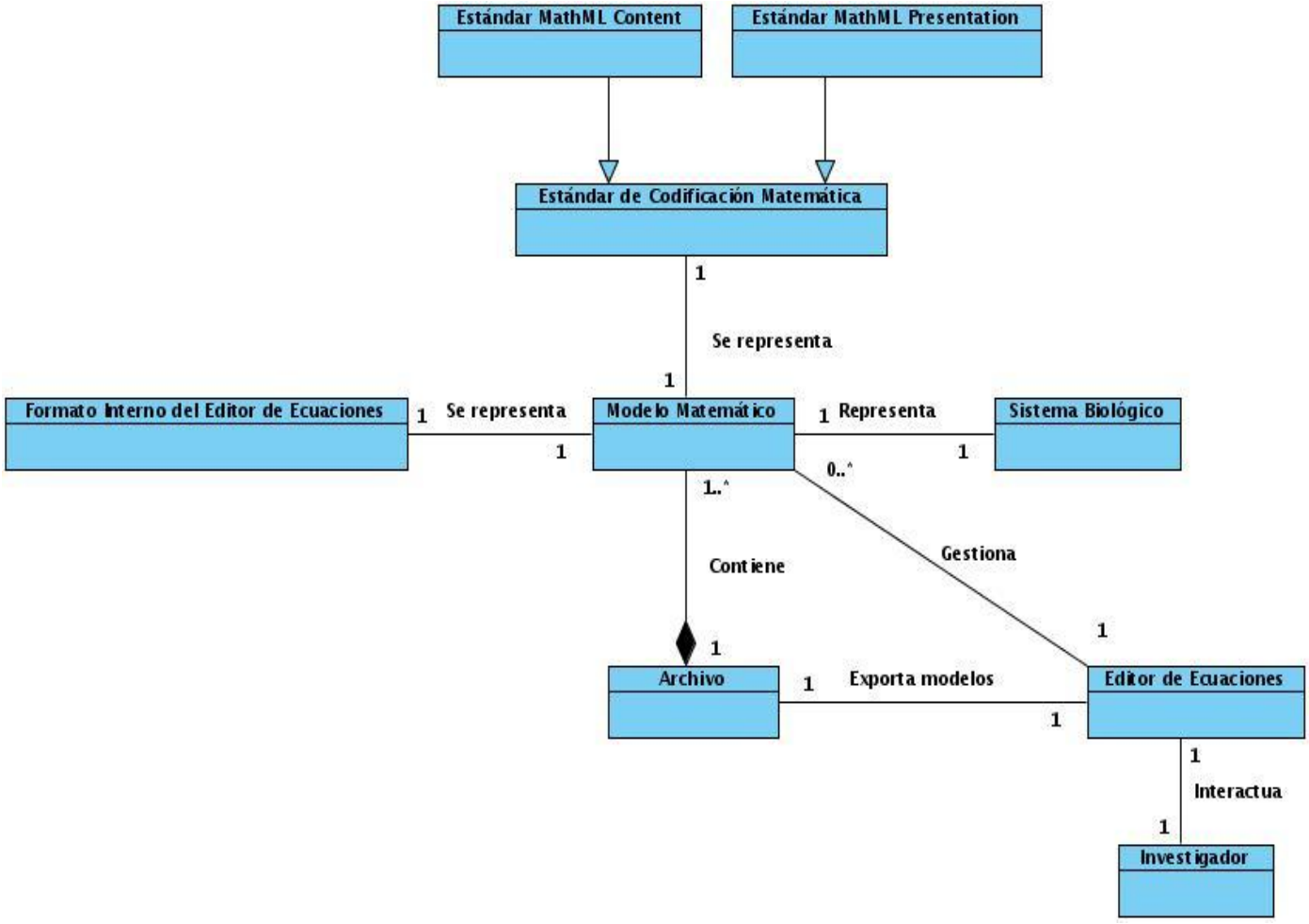


Fig. 3: Modelo de Dominio.

2.2 Especificación de los Requisitos del Software

Hacer una identificación exhaustiva y correcta de los requisitos fundamentales que debe cumplir un sistema es muy importante. “Un proyecto no puede ser exitoso sin una especificación correcta y exhaustiva de los requerimientos.” [9]

Una vez que se han definido los conceptos principales relacionados con el objeto de estudio y el dominio, se puede comenzar a analizar qué debe hacer el sistema para que se cumplan los objetivos planteados al inicio de este trabajo. “Para crear una aplicación de software hay que describir el problema y las necesidades o requerimientos: en qué consiste el conflicto y qué debe hacerse.” [9]
Para ello, se enumeran a través de requerimientos funcionales y no funcionales, las acciones que el sistema deberá ser capaz de realizar, así como las exigencias de despliegue para su uso efectivo. Las acciones que podrán ejecutar el usuario y el ambiente externo en que se usará el software son explicadas a continuación.

2.2.1 Requerimientos funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir. Permiten expresar una especificación más detallada de las responsabilidades del sistema en cuestión. Se mantienen invariables, sin importarle con que propiedades o cualidades se relacionen.
Los requerimientos funcionales del trabajo propuesto son los siguientes:

RF1. Importar Archivos.

1.1 Importar Archivos en código MathML Presentation.
1.2 Importar Archivos en código MathML Content.
1.3 Importar Archivos en código Interno.

2.2.2 Requerimientos no funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable. Normalmente están vinculados a requerimientos funcionales, es decir una vez se conozca lo que el sistema debe hacer podemos determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser:

- **Rendimiento:** Para la nueva funcionalidad es necesario mantener los niveles de rendimiento alcanzados por aplicaciones ya existentes, alta velocidad de procesamiento, respuesta rápida ante las solicitudes de los usuarios, además de alto grado de eficiencia y aprovechamiento máximo de los recursos en los clientes.

- **Usabilidad:** La nueva funcionalidad debe ser fácil de usar de manera que tenga gran aceptación entre los usuarios.

- **Extensibilidad:** La nueva funcionalidad debe ser capaz de permitir la integración con otros módulos del editor de ecuaciones, además de permitir la inserción de cambios.

- **Apariencia o interfaz externa:** La interfaz que debe brindar la nueva funcionalidad debe ser sencilla, amigable y de rápida respuesta frente a una petición del usuario de manera tal que agilice y facilite el trabajo con el software pues el sistema brindará servicios tanto a usuarios familiarizados con ambientes informáticos como a otros no familiarizados.

- **Soporte:** Para garantizar el soporte a los usuarios que interactúan con esta funcionalidad, se elaborará un manual de usuario y se realizarán encuentros con los responsables del área, con el fin de explicarles cómo funciona el proceso de importación.

- **Mantenimiento y actualización**: Debe dar facilidad de mantenimiento, y desarrollarse los más sencilla y eficientemente posible para que en un futuro pueda ser atendido por grupos de trabajo no especializados.

- **Compatibilidad:** La nueva funcionalidad debe ser capaz de correr de manera independiente a la plataforma sobre la que es ejecutada. Lo cual estará asegurado por las características del lenguaje de programación utilizado, en este caso, el lenguaje de programación java utilizado por los desarrolladores del polo garantiza la independencia de de la plataforma de desarrollo.

- **Software:** Para el uso del sistema es necesario tener instalado la máquina virtual de java 1,5 o una versión superior y disponer la herramienta AntLR con una versión superior a la 3.0.

- **Hardware:** En el caso de las PC deberán contar con un microprocesador Intel Pentium III o superior .Se debe contar con 256 de RAM como mínimo, preferentemente 512 MB o superior. Disco duro de 20GB como mínimo en dependencia a la información a almacenar.

- **Legales:** Se usarán herramientas de software libre y código abierto, o que funcionen bajo sistemas operativos representativos de código abierto, bajo las licencias GNU/GPL.

2.3 Actores del Sistema

Identificar quién o quiénes serán las entidades que interactuarán con nuestro sistema es el paso que sigue. “El actor es una entidad externa del sistema que de alguna manera participa en la historia del caso de uso. Por lo regular estimula el sistema con eventos de entrada o recibe algo de él.” [9]

Actor	Justificación
Investigador	Especialista encargado de interactuar con el sistema.

Tabla 2: Identificación de los Actores del Sistema.

2.4 Diagrama de casos de uso del sistema

Un caso de uso describe un proceso que deberá convertirse en una funcionalidad. “Un proceso describe, de comienzo a fin, una secuencia de eventos, de las acciones y las transacciones que se requieren para producir u obtener algo de valor para una empresa o actor.” [9]

Un diagrama de casos de uso del sistema representa gráficamente las funcionalidades principales del sistema y su interacción con los actores. En el caso que se está tratando solo se identifica un actor, el Investigador, que es el usuario que interactúa con el sistema para realizar el proceso de importación en los formatos antes mencionados.

En la siguiente figura se muestra el Diagrama de Caso de Uso Sistema:

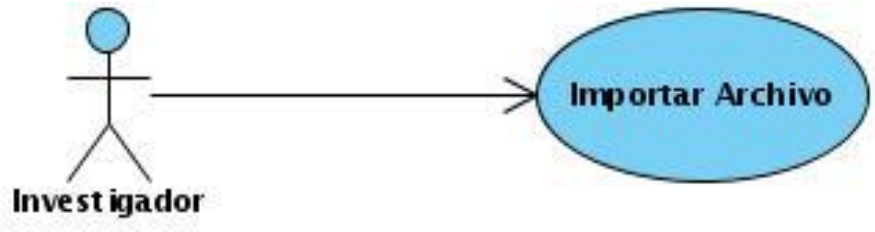


Fig. 4: Diagrama de Casos de Usos del Sistema.

2.5 Descripción de caso de uso de sistema

Para entender la funcionalidad asociada al caso de uso no es suficiente con la representación gráfica del diagrama de casos de uso, también se lleva a cabo la realización del mismo elaborando la descripción textual donde se especifican todas las acciones necesarias que realizan el actor y el

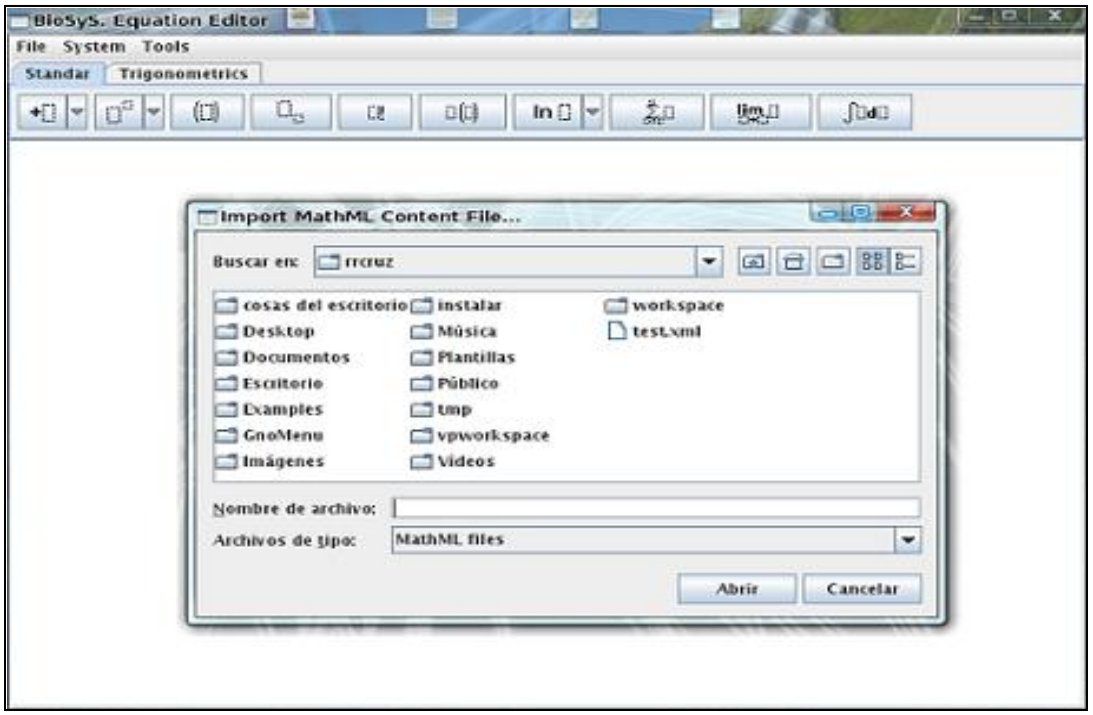
sistema. Con la descripción del caso de uso del sistema, que a continuación se presenta se obtendrá claramente la idea de cómo se realizara el proceso a automatizar y quienes intervienen directamente.

2.5.1 Descripción del Caso de Uso Importar Archivo

Caso de Uso:	Importar Archivo
Actores:	Investigador (Inicializa)
Propósito:	Permitir al investigador importar los modelos matemáticos de los sistemas biológicos contenidos en un archivo.
Resumen:	El caso de uso es iniciado por el investigador cuando selecciona la opción de importar. A continuación selecciona el tipo de archivo a importar y su ubicación. Inmediatamente el contenido de éste es visualizado en el editor de ecuaciones si no contiene errores.
Referencia:	RF 1
Precondiciones:	Debe existir un archivo en la ubicación seleccionada con uno de los siguientes formatos: • MathML Content • MathML Presentation • Formato Interno
Poscondiciones:	Se muestra el contenido del archivo importado en el Editor de Ecuaciones.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El investigador selecciona una opción para importar en los siguientes formatos: • MathML Content • MathML Presentation • Formato Interno	2. El sistema en dependencia del formato seleccionado realiza lo siguiente: • Si se selecciona MathML Content ir a la sección “Importar Archivo en formato MathML Content”. • Si se selecciona MathML Presentation ir a sección “Importar Archivo en formato MathML Presentation”. • Si se selecciona formato Interno ir a la sección “Importar Archivo en formato Interno”.

Sección "Importar Archivo en formato MathML Content"	
Acción del Actor	**Respuesta del Sistema**
	2 El sistema muestra una caja de diálogo para seleccionar el archivo que será importado.
3. El especialista selecciona el nombre del archivo a importar y presiona el botón aceptar.	4. El sistema analiza el archivo seleccionado. 5. Si el archivo no contiene errores el sistema visualiza en el área de edición su contenido.

Interfaz asociada a la sección del Caso de Uso Importar Archivo MathML Content.

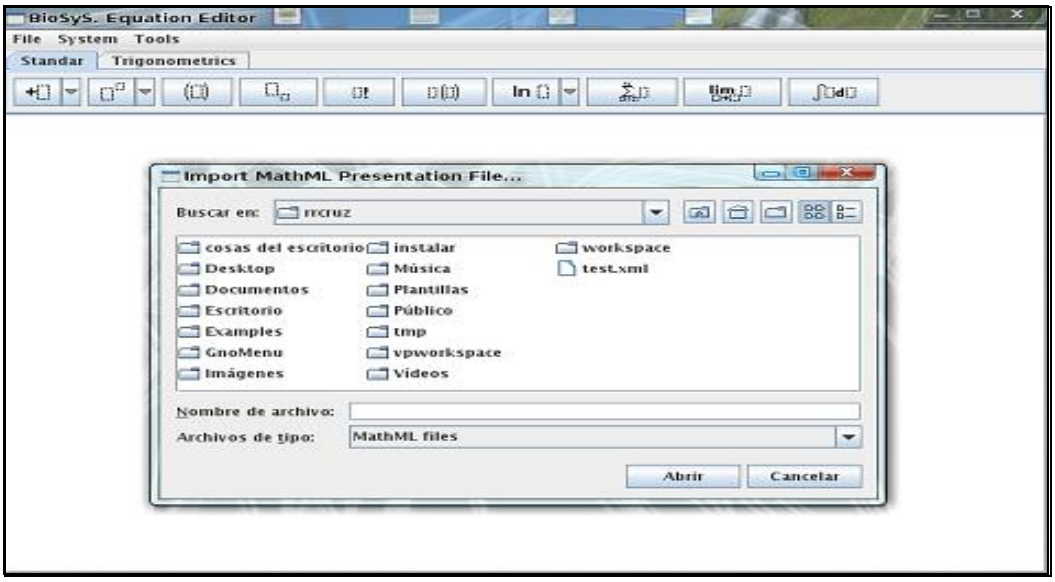


Curso Alternativo de Eventos Sección: "Importar Archivo en formato MathML Content"	
3 El especialista presiona el botón de cancelar.	4 El sistema cierra la caja de diálogo y no se ejecuta ninguna acción.
Curso Alternativo de Eventos Sección: "Importar Archivo en formato MathML Content"	
	5 Si el archivo contiene errores el sistema muestra una caja de diálogo reportando los errores detectados en el proceso de análisis.

Interfaz de error asociada a la sección del Caso de Uso Importar Archivo MathML Content

Sección "Importar Archivo en formato MathML Presentation"	
Acción del Actor	**Respuesta del Sistema**
	2. El sistema muestra una caja de diálogo para seleccionar el archivo que será importado.
3. El especialista selecciona el nombre del archivo a importar y presiona el botón aceptar.	4. El sistema analiza el archivo seleccionado. 5. Si el archivo no contiene errores el sistema visualiza en el área de edición su contenido.

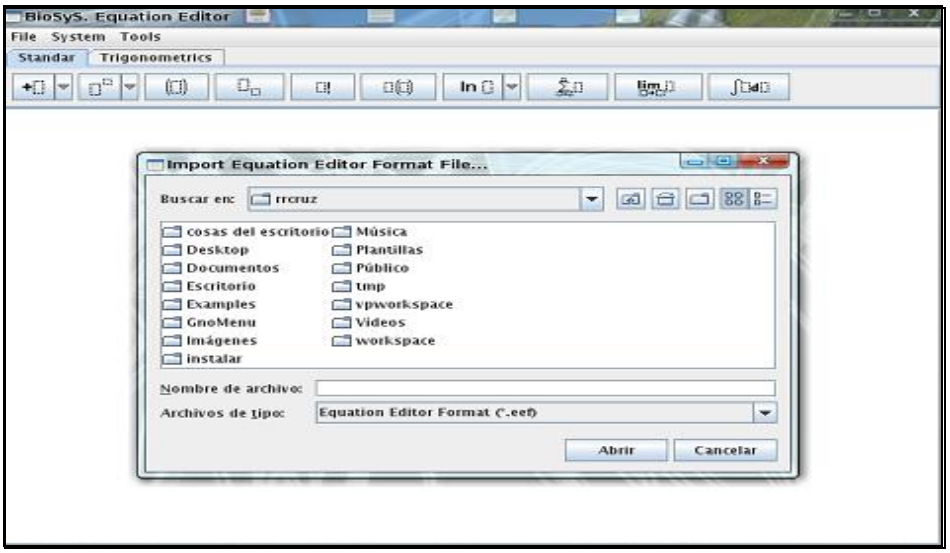
Interfaz asociada a la sección del Caso de Uso Importar Archivo MathML Presentation



Curso Alternativo de Eventos Sección: "Importar Archivo en formato MathML Presentation"	
3 El especialista presiona el botón de cancelar.	4 El sistema cierra la caja de diálogo y no se ejecuta ninguna acción.
Curso Alternativo de Eventos Sección: "Importar Archivo en formato MathML Presentation"	
	5 Si el archivo contiene errores el sistema muestra una caja de diálogo reportando los errores detectados en el proceso de análisis.

Interfaz de error asociada a la sección del CU Importar Archivo MathML Presentation

Sección "Importar Archivo en Formato Interno"

Acción del Actor	Respuesta del Sistema
	2. El sistema muestra una caja de diálogo para seleccionar el archivo que será importado.
3. El especialista selecciona el nombre del archivo a importar y presiona el botón aceptar.	4. El sistema analiza el archivo seleccionado. 5. Si el archivo no contiene errores el sistema visualiza en el área de edición su contenido.
Interfaz asociada a la sección del Caso de Uso Importar Archivo Formato Interno 	
Curso Alternativo de Eventos Sección: "Importar Archivo en Formato Interno"	
3 El especialista presiona el botón de cancelar.	4 El sistema cierra la caja de diálogo y no se ejecuta ninguna acción.
Curso Alternativo de Eventos Sección: "Importar Archivo en Formato Interno"	
	5. Si el archivo contiene errores el sistema muestra una caja de diálogo reportando los errores detectados en el proceso de análisis.
Interfaz de error asociada a la sección del CU Importar Archivo Equation Editor Format	

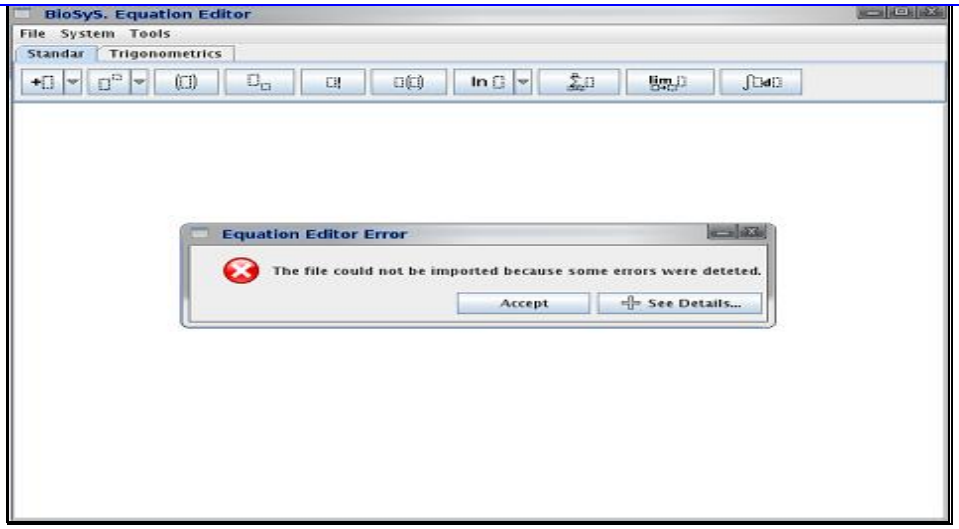


Tabla 3: Descripción del caso de uso Importar Archivo.

2.6 Conclusiones

En este capítulo se expuso la propuesta del sistema. Se trabajó en la fase de requerimientos, definiendo el modelo del dominio, así como los requerimientos funcionales y no funcionales que regirán el proceso de desarrollo del sistema. Se elaboró el diagrama de casos de uso del sistema y se describieron los mismos. Quedando listo el proceso para comenzar en la fase de diseño, donde se refinarán los requisitos y se ejecutarán otras actividades propias de ese flujo de trabajo.

Capítulo 3. Diseño del Sistema

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. La esencia de esta fase es la elaboración de diagramas de interacción, que muestran gráficamente cómo los objetos se comunican entre ellos a fin de cumplir con los requerimientos. Estos diagramas permiten la realización de los diagramas de clases del diseño, los cuales resumen la definición de las clases que se pueden implementar en el software. A través del presente capítulo se dará una descripción del estilo arquitectónico utilizado para el desarrollo de la funcionalidad, se describirán los patrones de diseño empleados y los diagramas de clases del diseño.

3.1 Patrón Arquitectónico

Los patrones arquitectónicos expresan un esquema organizativo estructural fundamental para sistemas software y definen las reglas generales de organización, las restricciones en la forma y la estructura de un grupo numeroso de sistemas de software. La selección de un patrón arquitectónico es una decisión fundamental de diseño en el desarrollo de un sistema de software.

Para el desarrollo de las funcionalidades a implementar se utilizó el patrón arquitectónico Modelo-Vista-Controlador (Model-View-Controller MVC). El patrón MVC es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos:

- El Modelo: Es la representación específica de la información con la cual el sistema opera. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos.
- La Vista: Presenta el modelo en un formato adecuado para interactuar con los usuarios, usualmente está conformada por las interfaces de usuario.
- El Controlador: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Como se mencionó anteriormente, este patrón 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.

El Patrón MVC (Ver Figura 5) está 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 del

negocio, o sea, para el diseño de herramientas que necesitan la habilidad de mantener múltiples vistas para los mismos datos, tales como editores gráficos. [15]

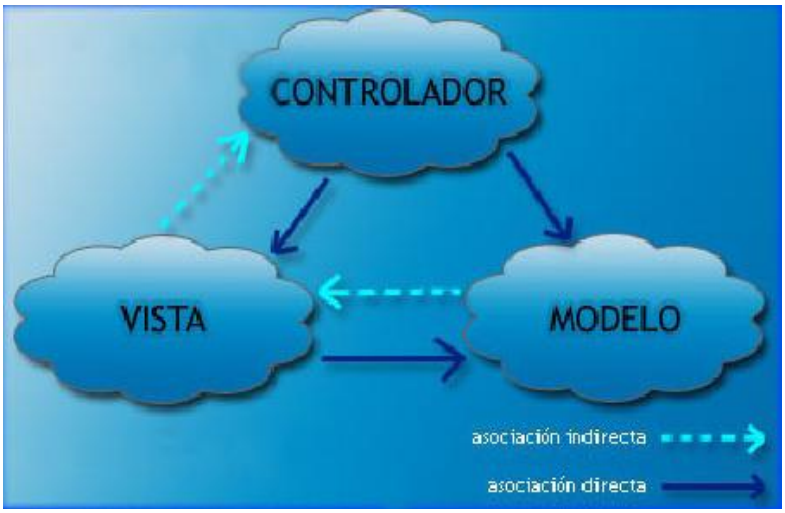


Fig. 5: Representación Patrón Modelo-Vista-Controlador.

Particularmente se decidió seleccionar este tipo de estilo arquitectónico para organizar los principales componentes del sistema a desarrollar y representar las relaciones que entre ellos se establece porque es el estilo que describe la arquitectura del Editor de Ecuaciones, además la herramienta a desarrollar constituye una nueva funcionalidad que será añadida al Editor de Ecuaciones y es por ello que adopta su estilo arquitectónico. También es válido aclarar que solo se operará sobre dos capas de estilo arquitectónico, en particular, sobre la **vista** pues serán añadidas nuevas interfaces para efectuar el proceso de importación de archivos, se modificarán interfaces ya existentes y sobre el **controlador** pues se añadirán un grupo de clases que coordinarán el proceso de importación de un archivo.

A continuación se ofrece una descripción más detallada de la arquitectura de la aplicación (Ver figura 6), donde se muestra cómo están distribuidos los componentes por las diferentes capas del patrón MVC en la vista arquitectónica.

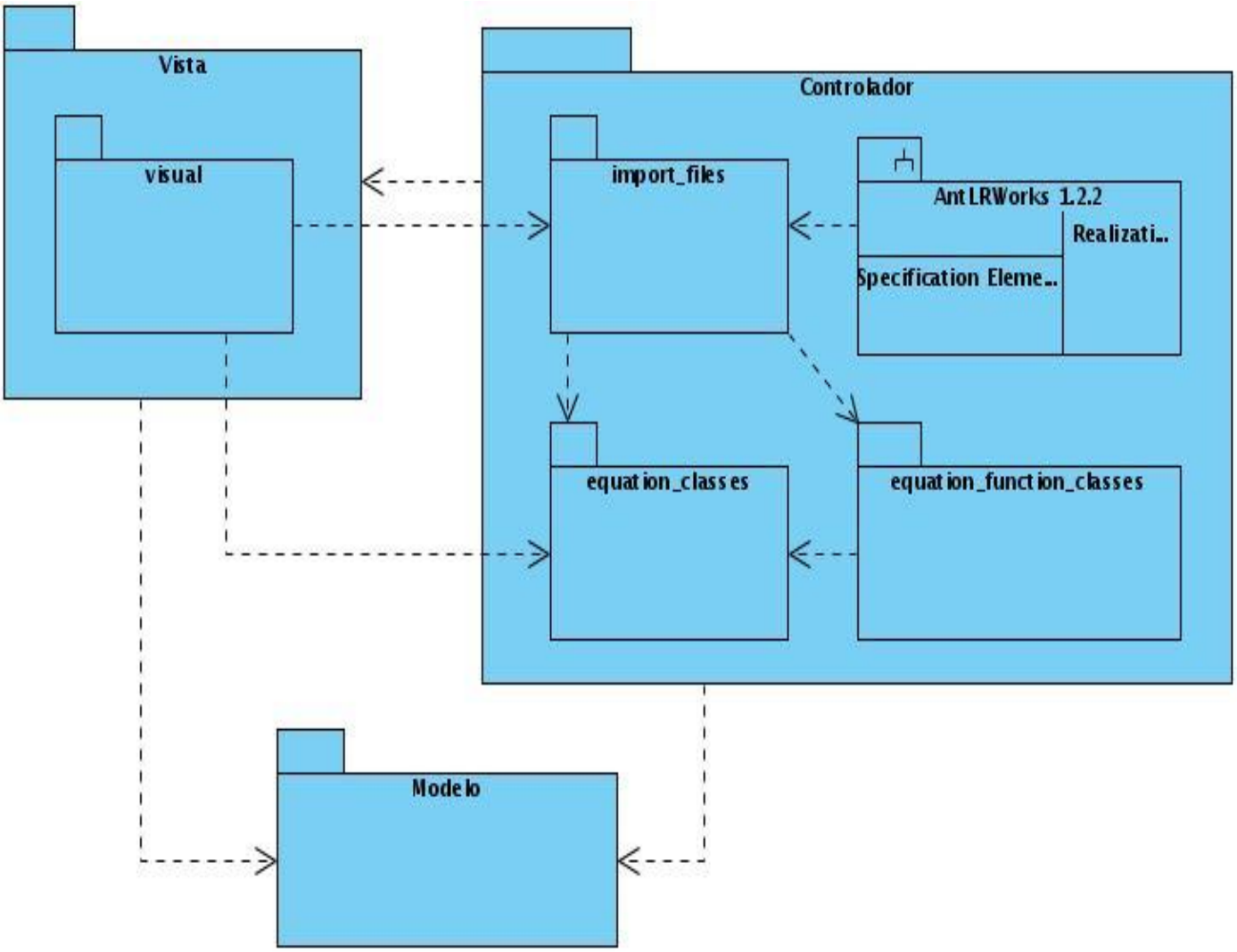


Fig. 6: Vista arquitectónica.

3.2 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. “Cada patrón describe un problema que ocurre una y otra vez en nuestro ambiente, y luego describe el núcleo de la solución a ese problema, de tal manera que puede usar esa solución un millón de veces más, sin hacer jamás la misma cosa dos veces.” [16]

Existen diferentes clasificaciones. Dentro de los patrones de producto de software se encuentran los de análisis, arquitectura, diseño y lenguaje de programación. Para el desarrollo de la solución se aplicaron diferentes patrones de diseño, fundamentalmente los patrones de diseño: GRASP y GoF,

con el objetivo de facilitar el mantenimiento del software y contribuir a la realización de un producto reutilizable y escalable.

3.2.1 Patrones GRASP

Los patrones de asignación de responsabilidades GRASP (del inglés: General Responsibility Assignment Software Patterns), permiten asignar correctamente las responsabilidades a cada una de las clases que intervienen en el modelo; de este grupo de patrones fueron tomados en cuenta para una correcta asignación de las relaciones entre las clases y un correcto diseño de las mismas, los siguientes:

- **Experto:** Describe la asignación de responsabilidades al experto en información, clase que posee la información necesaria para cumplir con la responsabilidad.
  El uso de este patrón queda evidenciado (ver figura 7) en el diseño de la clase Import_File. Esta clase puede catalogarse como un experto en información porque dispone de toda la información necesaria para coordinar todo el proceso de importación, o sea, cuenta con los datos necesarios para saber qué tipo de analizador utilizar para procesar el archivo seleccionado, donde está ubicado el mismo y además gestiona los posibles errores de este proceso.

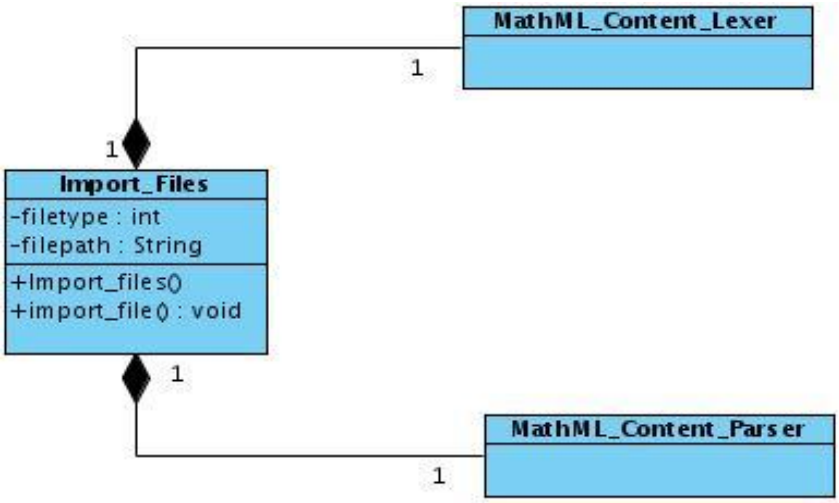


Fig. 7: Diagrama donde se evidencia el Patrón Experto.

- **Creador:** Se refiere a asignar responsabilidades a las clases de crear instancias de otras conociendo que las primeras son las que contienen la información para ello.

El uso de este patrón queda evidenciado (ver figura 8) en el diseño de la clase Import_File, pues esta clase es la encargada de crear las instancias de los diferentes analizadores a utilizar para el proceso de análisis de los archivos.

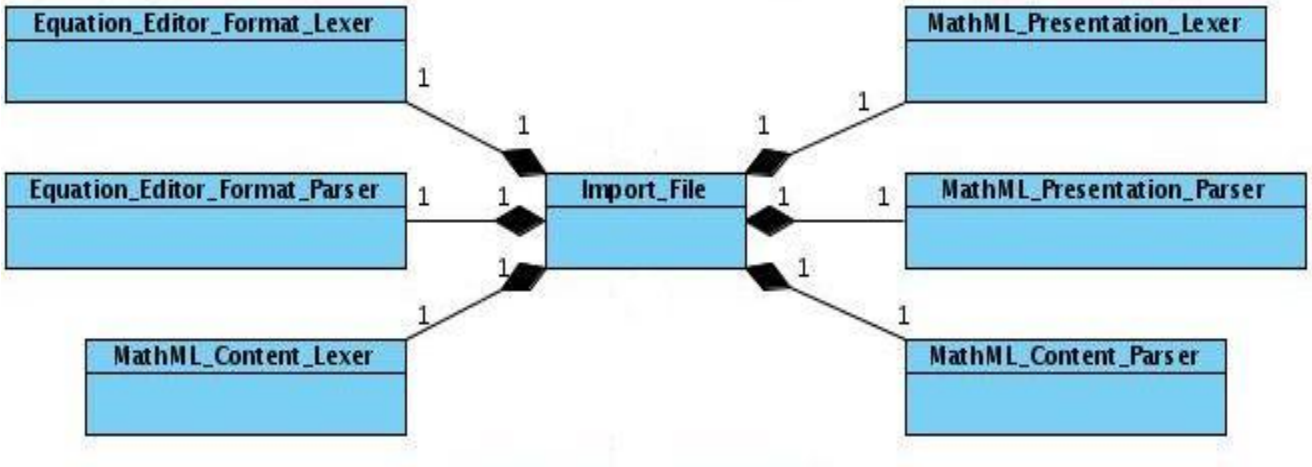


Fig. 8: Diagrama donde se evidencia el Patrón Creador.

- **Bajo acoplamiento:** Cada clase está acoplada (relacionada) a las clases estrictamente necesarias, garantizando un bajo impacto de los cambios que se producen en una clase para las demás clases que se relacionan con ella.
  El uso de este patrón queda evidenciado (ver figura 8) en la asignación de las relaciones entre las clases, pues cada una de ellas está relacionada de manera que se establezcan sólo las dependencias necesarias para cumplir con sus responsabilidades, esto favorece a la flexibilidad del diseño y a la actualización de los cambios del sistema pues las clases son menos dependientes entre si.

- **Alta cohesión:** Asignar responsabilidades a las clases de manera que todos sus métodos tuvieran un comportamiento bien definido.
  El uso de este patrón queda evidenciado en el diseño de las clases, pues cada una de ellas contiene sólo las funcionalidades que le corresponden según la información que manejan. Esto queda evidenciado, por ejemplo (ver figura 9), con la clase Import_Files a través del método **import_file()** se coordina el proceso de importación mientras que la responsabilidad de efectuar el proceso de análisis de un archivo queda delegada a los diferentes analizadores, en

este caso, el análisis léxico para el lenguaje MathML Content queda delegado al método **scan()** de la clase MathML_Content_Lexer.

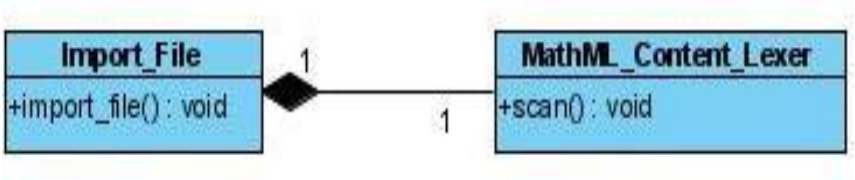


Fig. 9: Diagrama donde se evidencia el Patrón Alta Cohesión.

3.2.2 Patrones GoF

Otro conjunto de patrones de diseño bien conocidos son los patrones conocidos como grupo de los cuatro **GoF** (del inglés: Gang of Four) y se dividen en 3 tipos fundamentales:

- **Creacionales:** abstraen el proceso de creación de instancias.
- **Estructurales:** se ocupan de cómo clases y objetos son utilizados para componer estructuras de mayor tamaño.
- **Comportamiento:** atañen a los algoritmos y a la asignación de responsabilidades entre objetos.

Algunos de estos patrones fueron aplicados directamente en el diseño de este sistema, a continuación se menciona el nombre del patrón y las clases donde se aplicaron directamente.

- **Intérprete (Interpreter):** Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.

  El uso de este patrón queda evidenciado en el diseño de las clases que se utilizan para interpretar los diferentes lenguajes porque para realizar el proceso de importación de los archivos cada una de estas clases implementa gramáticas para comprobar la validez de los lenguajes y utilizan a la biblioteca AntLRWorks como herramienta de apoyo para efectuar un análisis descendente recursivo de los lenguajes.

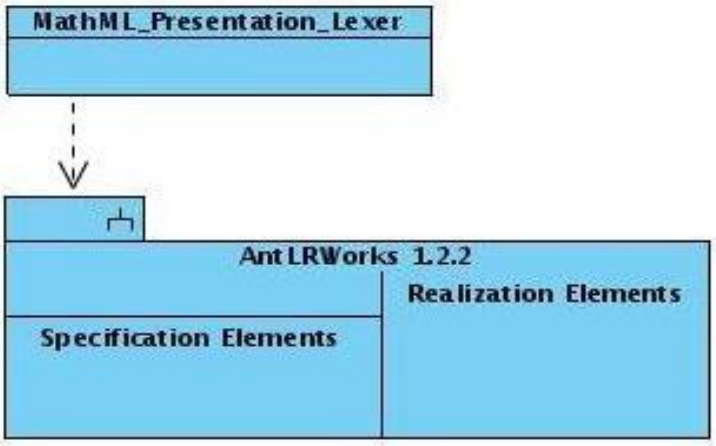


Fig. 10: Diagrama donde se evidencia el uso del Patrón Intérprete.

- **Mediador (Mediator):** Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
  El uso de este patrón queda evidenciado en el diseño con la instancia de la clase Import_File la cual establece un hilo conductor de los mensajes a ejecutar por las diferentes instancias de los analizadores y coordina como han de funcionar en conjunto.

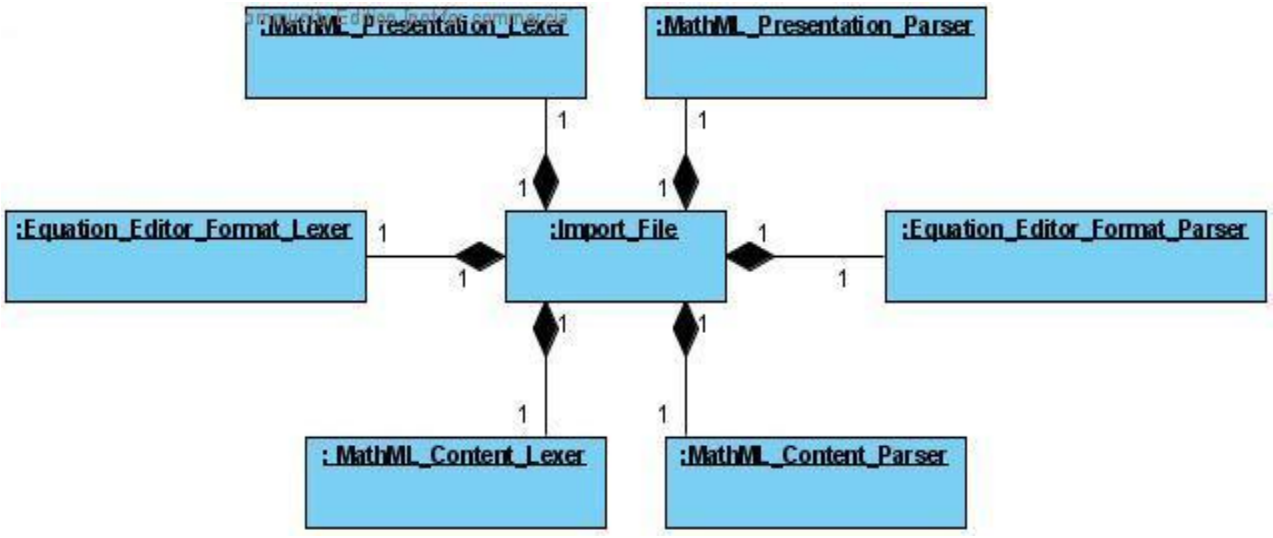


Fig. 11: Diagrama donde se evidencia el uso del Patrón Mediator.

- **Observador (Observer):** Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

El uso de este patrón queda evidenciado, por ejemplo (ver figura 11), con la clase controladora principal Import_File porque todas las instancias de los analizadores de lenguajes con las que está relacionada están subordinadas a su estado, de modo que al variar el estado de la instancia que representa a la clase Import_Files inmediatamente cambiarán y se actualizarán los estados de cada uno de las instancias con las que está relacionada.

3.3 Modelo Del Diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso. Se centra en cómo los requisitos funcionales y no funcionales tienen impacto en el sistema a desarrollar. [17]

Dentro de sus propósitos están: crear una entrada apropiada y un punto de partida para la implementación, descomponer los trabajos de implementación en partes más manejables que puedan ser llevadas a cabo por diferentes equipos de desarrollo. Adquirir una comprensión de los aspectos relacionados con los requisitos no funcionales y restricciones relacionadas con los lenguajes de programación, componentes reutilizables, sistemas operativos, tecnologías de distribución y concurrencia. El modelo de diseño puede estar conformado principalmente por paquetes, clases, subsistemas de diseño y sus relaciones.

Un paquete de diseño es una colección de clases relacionadas, realizaciones de CU, diagramas y otros paquetes. Es usado para estructurar el modelo de diseño mediante su división en partes más pequeñas y para agrupar elementos del modelo de diseño relacionados con propósitos organizacionales y frecuentemente para administración de configuración.

Diferente al artefacto subsistema del diseño, un paquete de diseño no ofrece ninguna interfaz formal aunque debe exponer algo de su contenido (marcado como público) que ofrece comportamiento.

Los subsistemas de diseño y las clases del diseño representan abstracciones del subsistema y componentes de implementación del sistema. Estas abstracciones son directas, y representan una sencilla correspondencia entre el diseño y la implementación. [17]

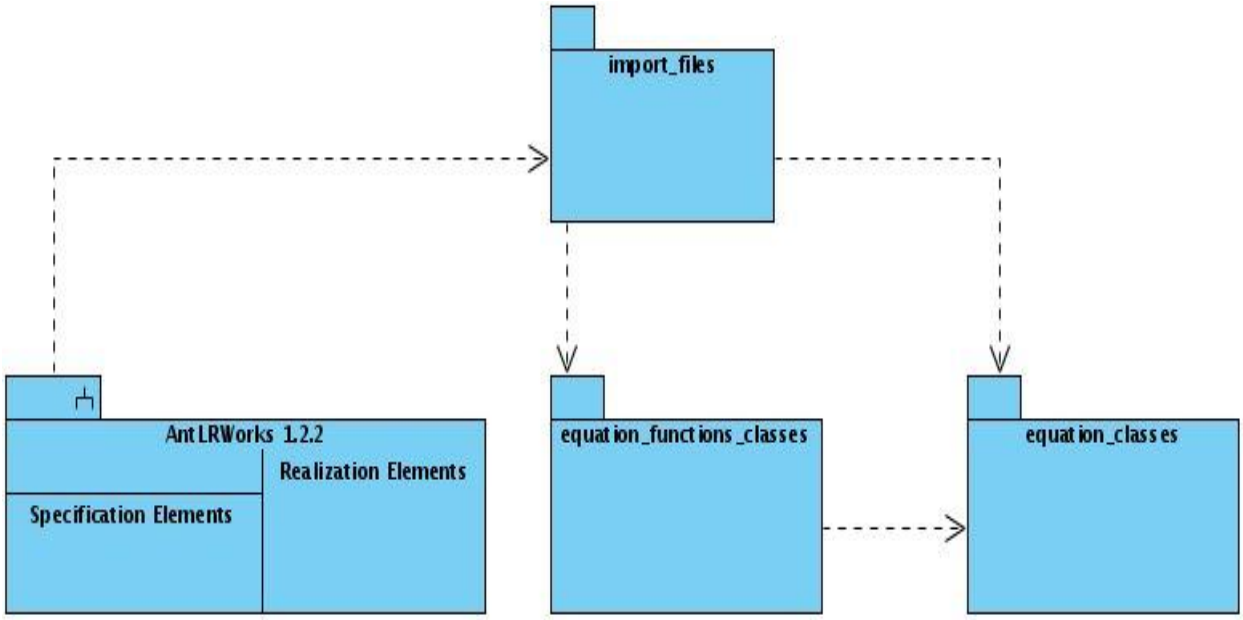


Fig. 12: Modelo del Diseño.

3.3.1 Descripción de paquetes y subsistemas del modelo del diseño

El modelo de diseño está constituido por los siguientes paquetes y subsistemas:

- **Paquete import_file:** Este paquete contiene las clases necesarias para controlar el proceso de importación. Cuenta con un grupo de analizadores de lenguajes, que básicamente son clases generadas por el subsistema **AntLRWorks 3.0 IDE**, cuyo objetivo es realizar el análisis léxico y sintáctico de los archivos a importar. A su vez, cada uno de los analizadores sintácticos genera un árbol de sintaxis abstracta que contiene la representación algebraica de los modelos matemáticos contenidos en el archivo importado. Este paquete dispone también de una réplica de la instancia de la clase **EquationController**, que pertenece al paquete **equation_classes**, con el objetivo de ordenarle la inserción de los modelos matemáticos encontrados en el archivo importado al área actual de edición del editor de ecuaciones.

- **Paquete equation_classes:** Este paquete contiene un grupo de clases encargadas de controlar el proceso de gestión de los modelos matemáticos y la representación de operaciones matemáticas que requieren del uso de interfaces especiales. Ejemplo de estas operaciones son la diferenciación y la sumatoria, cuya representación en el área de edición requiere de **cajas múltiples**.

- **Paquete equation_function_classes:** Este paquete contiene un grupo de clases encargadas de la representación de las operaciones algebraicas que pueden ser llevadas a cabo con las funciones matemáticas elementales como las trigonométricas y las exponenciales. Además contiene clases para la representación de operaciones como la integración que requiere el uso de ciertas clases pertenecientes al paquete equation_classes.

- **Subsistema AntLRWorks 3.0 IDE:** Herramienta encargada de generar las diferentes clases que servirán de analizadores léxicos y sintácticos para procesar los archivos a importar. Para generar estas clases la herramienta compila un grupo de archivos previamente editados que contienen las reglas gramaticales de los lenguajes a analizar, además implementa de forma automática en estas clases los mecanismos necesarios para generar los árboles de sintaxis abstracta utilizando las clases destinadas a la representación de las operaciones matemáticas adjuntas en los paquetes equation_classes y equation_function_classes respectivamente.

3.4 Diagrama de clases del diseño del paquete import_file

Un diagrama de clases del diseño es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso de análisis y diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargaran del funcionamiento y la relación entre uno y otro. Un diagrama de clases del diseño, muestra las clases del diseño enfocadas a un lenguaje en específico, ya en este diagrama se manejan especificaciones más técnicas y detalladas. Son la primicia para entender lo que posteriormente se va a implementar. [17]

A continuación, en la siguiente figura, se muestra el diagrama de clases del diseño correspondiente al paquete import_file.

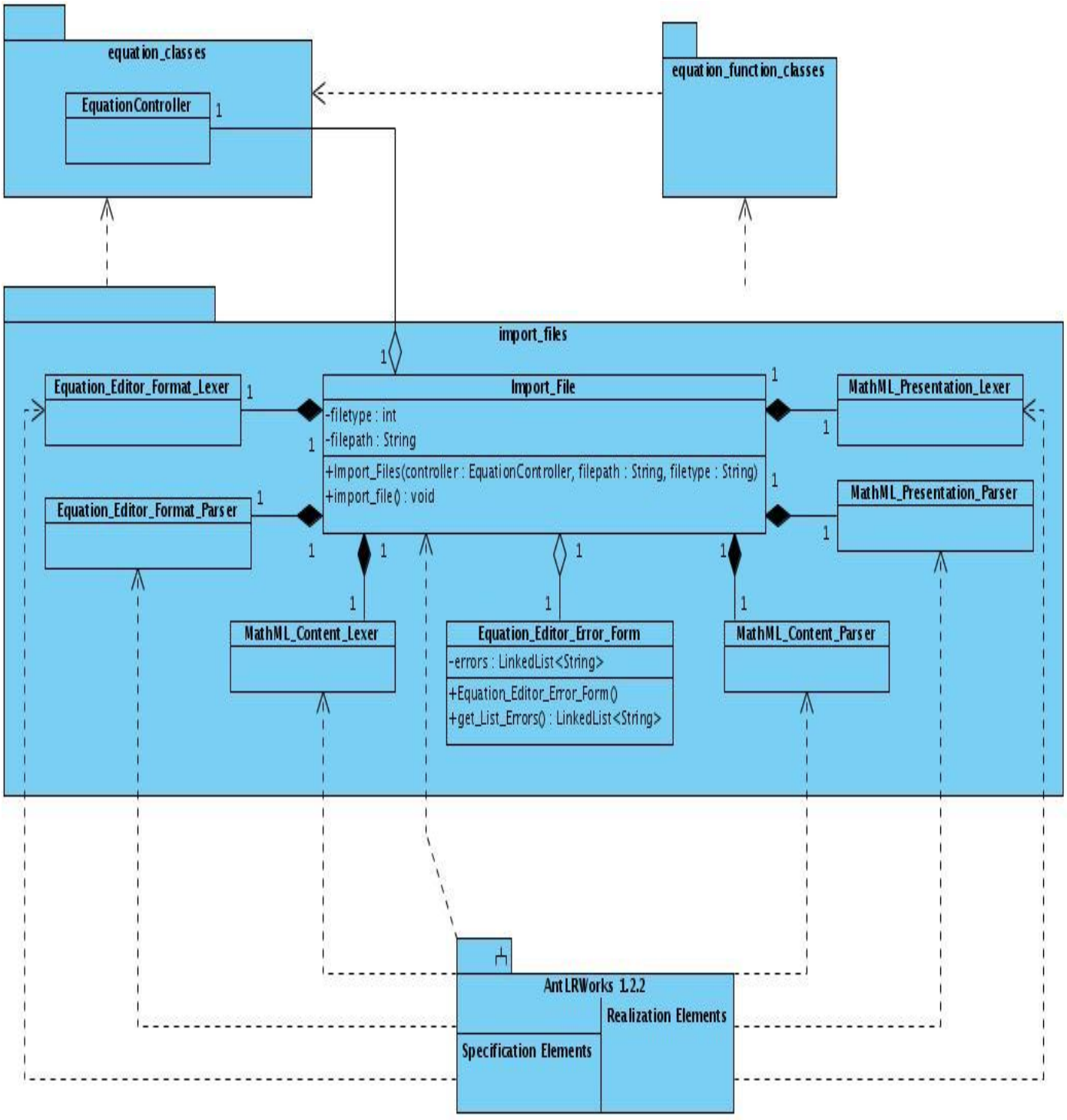


Fig. 13: Diagrama de Clases del Diseño del paquete import_file.

3.5 Descripción de las Clases del Diseño

A continuación se ofrece una descripción detallada de los atributos y métodos de las clases principales.

Nombre: Import_Files	
Tipo de Clase: Controladora	
Atributo	**Tipo**
controller	EquationController
lexer_ee	Equation_Editor_Format_Lexer
parser_ee	Equation_Editor_Format_Parser
lexer_c	MathML_Content_Lexer
parser_c	MathML_Content_Parser
lexer_p	MathML_Presentation_Lexer
parser_p	MathML_Presentation_Parser
form	Equation_Editor_Error_Form
filepath	String
filetype	int
Para cada responsabilidad	
Nombre	Import_File(EquationController controller, String filepath, int filetype)
Descripción:	Constructor de la clase, recibe como parámetros una variable de tipo EquationController para mantener una referencia a la clase encargada de administrar las ecuaciones que están siendo editadas, una variable de tipo cadena para indicar la ruta del fichero que será importado y una variable de tipo entero para indicar el tipo de fichero a importar.
Nombre	import_file()
Descripción	Método encargado de efectuar el proceso de importación del archivo seleccionado. Coordina en dependencia del tipo de archivo a importar los correspondientes analizadores a utilizar, además gestiona los

	posibles errores que se registren durante el proceso. Si este proceso culmina exitosamente, agrega al área de trabajo actual del editor de ecuaciones los modelos matemáticos encontrados.

Tabla 4: Descripción de la clase de diseño Import_Files.

Nombre: Equation_Editor_Error_Form	
Tipo de Clase: Interfaz de Usuario	
Atributo	**Tipo**
errors	LinkedList<String>
Para cada responsabilidad	
Nombre	Equation_Editor_Error_Form()
Descripción:	Constructor de la clase, método encargado de inicializar todos los componentes de la caja de diálogo con los valores deseados en sus propiedades.
Nombre	get_List_Errors()
Descripción	Método encargado de devolver una referencia de la lista de errores que posee la caja de diálogo para su posterior actualización.

Tabla 5: Descripción de la clase de diseño Equation_Editor_Error_Form.

Nombre	**Descripción**
Tipo de Clase:	Controladora
Equation_Editor_Format_Lexer	Clases generadas por la herramienta AntLRWorks cuya función es realizar el análisis léxico de los diferentes tipos de archivos a importar. Para ello disponen de un conjunto amplio de atributos para representar las posibles operaciones que pueden llevar a cabo y un gran grupo de operaciones para comprobar la validez del proceso de escaneo.
MathML_Content_Lexer	
MathML_Presentation_Lexer	

Tabla 6: Descripción de las clases encargadas de realizar el análisis léxico de los lenguajes.

Nombre	Descripción
Tipo de Clase	Controladora
Equation_Editor_Format_Parser	Clases generadas por la herramienta AntLRWorks cuya función es realizar el análisis sintáctico del contenido de los diferentes tipos de archivos a importar. Para ello disponen de un conjunto amplio de atributos que representan los identificadores de los símbolos terminales, un grupo de autómatas finitos deterministas para determinar los posibles estados del análisis sintáctico y un gran grupo de métodos que representan a los símbolos no terminales de la gramática y los posibles estados de transición de los autómatas.
MathML_Content_Parser	
MathML_Presentation_Parser	

Tabla 7: Descripción de las clases encargadas de realizar el análisis sintáctico.

3.6 Diagramas de secuencias del diseño

Un Diagrama de Secuencias contribuye a la descripción de la dinámica del sistema en términos de la interacción entre sus objetos. Los diagramas de interacción son usados para identificar las clases y los métodos que ellos entregan. [17]

Para ver los diagramas de secuencia del flujo normal de eventos, sus flujos alternos y cursos alternativos del caso de uso Importar Archivo consulte los **(Anexos 5, 6 y 7).**

3.7 Conclusiones

En este capítulo se identificó el estilo arquitectónico que se utilizó para el desarrollo de las funcionalidades así como los patrones de diseño utilizados y la fundamentación del por qué y cómo se usan. También fueron desarrollados los diagramas de clases del diseño y los diagramas de secuencia correspondientes al caso de uso Importar Archivo.

Capítulo 4. Implementación

En este capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, para esto se muestra el diagrama de componentes. Se mostrará el código fuente de los principales métodos de las clases; se desarrollarán, además, varias pruebas para conocer la calidad del producto, utilizando los métodos de caja negra en cada caso para asegurar que los requisitos funcionales demandados fueron cumplidos y asegurar el correcto funcionamiento de la aplicación.

4.1 Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes software. Los componentes pueden ser 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. [18]
A continuación se presenta el diagrama de componentes correspondiente al paquete **import_files** y se describen brevemente cada una de las funciones de los componentes que lo integran.

- **Import_Files.java:**
  Este componente representa la clase controladora principal, su función es coordinar todo el proceso de importación. Para ello dispone de la información necesaria para saber qué tipo de analizador utilizar para procesar un archivo y cómo gestionar los posibles errores. Para la realización de este proceso necesita del uso de la librería **AntLRWorks**.

- **Equation_Editor_Format_Lexer.java:**
  Este componente representa la clase encargada de realizar el análisis léxico del lenguaje definido por el Editor de Ecuaciones para representar las operaciones matemáticas, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **MathML_Content_Lexer.java:**
  Este componente representa la clase encargada de realizar el análisis léxico del lenguaje generado por el estándar de representación matemática **MathML Content**, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **MathML_Presentation_Lexer.java:**
  Este componente representa la clase encargada de realizar el análisis léxico del lenguaje generado por el estándar de representación matemática **MathML Presentation**, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **Equation_Editor_Format_Parser.java:**
  Este componente representa la clase encargada de realizar el análisis sintáctico del lenguaje definido por el Editor de Ecuaciones para representar las operaciones matemáticas, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **MathML_Content_Parser.java:**
  Este componente representa la clase encargada de realizar el análisis sintáctico del lenguaje generado por el estándar de representación matemática **MathML Content**, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **MathML_Presentation_Parser.java:**
  Este componente representa la clase encargada de realizar el análisis sintáctico del lenguaje generado por el estándar de representación matemática **MathML Presentation**, para realizar este proceso se auxilia de la librería **AntLRWorks.**

- **Equation_Editor_Error_Form.java:**
  Este componente representa la clase encargada de mostrar los mensajes de errores que pueden producirse durante el proceso de importación.

- **AntLRWorks 1.2.2.jar**:
  Este componente representa la librería utilizada para realizar el proceso de importación. Esta herramienta brinda una serie de clases que son utilizadas por los diferentes analizadores de lenguajes para realizar el análisis léxico y sintáctico de un archivo mediante el uso de analizadores descendente recursivos.

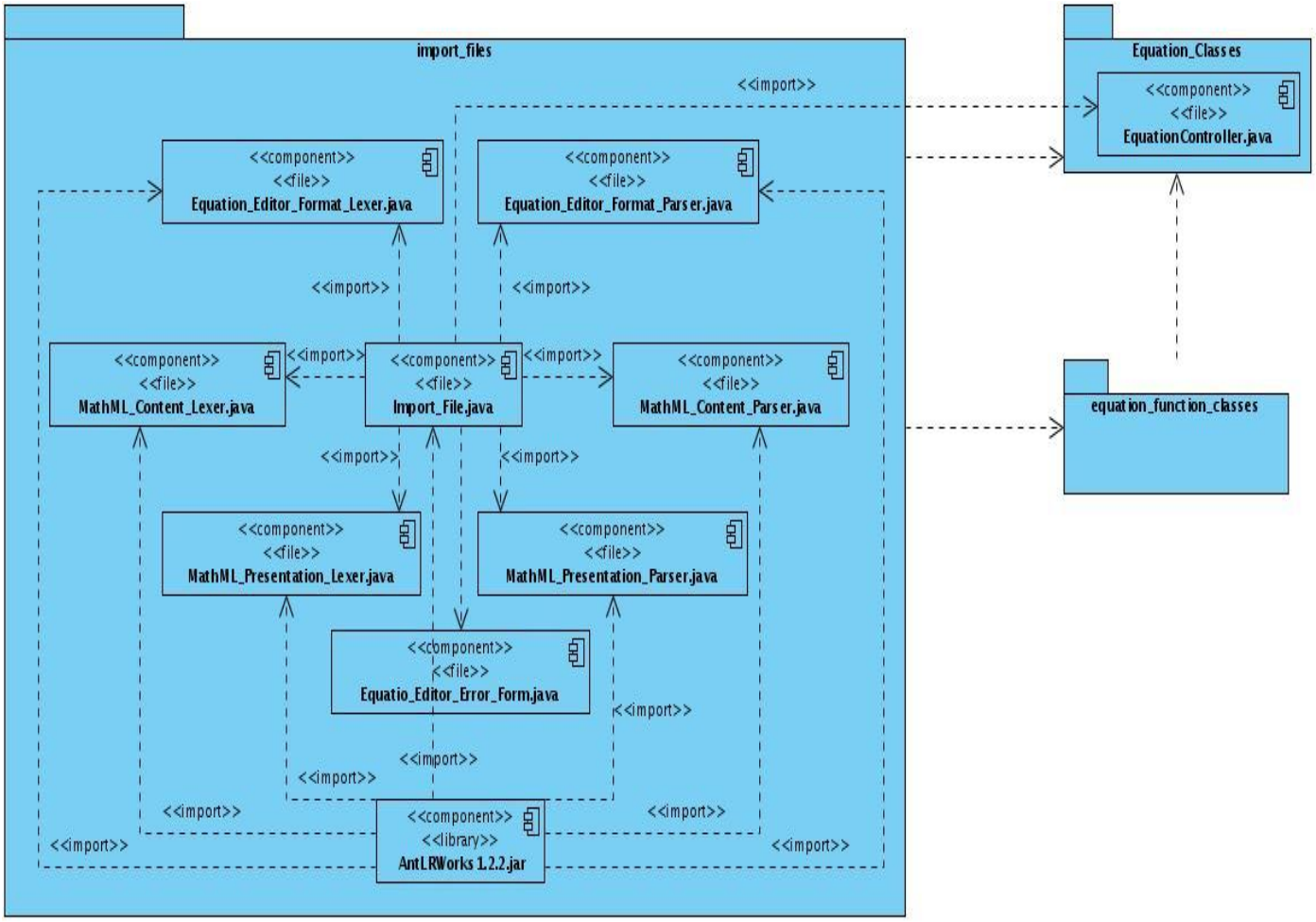


Fig. 14: Diagrama de componentes.

4.2 Descripción del código fuente de las principales clases

En las siguientes secciones se muestra el código de alguno de las principales clases.

4.2.1 Código fuente de la clase Import_Files

A continuación se describe el código fuente de la clase Import_Files que es la clase controladora principal pues gestiona el proceso de importación de los archivos. Posee diferentes tipos de instancias de los analizadores léxicos y sintácticos a utilizar para analizar un archivo. Posee un único método, **import_file(),** que se encarga de ejecutar el tipo analizador indicado por la variable filetype para el archivo con ubicación indicada por la variable filepath. En caso de detectar errores en este proceso se muestra la interfaz correspondiente a la clase Equation_Editor_Error_Form mostrando los errores al usuario.

```
public class Import_Files {
        private EquationController controller;
        private Equation_Editor_Format_Lexer lexer_ee;
        private Equation_Editor_Format_Parser parser_ee;
        private MathML_Presentation_Lexer lexer_p;
        private MathML_Presentation_Parser parser_p;
        private MathML_Content_Lexer lexer_c;
        private MathML_Content_Parser parser_c;
        private String filepath;
        private int filetype;
        public Import_Files (EquationController controller, String filepath, int filetype){
                this.controller=controller;
                this.filepath=filepath;
                this.filetype=filetype;
        }
        public void import_file ()
        {
                try
                {
                 CharStream input = new ANTLRFileStream (filepath);
                 LinkedList<String> lexing_errors=new LinkedList<String> ();
                 LinkedList<String> parsing_errors=new LinkedList<String> ();
                 LinkedList<BasicBox> boxes=new LinkedList<BasicBox> ();
                 switch (filetype)
                 {
                   case 0:
                                lexer_ee = new Equation_Editor_Format_Lexer (input);
                                lexer_ee.setList (lexing_errors);
                                CommonTokenStream tokens_ee = new CommonTokenStream (lexer_ee);
                                parser_ee=new Equation_Editor_Format_Parser (tokens_ee);
                                parser_ee.setController (controller);
                                parser_ee.setList (parsing_errors);
                                parser_ee. Programa ();
                                boxes =parser_ee.getBoxes ();
                                break;

                   case 1:
                                lexer_p = new MathML_Presentation_Lexer (input);
                                lexer_p.setList (lexing_errors);
                                CommonTokenStream tokens_p = new CommonTokenStream (lexer_p);
                                parser_p=new MathML_Presentation_Parser (tokens_p);
                                parser_p.setController (controller);
                                parser_p.setList (parsing_errors);
                                parser_p.mathml ();
                                boxes=parser_p.getBoxes ();
                                break;
                   case 2:
                                lexer_c=new MathML_Content_Lexer (input);
```

```
                        lexer_c.setList (lexing_errors);
                        CommonTokenStream tokens_c=new CommonTokenStream (lexer_c);
                        parser_c=new MathML_Content_Parser (tokens_c);
                        parser_c.setController (controller);
                        parser_c.setList (parsing_errors);
                        parser_c.mathml ();
                        boxes=parser_c.getBoxes ();
                        break;
                }
                if (lexing_errors.size ()! =0 || parsing_errors.size ()! =0)
                 {

                        Equation_Editor_Error_Form form=new Equation_Editor_Error_Form ();
                        LinkedList<String> errors=form.get_List_Errors ();
                        for (int i=0; i < lexing_errors.size (); i++)
                                errors.add (lexing_errors.get (i));
                        for (int i=0; i < parsing_errors.size (); i++)
                                errors.add (parsing_errors.get (i));
                        form.setVisible (true);
                 }
                 else
                 {
                        for (int i=0; i<boxes.size (); i++)
                          controller.addBox (boxes.get (i));
                 }

                }
                catch (OutOfMemoryError er)
                {
                        JOptionPane message=new JOptionPane ();
                        message.showMessageDialog (null, "The file could not be imported because
there
                        is not enough memory.","Equation Editor Error", 0);
                }
                catch (Throwable e) {}
        }
}
```

4.2.2 Código fuente de las reglas gramaticales del analizador léxico Equation_Editor_Format

A continuación se muestran las reglas gramaticales correspondientes al fichero de gramáticas del analizador léxico para el lenguaje Formato Interno del Editor de Ecuaciones. Cada regla está escrita en notación **EBNF** y en ellas se combinan tanto símbolos terminales como no terminales. Por ejemplo, la primera regla del fichero:

OP_ADD: A D D;

Chequea todas las posibles combinaciones que se puedan dar con letras mayúsculas o minúsculas a y d en el orden que aparezcan, con el objetivo de lograr que el análisis léxico no distinga entre mayúsculas y minúsculas y siempre permita escanear sentencias como: ¨**ADD**¨, ¨**add**¨**,** ¨**Add**¨**,** etc
Por la misma regla y así sucesivamente se hace para el resto de las sentencias válidas del lenguaje. Como puede notarse en cada una de estas reglas léxicas se utilizan reglas auxiliares, en este caso la regla **A** y **D**, además existen reglas como **ID, NUMERO, ESPACIO_BLANCO** y **COMMENT** utilizadas con para otros propósitos como pueden ser el escaneo de identificadores o números.

```
//LEXER RULES
OP_ADD: A D D;
OP_MINUS: M I N U S;
OP_MULT: M U L T;
OP_DIV: D I V;
OP_EQUALS: E Q U A L S;
OP_EQUATION: E Q U A T I O N;
OP_LEFTHAND: L E F T H A N D;
OP_FUNCTION: F U N C T I O N;
OP_DEFINEDINTEGRAL: D E F I N E D I N T E G R A L;
OP_DIFFERENTIALEQUATION: D I F F E R E N T I A L E Q U A T I O N;
OP_DERIVE: D E R I V E;
OP_DIFF: D I F F;
OP_EXP: E X P;
OP_FACTORIAL: F A C T O R I A L;
OP_LIM: L I M;
OP_ROOT: R O O T;
OP_SUBINDEX: S U B I N D E X;
OP_SUMMATOR: S U M M A T O R;
OP_UNDEFINEDINTEGRAL: U N D E F I N E D I N T E G R A L;
FUNC_LOG: (L N)| (L G);
FUNC_LOG_N: L O G;
FUNC_TRIG:
(
```

```
 (S I N)|(C O S)|(T A N)|(C O T)|(S E C)|(C S C)|(S I N H)|(C O S H)|(T A N H)|(C O T H)|(S E C H)|(C S C H)| (A R C S I N)|(A R C C O S)|(A R C T A N)|(A R C C O T)|(A R C S E C)|(A R C C S C)|(A R C) |(S I N H)|(A R C C O S H)| (A R C T A N H)|( A R C C O T H)|(A R C S E C H)|(A R C C S C H)
);
//AUXILIARY RULES
fragment A:('a'| 'A');
fragment B:('b'| 'B');
fragment C:('c'| 'C');
fragment D:('d'| 'D');
fragment E:('e'| 'E');
fragment F:('f'| 'F');
fragment G:('g'| 'G');
fragment H:('h'| 'H');
fragment I:('i'| 'I');
fragment J:('j'| 'J');
fragment K:('k'| 'K');
fragment L:('l'| 'L');
fragment M:('m'| 'M');
fragment N:('n'| 'N');
fragment O:('o'| 'O');
fragment P:('p'| 'P');
fragment Q:('q'| 'Q');
fragment R:('r'| 'R');
fragment S:('s'| 'S');
fragment T:('t'| 'T');
fragment U:('u'| 'U');
fragment V:('v'| 'V');
fragment W:('w'| 'W');
fragment X:('x'| 'X');
fragment Y:('y'| 'Y');
fragment Z:('z'| 'Z');

//OPERATORS
```

```
PARENT_ABIERTO: '(';
PARENT_CERRADO: ')';
OP_COMA:',';
ID: (LETRA | '_') (CHAR)*;
NUMERO: '-'? (DIGITO? ('x'| 'X'))? (DIGITO| ('a'.. 'f')|('A'..'F'))+ ('.'(DIGITO)+)* ('e'(DIGITO)+)?;
fragment CHAR: LETRA | DIGITO | '_';
fragment DIGITO: '0'..'9';
fragment LETRA: 'a'..'z'| 'A'..'Z';
ESPACIO_BLANCO: (' '|'\r'|'\t'|'\u000C'|'\n') {skip();};
COMMENT: '//' (~('\n'|'\r'))* {skip();};
```

4.2.3 Código fuente de las reglas gramaticales del analizador sintáctico Equation_Editor_Format

A continuación se muestran algunas de las principales reglas gramaticales correspondientes al fichero de gramáticas del analizador sintáctico para el lenguaje Formato Interno del Editor de Ecuaciones. Cada regla está escrita en notación **EBNF** y en ellas se combinan tanto símbolos terminales como no terminales.

Cada regla tiene asociado un grupo de acciones propias del código nativo, en este caso java, con el objetivo de ir generando el árbol de sintaxis abstracta de cada tipo de expresión que describe la regla; además como puede observarse cada regla, excepto el axioma de la gramática, devuelve una variable de tipo BasicBox que contiene la representación de la expresión de la cual se derivó. Esta variable es utilizada posteriormente en las acciones que se llevan a cabo por su regla predecesora para la conformación del árbol de sintaxis abstracta. Todo lo explicado anteriormente se evidencia, por ejemplo, con las siguientes reglas:

```
elemento_simple returns [BasicBox box]:
( vbox=variable{box=vbox;} | num=numero{box=num;} );
variable returns [BasicBox box]: ID
{
  box=new SimplexBox("",controller);
  String text=$ID.text;
  ((SimplexBox)box).setValue(text);
};
```

Cuando la regla **elemento_simple** derive por la regla **variable**, devolverá la variable que retorna la regla **variable**, en este caso una caja simple que contiene el texto escaneado por la regla **ID** y así ocurrirá sucesivamente con la regla que derivó por **elemento_simple**.

```
//GRAMATICA
programa: (sent=sentencia{list.add(sent);})* EOF;
//SENTENCIA
sentencia returns [BasicBox box]: (eq=ecuacion{box=eq;}|func=funcion{box=func;});
//ELEMENTO BASICO
elemento_simple returns[BasicBox box]:
(  vbox=variable{box=vbox;} |num=numero{box=num;} );
//VARIABLE
variable returns[BasicBox box]:ID
{
  box=new SimplexBox("",controller);
  String text=$ID.text;
  ((SimplexBox)box).setValue(text);
};
//NUMERO
numero returns [BasicBox box]:NUMERO
{
  box=new SimplexBox("",controller);
  String text=$NUMERO.text;
  ((SimplexBox)box).setValue(text);
};
//ARGUMENTO
argumento_funcion returns [BasicBox box] @init{ box=new ArgumentBox("",controller); } :
f=factor ( OP_COMA v=factor {((ArgumentBox)box).addBox(v);})*
{
  ((ArgumentBox)box).setBoxAt(0,f);
};
//FUNCIONES
```

```
funcion_elemental returns[BasicBox box]:
f=funcion_trigonometrica{box=f;}|f=funcion_logaritmica{box=f;}| f=funcion_exponencial{box=f;}|
f=funcion_radical{box=f;}| f=funcion_id{box=f;}| f=factorial{box=f;}| s=subindice{box=s;};
//FUNCION EXPONENCIAL
funcion_exponencial returns[BasicBox box]: OP_EXP PARENT_ABIERTO base=factor OP_COMA
exp=factor PARENT_CERRADO
{
    box=new ExponentialBox("",controller);
    ((ExponentialBox)box).setPrimaryBox(base);
    ((ExponentialBox)box).setSecondaryBox(exp);
};
//FUNCION RADICALICA
funcion_radical returns[BasicBox box]: OP_ROOT PARENT_ABIERTO base=factor OP_COMA
grado=factor PARENT_CERRADO
{
    box=new RootBox("",controller);
    ((RootBox)box).setPrimaryBox(base);
    ((RootBox)box).setSecondaryBox(grado);
};
//ECUACIONES
ecuacion returns[BasicBox
box]:ec=ecuacion_algebraica{box=ec;}|e=ecuacion_diferencial{box=e;};
//ECUACION ALGEBRAICA
ecuacion_algebraica returns[BasicBox box]: OP_EQUATION PARENT_ABIERTO fact1=factor
OP_COMA fact2=factor PARENT_CERRADO
{
     box=new EquationBox("",controller);
     ((EquationBox)box).setPrimaryBox(fact1);
     ((EquationBox)box).setSecondaryBox(fact2);
};
//ECUACION DIFERENCIAL
ecuacion_diferencial returns[BasicBox box]: OP_DIFFERENTIALEQUATION PARENT_ABIERTO
exp=expresion_diferencial OP_COMA fact=factor PARENT_CERRADO
```

```
{
    box=new DifferentialEquationBox("",controller);
    ((DifferentialEquationBox)box).setPrimaryBox(exp);
    ((DifferentialEquationBox)box).setSecondaryBox(fact);
};
//FUNCION
funcion returns[BasicBox box]: OP_FUNCTION PARENT_ABIERTO func=funcion_id OP_COMA
fact=factor PARENT_CERRADO{
    box=new FunctionBox("",controller);
    ((FunctionBox)box).setPrimaryBox(func);
    ((FunctionBox)box).setSecondaryBox(fact);
};
```

4.2.4 Código fuente de la clase controladora Equation_Editor_Format_Lexer

A continuación se muestra el código de algunos de los principales métodos de la clase Equation_Editor_Format_Lexer que es la encargada de efectuar el proceso de análisis léxico para los archivos que contienen código en formato interno del editor de ecuaciones.

```
public final void mOP_ADD() throws RecognitionException {
    try {
        int _type = OP_ADD;
        int _channel = DEFAULT_TOKEN_CHANNEL;
        {
            mA(); if (state.failed) return ;
            mD(); if (state.failed) return ;
            mD(); if (state.failed) return ;
        }
        state.type = _type;
        state.channel = _channel;
    }
    finally { }
}
```

Este método chequea la operación definida en las reglas gramaticales correspondiente al analizador léxico para escanear al operador **OP_ADD**.

```
public final void mOP_MINUS() throws RecognitionException {
    try {
        int _type = OP_MINUS;
        int _channel = DEFAULT_TOKEN_CHANNEL;
        {
          mM(); if (state.failed) return ;
          mI(); if (state.failed) return ;
          mN(); if (state.failed) return ;
          mU(); if (state.failed) return ;
          mS(); if (state.failed) return ;
        }
        state.type = _type;
        state.channel = _channel;
    }
    finally
   { }
}
```

Este método chequea la operación definida en las reglas gramaticales correspondiente al analizador léxico para escanear al operador **OP_MINUS**.

```
public final void mOP_EQUATION() throws RecognitionException {
    try {
        int _type = OP_EQUATION;
        int _channel = DEFAULT_TOKEN_CHANNEL;
        {
          mE(); if (state.failed) return ;
          mQ(); if (state.failed) return ;
          mU(); if (state.failed) return ;
          mA(); if (state.failed) return ;
          mT(); if (state.failed) return ;
          mI(); if (state.failed) return ;
```

```
            mO(); if (state.failed) return ;
            mN(); if (state.failed) return ;
          }
          state.type = _type;
          state.channel = _channel;
      } finally
      {}
}
```

Este método chequea la operación definida en las reglas gramaticales correspondiente al analizador léxico para escanear al operador **OP_EQUATION**.

Para todas las reglas definidas en el fichero de gramática correspondiente a este tipo de analizador se genera un código fuente muy similar solo varía en dependencia del tipo de sentencia.

4.2.5 Código fuente de la clase controladora Equation_Editor_Format_Parser

A continuación se muestra el código de algunos de los principales métodos de la clase Equation_Editor_

Format_Parser que es la encargada de efectuar el proceso de análisis sintáctico para los archivos que contienen código en formato interno del editor de ecuaciones.

```
public final void programa() throws RecognitionException {
      BasicBox sent = null;
      try {
          (sent= sentencia )* EOF
          {
           loop1:
           do {
           int alt1=2;
           int LA1_0 = input.LA(1);
           if((LA1_0==OP_EQUATION||LA1_0==OP_FUNCTION||
              LA1_0==OP_DIFFERENTIALEQUATION) ) {
              alt1=1;
           }
```

```
            switch (alt1) {
                case 1 :
                    sent= sentencia
                    {
                      pushFollow(FOLLOW_sentencia_in_programa55);
                      sent=sentencia();
                      state._fsp--;
                      if (state.failed) return ;
                      if ( state.backtracking==0 ) {
                        list.add(sent);
                      }
                    }
                    break;
                default :
                    break loop1;
            }
        }
        while (true);
          match(input,EOF,FOLLOW_EOF_in_programa60); if (state.failed) return ;
        }
    }
    catch(Exception exc)
    {}
    finally
    {}
    return ;
}
```

Este método corresponde al axioma de la gramática, en este caso a la regla programa, donde se chequea por cual regla hacer la derivación correcta.

```
public final BasicBox sentencia() throws RecognitionException {
    BasicBox box = null;
```

```
BasicBox eq = null;
BasicBox func = null;
try
{
  {
    int alt2=2;
    int LA2_0 = input.LA(1);
    if ( (LA2_0==OP_EQUATION||LA2_0==OP_DIFFERENTIALEQUATION) ) {
      alt2=1;
    }
    else if ( (LA2_0==OP_FUNCTION) ) {
      alt2=2;
    }
    else {
      if (state.backtracking>0)
      {
        state.failed=true;
        return box;
      }
      NoViableAltException nvae = new NoViableAltException("", 2, 0, input);
      throw nvae;
    }
    switch (alt2)
    {
      case 1 :
        {
          pushFollow(FOLLOW_ecuacion_in_sentencia75);
          eq=ecuacion();
          state._fsp--;
          if (state.failed)
            return box;
          if ( state.backtracking==0 )
          {
```

```
                box=eq;
              }
            }
            break;
          case 2 :
            {
              pushFollow(FOLLOW_funcion_in_sentencia80);
              func=funcion();
              state._fsp--;
              if (state.failed)
                 return box;
              if ( state.backtracking==0 )
              {
                box=func;
              }
            }
            break;
        }
    }
    catch(Exception exc)
    {
    }
    finally
    {
    }
    return box;
  }
```

Este método corresponde a la regla sintáctica sentencia y en el se decide por cuál variante de sentencia derivar, estas pueden ser lo mismo que una ecuación algebraica, una diferencial o una función.

4.3 Principales interfaces

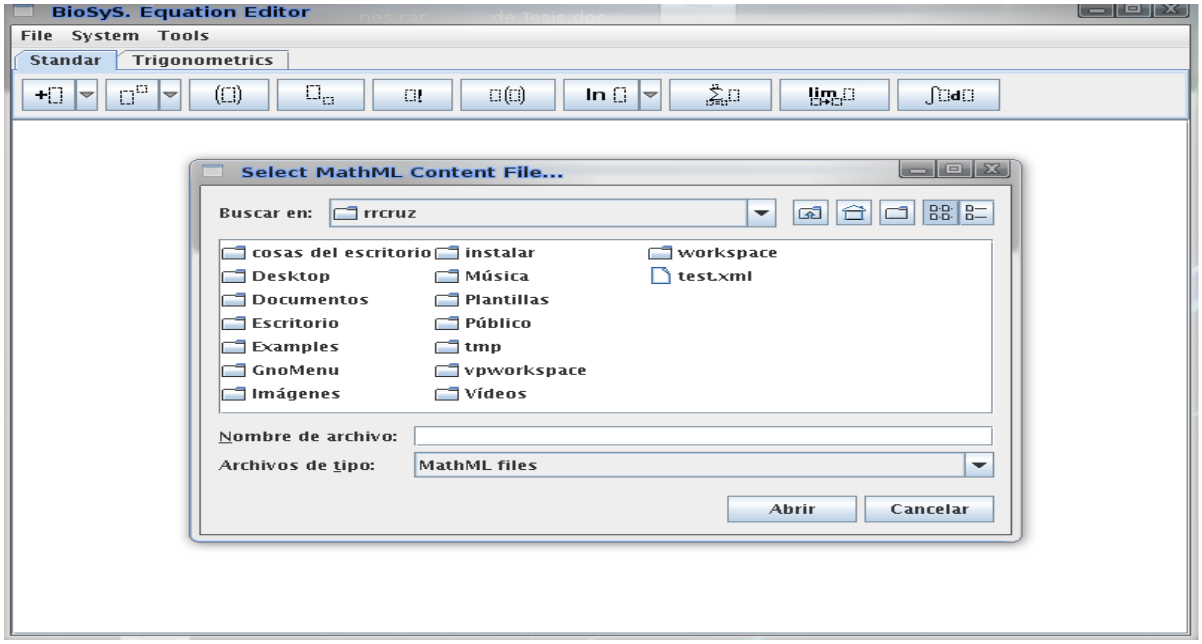


Fig. 15: Interfaz para importar archivos MathML Content.

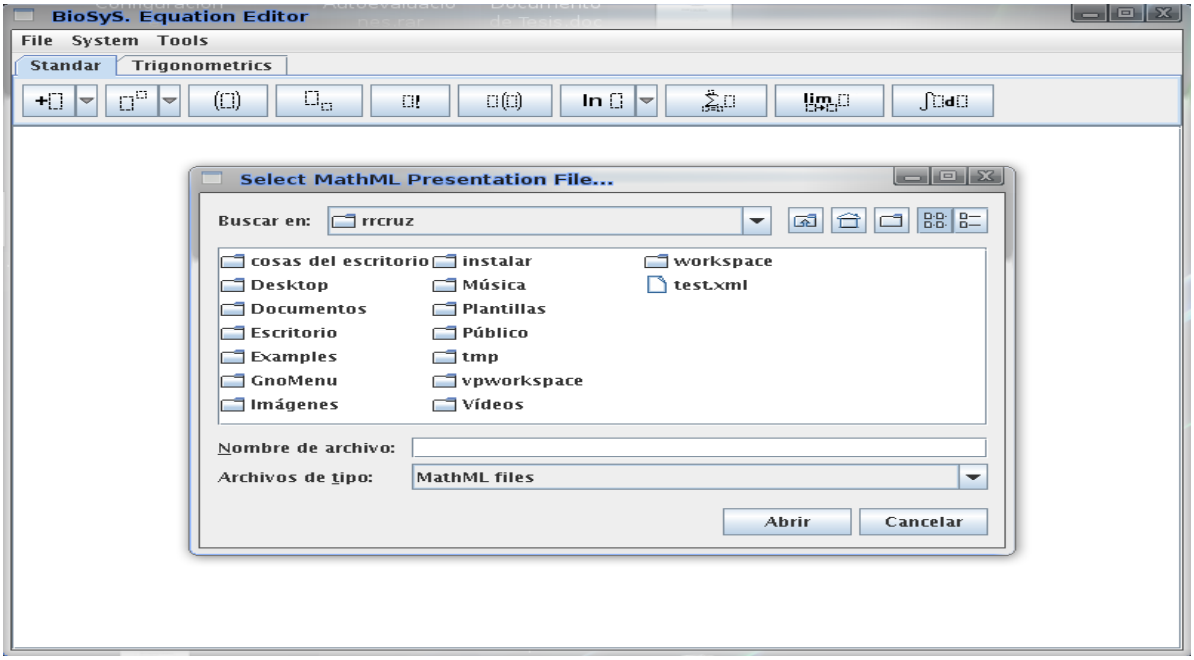


Fig. 16: Interfaz para importar archivos MathML Presentation.

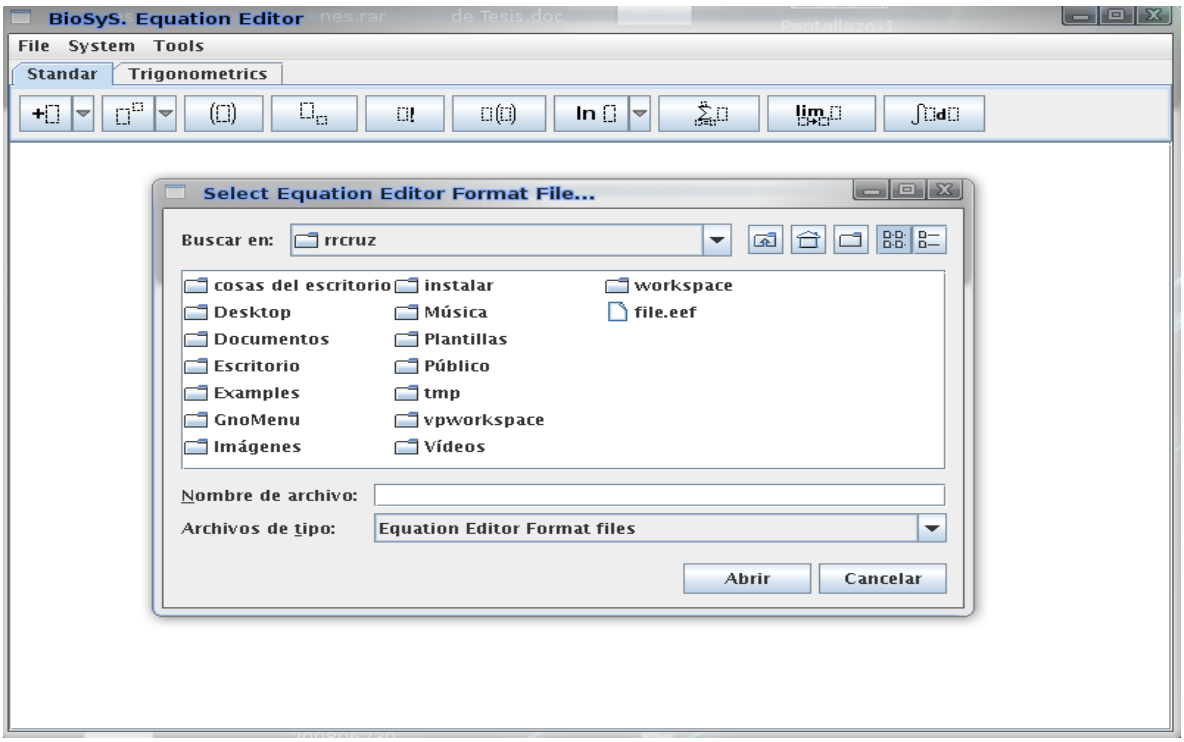


Fig. 17: Interfaz para importar archivos Formato Interno del Editor de Ecuaciones.

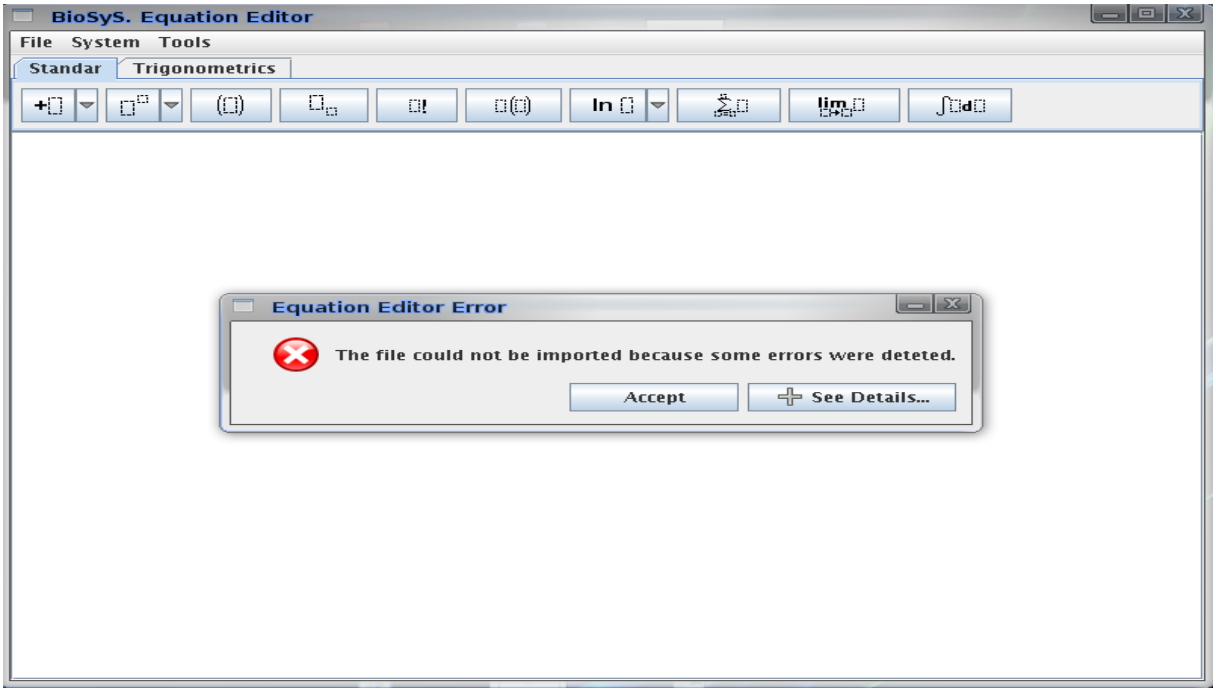


Fig. 18: Interfaz de error en el proceso de importación.

4.4 Validación Funcional

A continuación se describen los casos de prueba realizados sobre las interfaces que implementan las funcionalidades para comprobar que se cumplen con los requisitos funcionales.

4.4.1 Modelo de Prueba

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. La prueba de software es un elemento crítico para lograr obtener mejoras en la calidad y representa una revisión final de las especificaciones del diseño y de la codificación. Uno de los objetivos de la fase de pruebas es verificar que el comportamiento del sistema satisfaga los requisitos establecidos por los clientes y futuros usuarios. [19]

A medida que los sistemas sean más complejos y aumente la demanda de calidad, se hacen necesarios procesos y métodos que la garanticen. La etapa de prueba es tan o más importante que todas las realizadas hasta el momento puesto que en ella se refleja la calidad con que ha sido llevada a cabo la proyección del sistema. 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. [19]

Como la aplicación desarrollada es una nueva funcionalidad correspondiente al módulo del Editor de Ecuaciones, el tipo de prueba realizado corresponde al de Prueba de Unidad con el objetivo de comprobar que esta cumpla correctamente los requisitos funcionales establecidos. 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.

4.4.2 Casos de prueba Importar Archivo MathML Content

Con el objetivo de comprobar el correcto funcionamiento del sistema se realizaron diferentes casos de prueba, a continuación se describen:

Caso de uso	Importar Archivo
Caso de prueba	Importar Archivo en formato MathML Content
Entrada	Inicialmente el investigador selecciona la opción de Importar Archivo MathML

	Content, luego selecciona el archivo test.xml ubicado en el escritorio, este contendrá la expresión matemática: ¨**a+b=c+d**¨ en formato MathML Content (Ver anexo 8), y finalmente selecciona la opción aceptar. (Ver figura 19).
Resultado esperado	Se visualiza la expresión: ¨**a+b=c+d**¨ en el área de edición del editor de ecuaciones. (Ver figura 20)
Resultado de la prueba	Después de efectuar el proceso de importación del archivo con la expresión: ¨**a+b=c+d**¨ en formato MathML Content, esta fue visualizada correctamente en el área de edición, por lo que el resultado de la prueba fue el esperado.
Condiciones	El archivo a importar debe tener una de las siguientes extensiones: xml, mml o txt. En caso de que se seleccione un archivo con una extensión que no sea una de las anteriores se debe mostrar un cartel de error, similarmente ocurre si el contenido del archivo es incorrecto, o sea, se debe mostrar un mensaje de error indicando los posibles errores.

Tabla 8: Caso de Prueba Importar Archivo en formato MathML Content.

Entrada del caso de prueba: ¨**Importar Archivo en formato MathML Content**¨

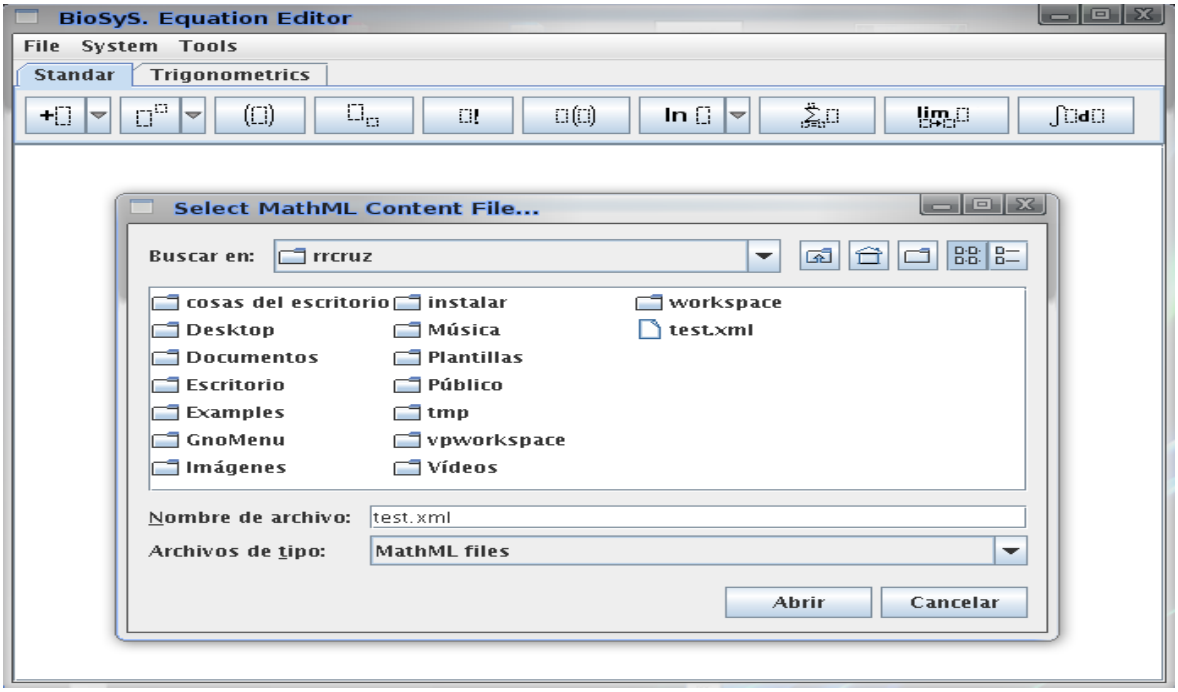


Fig. 19: Entrada del caso de prueba ¨Importar Archivo en formato MathML Content¨.

Resultado del caso de prueba: ¨**Importar Archivo en formato MathML Content**¨.

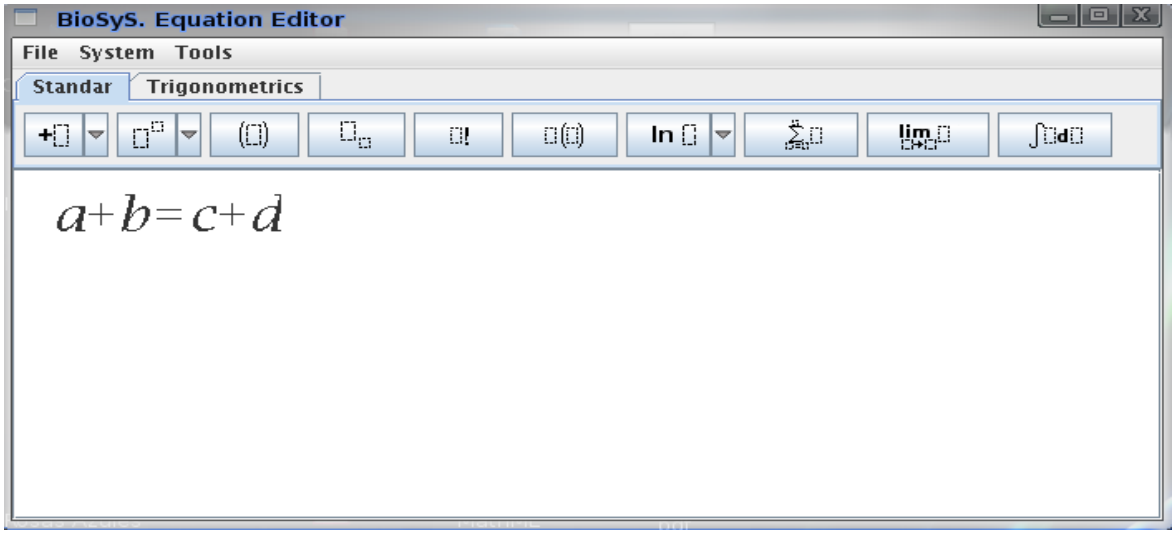


Fig. 20: Resultado del caso de prueba ¨Importar Archivo en formato MathML Content¨.

Caso de uso	Importar Archivo
Caso de prueba	Importar Archivo MathML Content de extensión incorrecta
Entrada	Inicialmente el investigador selecciona la opción de Importar Archivo MathML Content, luego selecciona el archivo test.doc ubicado en el escritorio y finalmente selecciona la opción aceptar. (Ver figura 21).
Resultado esperado	El editor de ecuaciones muestra un mensaje de error indicando que la extensión es incorrecta. (Ver figura 22).
Resultado de la prueba	Después de efectuar el proceso de importación del archivo test.doc, el editor de ecuaciones muestra mensaje de error indicando que la extensión es incorrecta, por lo que el resultado de la prueba fue el esperado.
Condiciones	El archivo a importar no debe tener una de las siguientes extensiones: xml, mml o txt.

Tabla 9: Caso de Prueba Importar Archivo MathML Content de extensión incorrecta.

Entrada del caso de prueba ¨ **Importar Archivo MathML Content de extensión incorrecta** ¨.

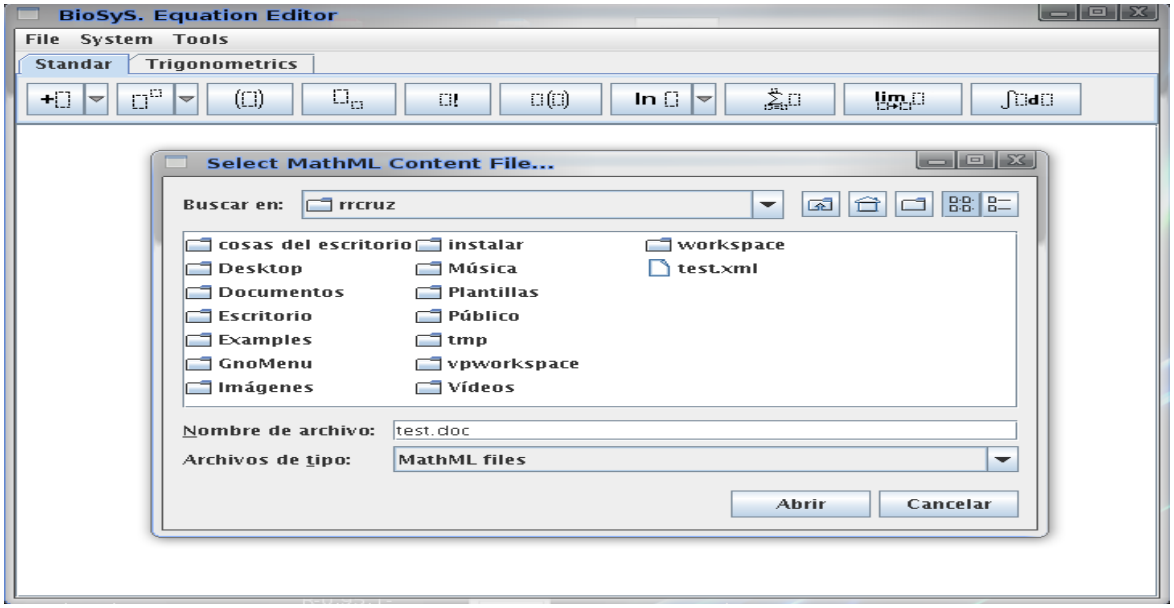


Fig. 21: Entrada del caso de prueba: ¨ Importar Archivo MathML Content de extensión incorrecta¨.

Resultado del caso de prueba ¨ **Importar Archivo MathML Content de extensión incorrecta** ¨.

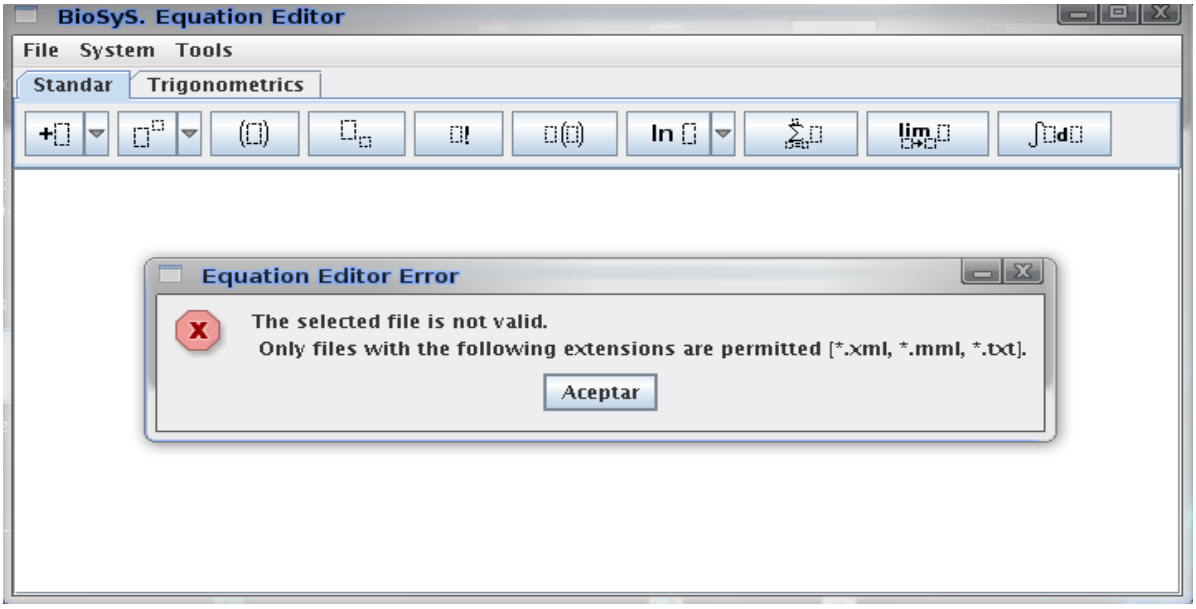


Fig. 22: Resultado del caso de prueba ¨ Importar Archivo MathML Content de extensión incorrecta ¨.

Caso de uso	Importar Archivo
Caso de prueba	Importar Archivo MathML Content con errores de codificación
Entrada	Inicialmente el investigador selecciona la opción de Importar Archivo MathML Content, luego selecciona el archivo test.xml ubicado en el escritorio, este contendrá la expresión matemática: ¨**a+b=c+d**¨ con errores en su codificación en formato MathML Content (Ver anexo 9), y finalmente selecciona la opción aceptar. (Ver figura 23).
Resultado esperado	El editor de ecuaciones muestra un mensaje de error indicando que se detectaron errores durante el proceso de importación. (Ver figura 24)
Resultado de la prueba	Después de efectuar el proceso de importación del archivo test.xml, el editor de ecuaciones muestra mensaje de error indicando que se detectaron errores durante el proceso de importación, por lo que el resultado de la prueba fue el esperado.
Condiciones	El archivo debe tener una de las siguientes extensiones: xml, mml o txt.

Tabla 10: Caso de Prueba Importar Archivo MathML Content con errores de codificación.

Entrada del caso de prueba ¨ **Importar Archivo MathML Content con errores de codificación** ¨.

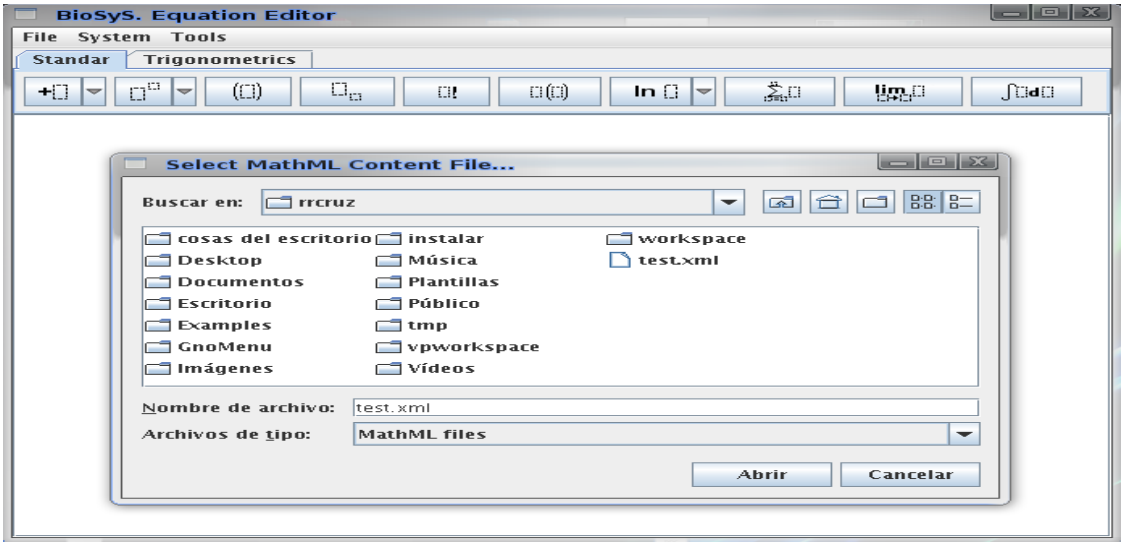


Fig. 23: Entrada del caso de prueba: Importar Archivo MathML Content con errores de codificación.

Resultado del caso de prueba ¨ **Importar Archivo MathML Content con errores de codificación** ¨.

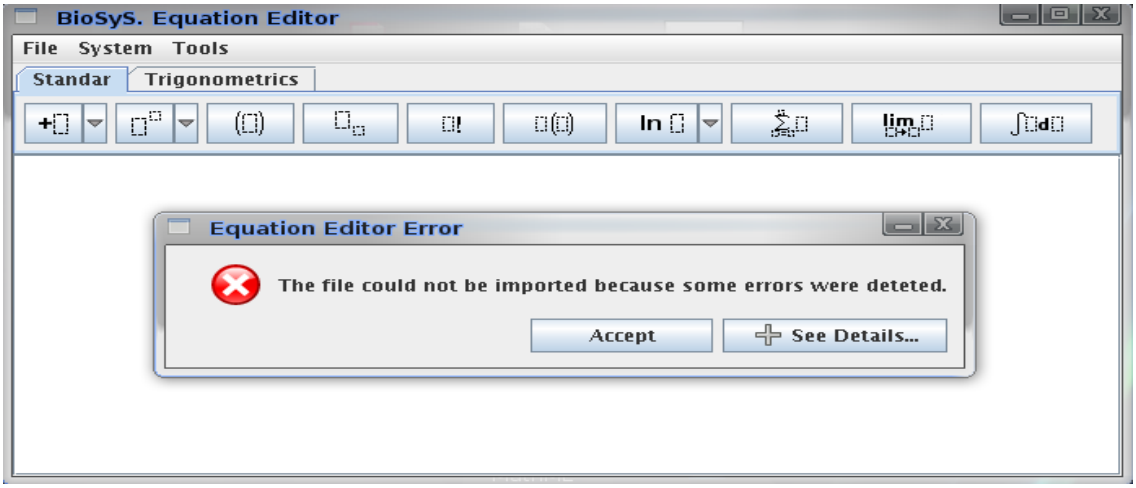


Fig. 24: Resultado del caso de prueba ¨ Importar Archivo MathML Content con errores de codificación ¨.

Caso de uso	Importar Archivo
Caso de prueba	Importar Archivo en formato MathML Presentation
Entrada	Inicialmente el investigador selecciona la opción de Importar Archivo MathML Presentation, luego selecciona el archivo test.xml ubicado en el escritorio, este contendrá la expresión matemática: ¨**a+b=c+d**¨ en formato MathML Presentation (Ver anexo 10), y finalmente selecciona la opción aceptar. (Ver figura 25).
Resultado esperado	Se visualiza la expresión: ¨**a+b=c+d**¨ en el área de edición del editor de ecuaciones. (Ver figura 26)
Resultado de la prueba	Después de efectuar el proceso de importación del archivo con la expresión: ¨**a+b=c+d**¨ en formato MathML Presentation, esta fue visualizada correctamente en el área de edición, por lo que el resultado de la prueba fue el esperado.
Condiciones	El archivo a importar debe tener una de las siguientes extensiones: xml, mml o txt. En caso de que se seleccione un archivo con una extensión que no sea una de las anteriores se debe mostrar un cartel de error, similarmente ocurre si el contenido del archivo es incorrecto, o sea, se debe mostrar un mensaje

	de error indicando los posibles errores.

Tabla 11: Caso de Prueba Importar Archivo en formato MathML Presentation.

Entrada del caso de prueba: ¨ **Importar Archivo en formato MathML Presentation** ¨.

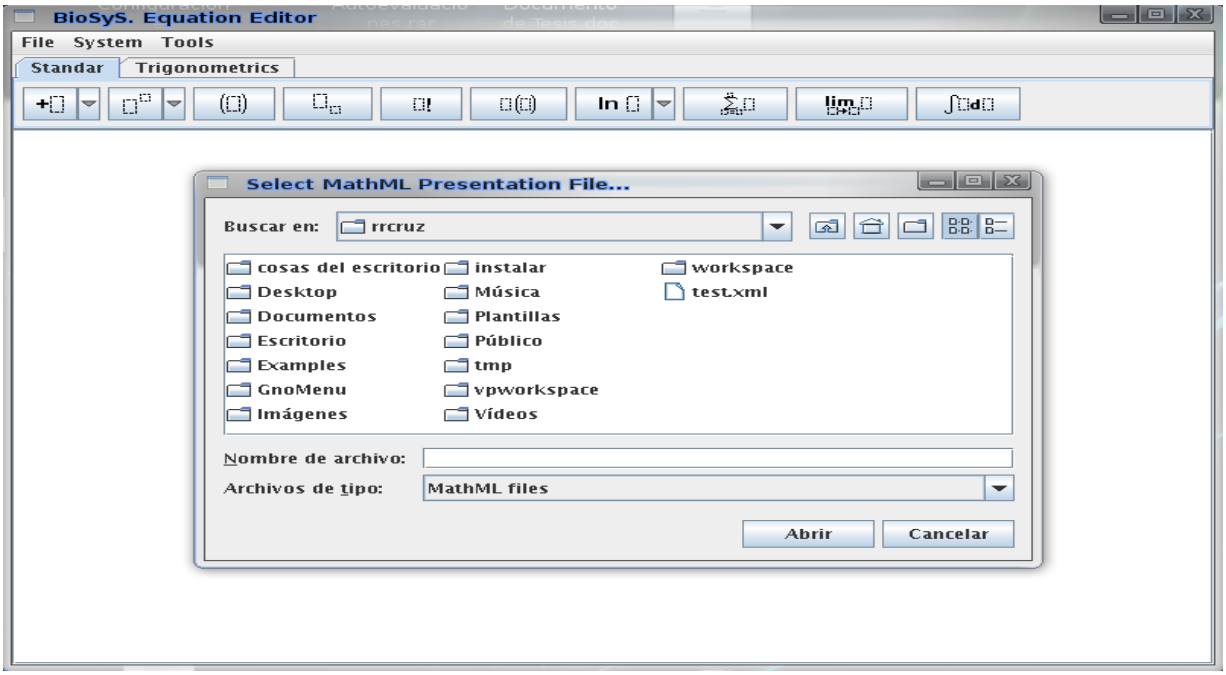


Fig. 25: Entrada del caso de prueba: Importar Archivo en formato MathML Presentation.

Resultado del caso de prueba: ¨ **Importar Archivo en formato MathML Presentation** ¨.

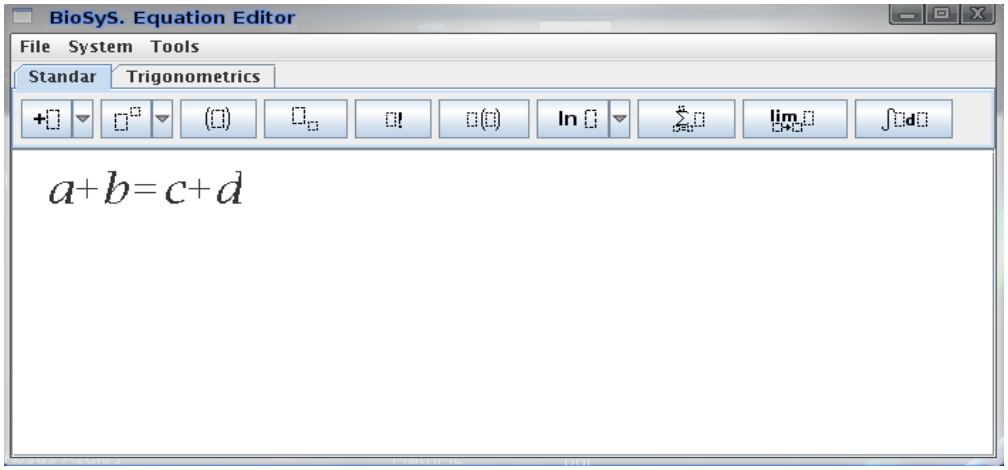


Fig. 26: Resultado del caso de prueba ¨ Importar Archivo en formato MathML Presentation¨.

Caso de uso	Importar Archivo
Caso de prueba	Importar Archivo en Formato Interno
Entrada	Inicialmente el investigador selecciona la opción de Importar Archivo en Formato Interno, luego selecciona el archivo test.eef ubicado en el escritorio, este contendrá la expresión matemática: ¨**a+b=c+d**¨ en Formato Interno (Ver anexo 11), y finalmente selecciona la opción aceptar. (Ver figura 27).
Resultado esperado	Se visualiza la expresión: ¨**a+b=c+d**¨ en el área de edición del editor de ecuaciones. (Ver figura 28)
Resultado de la prueba	Después de efectuar el proceso de importación del archivo con la expresión: ¨**a+b=c+d**¨ en Formato Interno, esta fue visualizada correctamente en el área de edición, por lo que el resultado de la prueba fue el esperado.
Condiciones	El archivo a importar debe tener una de las siguientes extensiones: eef o txt. En caso de que se seleccione un archivo con una extensión que no sea una de las anteriores se debe mostrar un cartel de error, similarmente ocurre si el contenido del archivo es incorrecto, o sea, se debe mostrar un mensaje de error indicando los posibles errores.

Tabla 12: Caso de Prueba Importar Archivo en Formato Interno.

Entrada del caso de prueba ¨ **Importar Archivo en Formato Interno**¨.

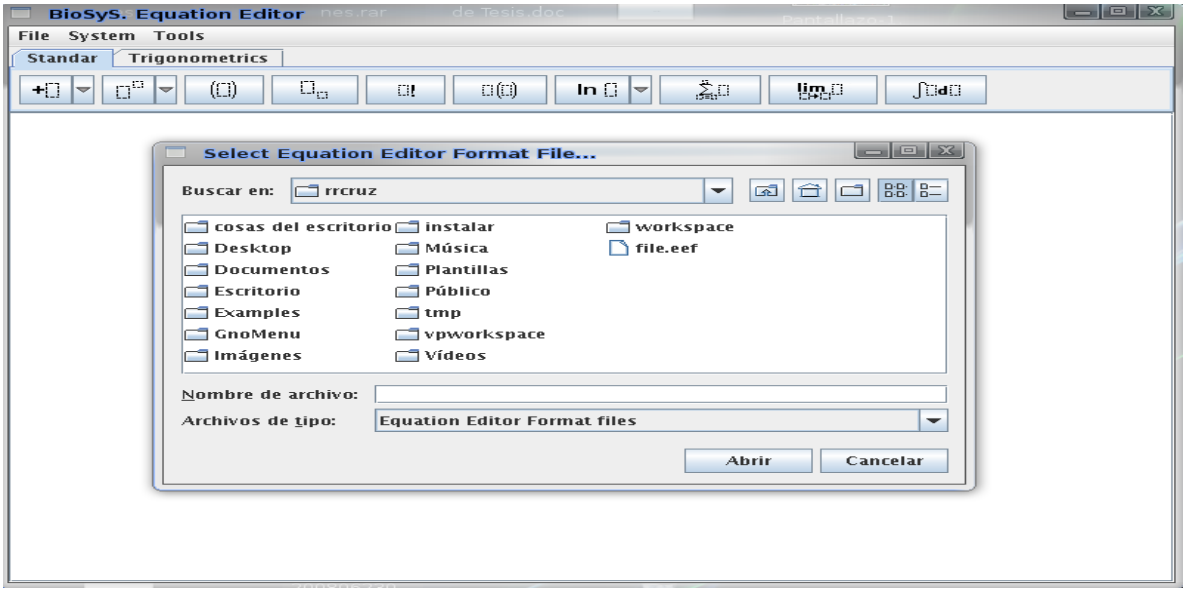


Fig. 27: Entrada del caso de prueba: Importar Archivo en Formato Interno.

Resultado del caso de prueba ¨ **Importar Archivo en Formato Interno**¨.

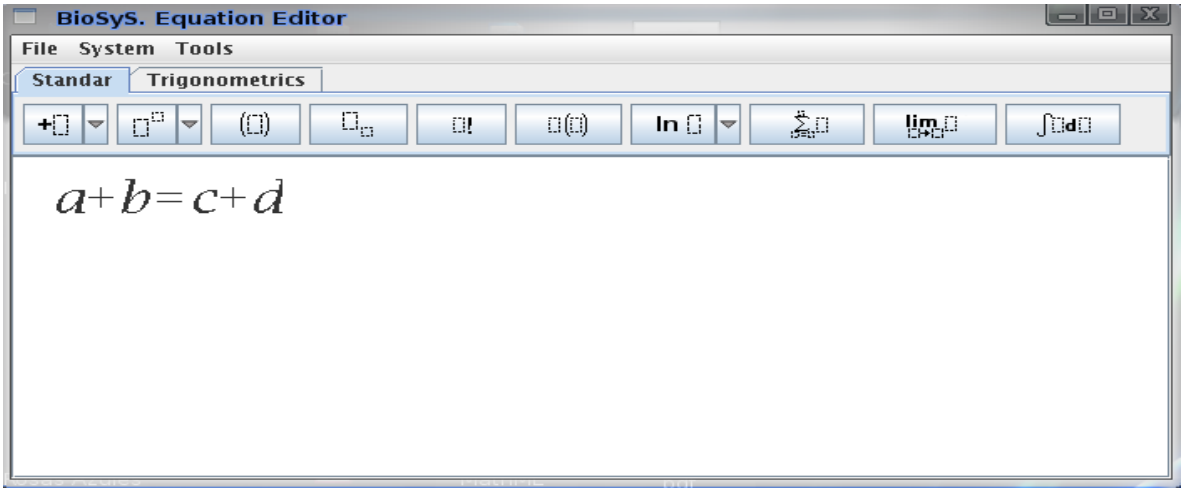


Fig. 28: Resultado del caso de prueba ¨ Importar Archivo en Formato Interno¨.

4.5 Conclusiones

La implementación es una etapa fundamental para lograr materializar todo el proceso de creación de un software, este no es el final, pero es una parte muy importante para todo el equipo de trabajo involucrado. 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 para garantizar que los requisitos funcionales fueron cumplidos.

Conclusiones Generales

Con el desarrollo del presente trabajo a lo largo de sus 4 capítulos se demostró la importancia que revierte para el Editor de Ecuaciones de la Plataforma de Simulación y Análisis de Sistemas Biológicos la adición de nuevas funcionalidades para efectuar el proceso de importación de archivos que contienen modelos matemáticos codificados con los estándares de representación matemática MathML y el lenguaje de descripción Formato Interno del Editor de Ecuaciones, además:

- Se diseñaron las funcionalidades para importar de los estándares de representación de fórmulas matemáticas MathML Content, MathML Presentation y desde el Lenguaje de descripción de expresiones matemáticas Formato Interno del Editor de Ecuaciones.
- Fueron implementadas las funcionalidades para importar de los estándares de representación de fórmulas matemáticas MathML Content, MathML Presentation y desde el Lenguaje de descripción de expresiones matemáticas Formato Interno del Editor de Ecuaciones.
- Se comprobó la manera en que se gestionan los posibles errores que se pueden presentar durante el proceso de importación, permitiendo incluso, conocer a los usuarios la naturaleza de las causas de los errores.
- Y fueron comprobadas las funcionalidades implementadas a través de un grupo de pruebas de caja negra que demostraron que se cumplieron cabalmente los requisitos especificados.

Recomendaciones

Al concluir el presente trabajo el equipo de desarrollo recomienda:

- Implementar gramáticas más abarcadoras para los analizadores de lenguajes desarrollados de modo que permitan escanear todo tipo de expresiones posibles descritas por el estándar de representación matemática MathML en sus dos variantes.
- Realizar manual de ayuda al usuario donde se expliquen cuáles son las expresiones matemáticas que actualmente pueden ser representadas en el editor de ecuaciones, cómo son exportadas en el formato MathML y cómo deben estar codificadas para que puedan ser importadas correctamente.

Referencias Bibliográficas

1. Barzanallana, Rafael. Universidad de Murcia. *Capítulo 4. Historia de la Informática.* [En línea] 26 de 11 de 2008. [Citado el: 1 de 01 de 2009.] http://www.um.es/docencia/barzana/II/Ii04.html.

2. Pérez Ayup, Noel y Arnaiz Rey, Anay.*La informatización de la sociedad.* 2001. Pag 3.

3. Centelles Rubiera, Ileana y Céspedes Martínez, José Luis. *Software para la Simulación de Sistemas Biológicos. Módulo "Editor de Ecuaciones Diferenciales".* Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2007. Pág. 11-12.

4. Lueno Diez, Candida, Ortin Soler, Francisco y Labra Gallo, Jose Emilio. Universidad de Oviedo. Enero 2005. *Cuaderno Sintáctico: Análisis Sintáctico en Procesadores de Lenguajes.* Pags 20-28.

5. Gracia, Joaquin. *Diseño de Software Orientado a Objetos.* [En línea] 27 de 2 de 2009. [Citado el: 10 de 04 de 2009.] http://www.ingenierosoftware.com/analisisydiseno/patrones-diseno.php.

6. Sitio de la Asignatura Ingeniería de Software I. *Conferencia 5: Profundización en el Flujo de Trabajo de Requerimientos.* [En línea] [Citado el: 23 de 3 de 2009] http://teleformacion.uci.cu.

7. Pérez de Ovalles, María A. UNIVERSIDAD SIMÓN BOLÍVAR. *SISTEMAS DE INFORMACIÓN II.* [En línea] [Citado el: 23 de 4 de 2009.] http://prof.usb.ve/lmendoza/Documentos/PS-6/Teor%EDa%20PS6116%20Arq.%20de%20Software.pdf.

8. OpenUP. [En línea] [Citado el: 2 de 5 de 2009.] http://10.34.20.5:5800/OpenUP/.

9. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El Lengaje Unificado de Modelado.Manual de Referencia.* Massachusetts,USA : s.n., 2000.

10. Arias Naranjo, Gilberto. Tesis de Maestría**:** *Editor de ecuaciones para la Plataforma de Simulación de Sistemas Biológicos.* Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2007. Pág. 26-27.

11. Ramírez, Alejandro. *Subversion 2004*. [En línea] [Citado el: 23 de febrero 209] http://polaris.dit.upm.es/~rubentb/docs/subversion/TutorialSubversion/index.html

12. Larman, Craig. *UML y Patrones: Introducción al análisis y diseno orientado a objetos.* Mexico : Prentice Hall, pag 87., 1999.

13. Ramirez, Karel. Plataforma Computacional pra el desarrollo de la Biologia de Sistemas. *Trabajo de diploma ,Facultad de Matematica_Computacion.* Ciudad de La Habana : s.n., 2004.

14. Len, Clements, Paul y Kazman, Rick Bass. *Software Architecture in Practice.* Hardcover : s.n., 2003.

15. Lago, Ramiro. Patrones de diseño software. *Patrón "Modelo-Vista-Controlador".* [En línea] 23 de 4 de 2007. [Citado el: 4 de 5 de 2009.] http://www.proactiva-calidad.com/java/patrones/index.html/algunos_patrones.

16. Gamma, Erich, Helm, Richard y Johnson, Ralph. *Design Patterns: Elements of Reusable Object-Oriented Software.* s.l. : Addison-Wesley, 1995. Pag 34.

16. Otero, Andres. *El Proceso Unificado de Desarrollo de Software.* s.l. : Fareso,SA, 2000.Pag 55.

17. Rambaugh, James,Jacobson,Booch. *El Lenguaje unificado del modelado. Manual de Referencia.*

18. Martínez Figueredo, Alfredo y Luna Gallardo, Arisbel. *BioSyS: Módulo de Modelación gráfica de Sistemas Biológicos.* Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2008. Pág. 54.

19. Paz Noa, Eliane. *Diseño y aplicación de pruebas al producto "Árbol Genealógico.".* Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2007. Pág. 20.

Bibliografía

1. Visual Paradigm. [En línea: 20 de feb. De 2009] [Consultado el: 20 de feb. de 2009]. Disponible en: http://www.visual-paradigm.com

2. Letelier, Patricio y Penadés, M. C. *Metodologías ágiles para el desarrollo de software:(OpenUP)* .Universidad Politécnica de Valencia. 2000: [En línea: 20 de feb. De 2009] [Consultado el: 23 de feb. de 2008]. Disponible en: http://www.willydev.net/descargas/masyxp.pdf

3. Mendoza Sánchez, María. *Metodologías De Desarrollo De Software*. [Consultado el: 5 de abril de 09] Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf.

4. JACOBSON, IVAR; BOOCH, GRADY Y RUMBAUGH, JAMES. *El proceso unificado de software*. Primera edición. Pearson Educación, S.A. Año: 2000.

5. *W3C Math Home*. *W3C.* [En línea] [Consultado el: 5 de nov. de 08] W3C. http://www.w3.org/Math/.

6. Aho, Alfred V., Sethi, Ravi y Ullman, Jeffrey D. *Compilers: Principles, Techniques and Tools.* s.l.: Addison-Wesley, 1986.

7. Freeman, Eric y Freeman, Elisabeth. *Head First Design Patterns.* s.l.: O'Reilly, 2004.

8. Miner, Robert. *The Importance of MathML to Mathematics Communication.* 5, 2005, Notice of the AMS, Vol. 52, Págs. 532-538.

9. Larman, Craig. 1999. *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* Upper Saddle River, EUA: Prentice Hall, 1999.

10. Medina Pasaje, Julio Luis. Marzo 7, 2006. *Metodología y herramientas UML para el modelado y análisis de sistemas de tiempo real orientados a objetos.* Madrid, España: s.n., Marzo 7, 2006

11. Reynoso, Carlos y Kiccillof, Nicolás. 2004. *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. s*.l: Universidad de Buenos Aires, 2004.

12. Sitio de la Asignatura Ingeniería de Software I. Conferencia 4: *FT Requerimientos*. [En línea] [Consultado el: 5 de feb. de 09] Disponible en: http://teleformacion.uci.cu.

13. Sitio de la Asignatura Ingeniería de Software I. Conferencia 7: FT Análisis de Diseño. [En línea] [Consultado el: 29 de enero de 09] Disponible en: http://teleformacion.uci.cu.

14. Sitio de la Asignatura Ingeniería de Software II. Conferencia 2: Arquitectura y Patrones de Diseño. [En línea] [Consultado el: 12 de marzo de 09] Disponible en: http://teleformacion.uci.cu.

15. Sitio de la Asignatura Ingeniería de Software II. Conferencia 4: *FT Implementación. Modelo de implementación.* [En línea] [Consultado el: 5 de marzo de 09] Disponible en: http://teleformacion.uci.cu.

16. Sitio de la Asignatura Ingeniería de Software II. Conferencia 5: *FT Prueba.* [En línea] [Consultado el: 15 de abril de 09] Disponible en: http://teleformacion.uci.cu.

17. Brena, Ramón. *AUTOMATAS Y LENGUAJES: un enfoque de diseño.* Tecnológico de Monterrey, Verano 2003, 1ra edición.

18. Pressman, Roges S, 2005. *Ingeniería de Software un Enfoque Práctico*. 2005, 5ta edición, 589 páginas, [En línea] [Consultado el: 12 de feb. de 09] Disponible en: http://biblioteca.uci.cu/bives/titdigitales.

19. Sitio de la Asignatura Programación IV. Tema 1, Teoría de Lenguajes y Compilación, Libros y Manuales, semántico.pdf. [En línea] [Consultado el: 12 de marzo de 09] Disponible en: http://teleformacion.uci.cu.

20. *AntLR*. [En línea] [Consultado el: 12 de ene. de 2009] Disponible en: www.antlr.org.

21. Parr, Terence. *AntLR Reference Manual Versión 2.7.1*. Universidad de San Franciso, May. 09, 2004. [En línea] [Consultado el: 12 de ene. de 2009] Disponible en: http://www.antlr.org/download.

22. García Cota, Enrique José y Troyano Jiménez, José Antonio. *Guía práctica de ANTLR 2.7.2. Versión 1.0 (Septiembre 2003).* E.T.S. de Ingeniería Informática de la Universidad de Sevilla. Departamento de Lenguajes y Sistemas Informáticos.

23. *Compilando un compilador de Java***.** [Consultado el: 20 de nov. de 2008] Disponible en: http://www.albertnogues.com.

24. Morales, Pedro. *XML y Marcado: MathML.* [En línea] [Consultado el: 20 de dic. de 2008] Disponible en: http://www.tejedoresdelweb.com.

25. Mathematical Markup Language (MathML) 1.0 Specification. [En línea] [Consultado el: 20 de dic. de 2008] Disponible en: http://www.w3.org/TR/REC-MathML.

26. MathML Test Suite 2.0. [En línea] [Consultado el: 20 de dic. de 2008] Disponible en: www.w3.org/Math/testsuite/

27. Volkmann, Mark. *ANTLR 3. Object Computing, Inc*. 2008. [En línea] [Consultado el: 30 de marzo de 2009] Disponible en: http://jnb.ociweb.com/jnb/jnbJun2008.html

28. *Abimath-plugin.* [En línea] [Consultado el: 28 de nov. de 2008] Disponible en: http://www.abisource.com/

29. *GtkMathView.* [En línea] [Consultado el: 12 de oct. de 2008] Disponible en: helm.cs.unibo.it/mml-widget/

30. GÁLVEZ ROJAS, SERGI y MORA MATA, MIGUEL ÁNGEL *.JAVA A TOPE: TRADUCTORES Y COMPILADORES CON LEX/YACC, JFLEX/CUP Y JAVACC.* Versión corregida Octubre 2005, ISBN: 84-689-1037-6. Universidad de Málaga.

Anexos

Anexo 1

Ejemplo del código para notación de presentación de MathML para la expresión: "a-db".

```
<?xml version="1.0" encoding="UTF-8"?>
<math display="block" xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi> a </mi>
    <mo> - </mo>
    <mrow>
      <mi> d </mi>
      <mo> &InvisibleTimes; </mo>
      <mi> b </mi>
    </mrow>
  </mrow>
</math>
```

Anexo 2

Ejemplo del código para notación de contenido de MathML para la expresión "a-db":

```
<?xml version="1.0" encoding="UTF-8"?>
<math display="block" xmlns="http://www.w3.org/1998/Math/MathML">
  <apply/>
    <minus/>
    <ci> a </ci>
    <apply/>
      < times />
      <ci> d </ci>
      <ci> b </ci>
    </apply>
  </apply>
</math>
```

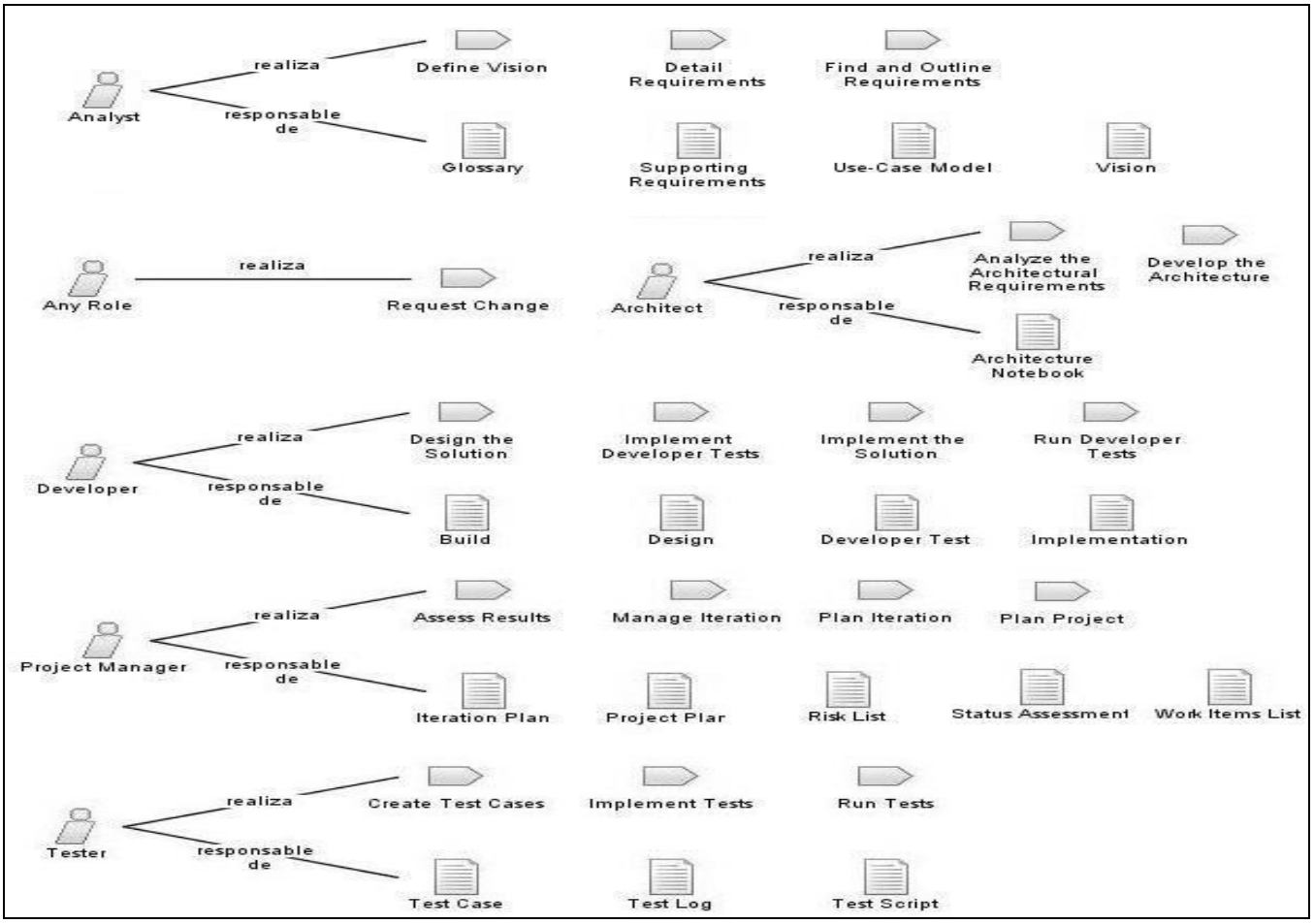
Anexo 3

Representación en formato interno del Editor de Ecuaciones de las expresiones: ¨x=2¨, ¨x+2¨, ¨x-2¨, ¨2*x¨, ¨x/2¨ y ¨x+2=a¨.

```
EQUATION(x,2)
ADD(x,2)
MINUS(x,2)
MULT(x,2)
DIV(x,2)
EQUATION(ADD(x,2), a)
```

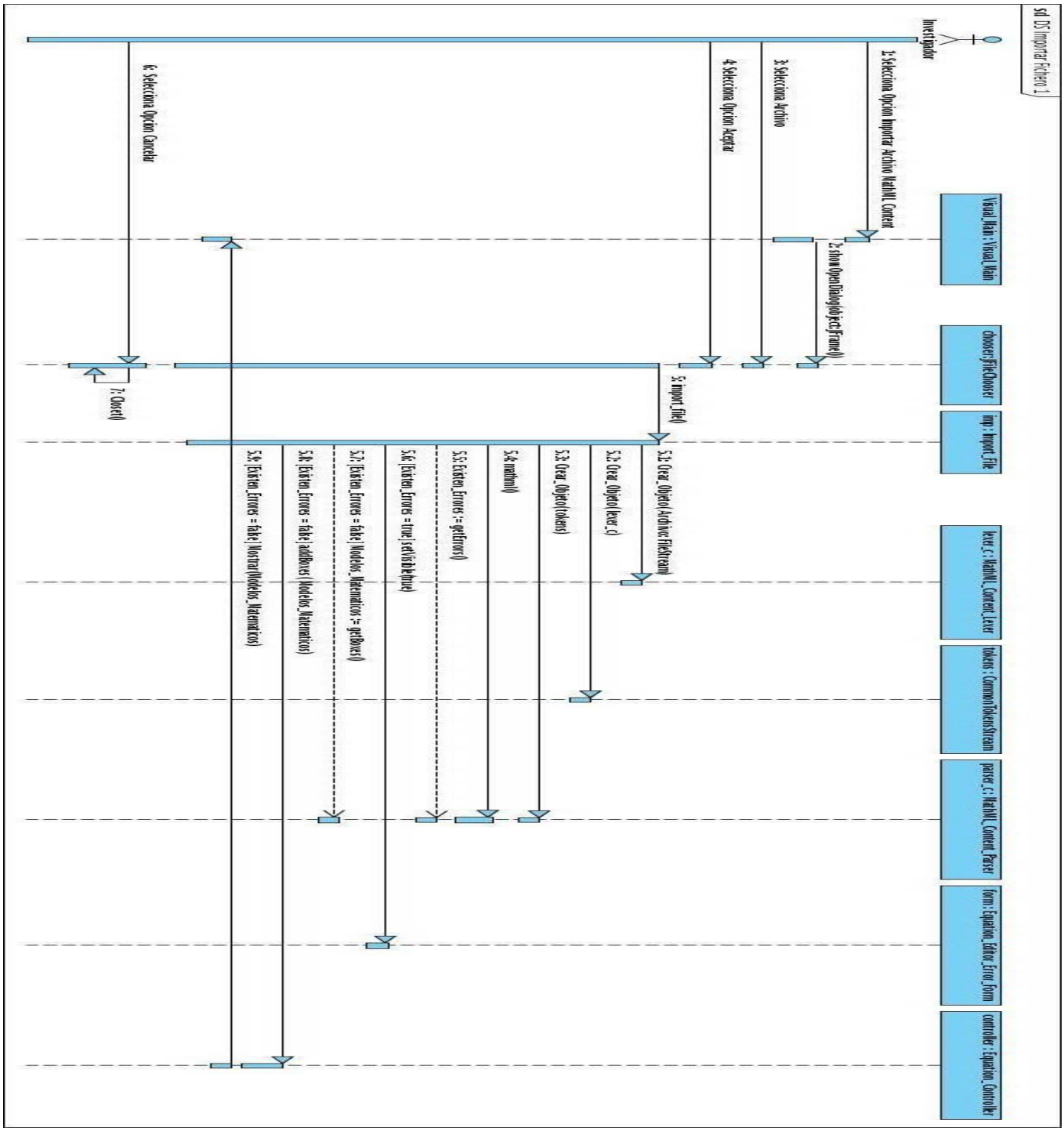
Anexo 4

Roles y Artefactos de la Metodología OpenUP.



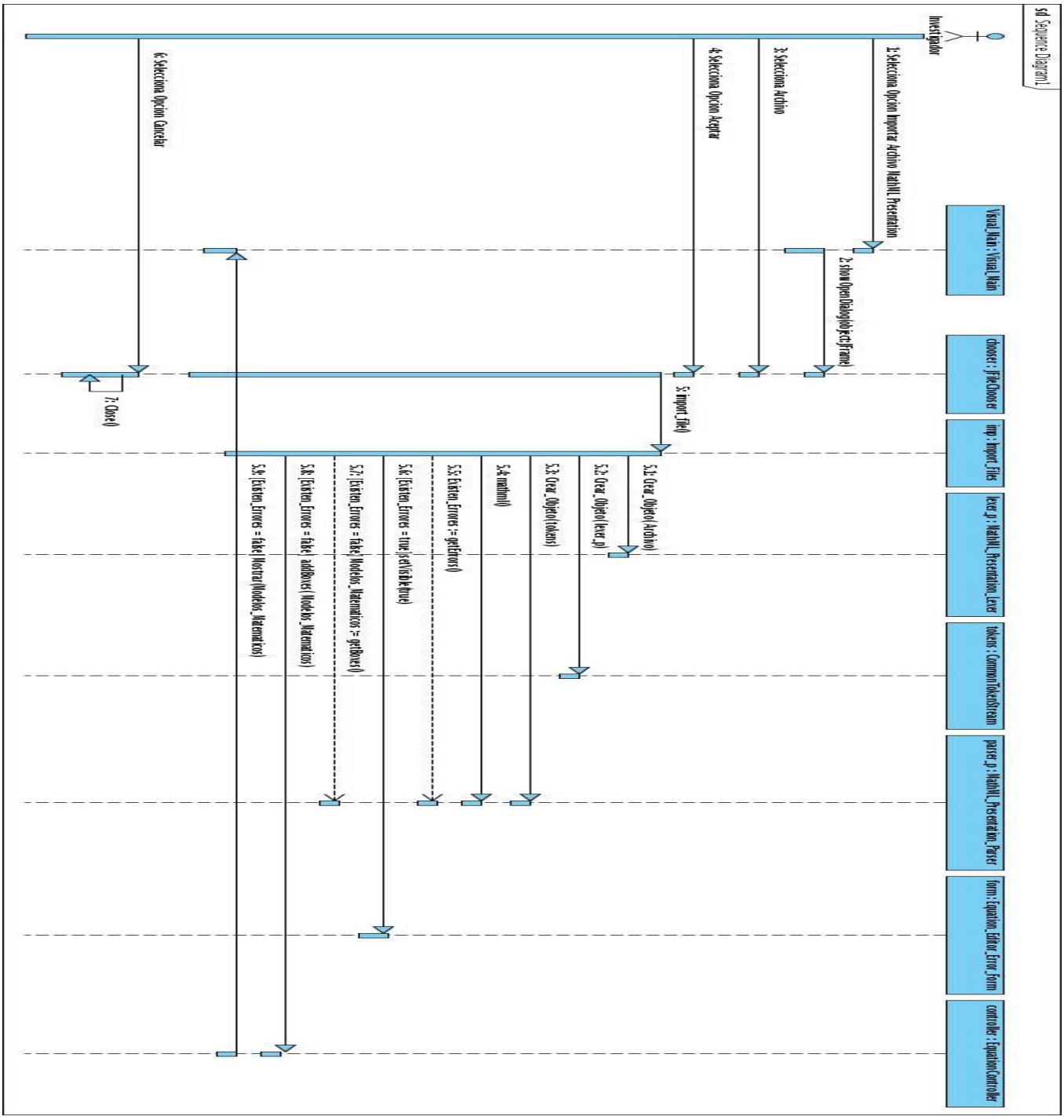
Anexo 5

Diagrama de secuencia para el flujo alterno ¨Importar Archivo en Formato MathML Content¨.



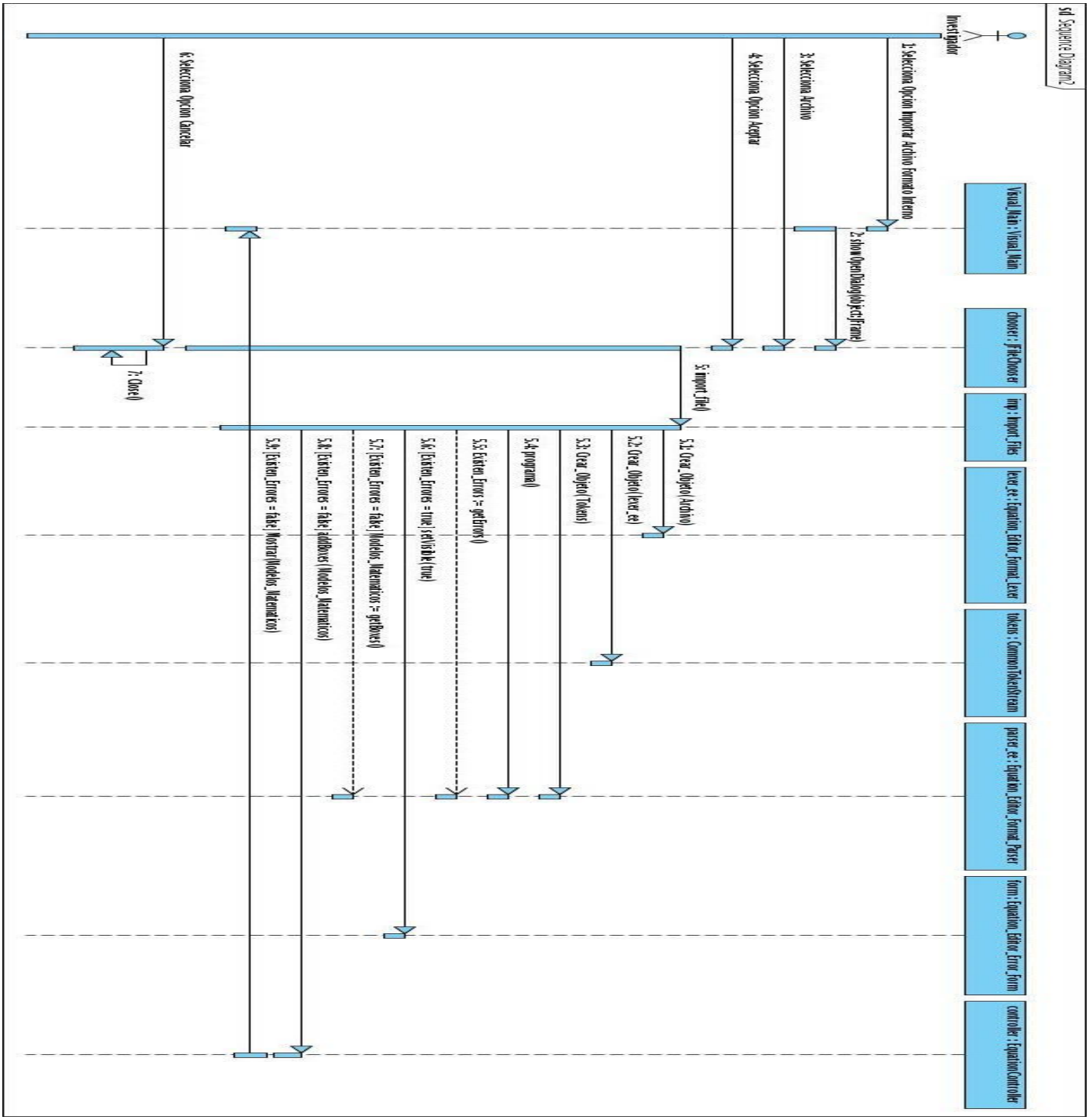
Anexo 6

Diagrama de secuencia para el flujo alterno ¨Importar Archivo en Formato MathML Presentation¨.



Anexo 7

Diagrama de secuencia para el flujo alterno ¨Importar Archivo en Formato Interno¨.



Anexo 8

MathML Content para la expresión: ¨a+b=c+d¨.

```
<?xml version="1.0" encoding="UTF-8" ?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <apply>
    <eq />
    <apply>
      <plus />
      <ci>a</ci>
      <ci>b</ci>
    </apply>
    <apply>
      <plus />
      <ci>c</ci>
      <ci>d</ci>
    </apply>
  </apply>
</math>
```

Anexo 9

MathML Content para la expresión: ¨a+b=c+d¨ con errores.

```
<?xml version="1.0" encoding="UTF-8" ?>
<math xmlns="http://www.w3.org/1998/Math/MathML"
  <apply>
    <eq />
    <apply>
      <plus />
      <ci>a
      <ci>b</ci>
    </apply>
    <apply>
      <plus
      <ci>c</ci>
      <ci>d</ci>
    </apply>
  </apply>
</math>
```

Anexo 10

MathML Presentation para la expresión: ¨a+b=c+d¨.

```
<? xml version="1.0" encoding="UTF-8"?>
<math xmlns="http://www.w3.org/1998/Math/MathML">
  <mrow>
    <mi> a </mi>
    <mo> + </mo>
    <mi> b </mi>
  </mrow>
  <mo> = </mo>
  <mrow>
    <mi> c </mi>
    <mo> + </mo>
    <mi> d </mi>
  </mrow>
</math>
```

Anexo 11

Expresión: ¨a+b=c+d¨ codificada en Formato Interno.

```
EQUATION( ADD( a , b ), ADD( c , d ) )
```

Glosario de términos

ASCII: Es un código de caracteres basado en el alfabeto latino tal como se usa en inglés moderno y otras lenguas occidentales.

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.

Código fuente: El código fuente es un texto escrito generalmente por una persona que se utiliza como base para generar otro código con un compilador o intérprete para ser ejecutado por una computadora.

DTD: Document Type Definition o definición del tipo de documento, especifica para los lenguajes de marcado el tipo de codificación a utilizar.

LaTeX: Lenguaje para procesar textos matemáticos basado en un lenguaje de marcado formado por un gran conjunto de marcos de Tex.

MatLab: es una potente herramienta de cálculo. Puede ser utilizada en computación matemática, modelado y simulación, análisis y procesado de datos, visualización y representación de gráficos y desarrollo de algoritmos.

Notación EBNF: Extended Backus-Naur Form (también conocida como notación extendida Backus-Naur) es una metasintaxis usada para expresar gramáticas libres de contexto: es decir, una manera formal de describir lenguajes formales. Una especificación de EBNF es un sistema de reglas de derivación, escrito como por ejemplo para la dirección postal de un país: **<dirección postal>:= <nombre> <dirección> <código postal>.**

Plug-in: Es una aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica, generalmente muy específica, como por ejemplo servir como driver en una aplicación, para hacer así funcionar un dispositivo en otro programa.
Plataforma: Combinación de hardware y software usada para ejecutar aplicaciones.

Polimórfico: Se define un objeto como polimórfico a la entidad que puede contener valores de diferentes tipos durante la ejecución del programa.

SED: Sistema de Ecuaciones Diferenciales.

Software: Término genérico que designa al conjunto de programas que posibilitan realizar una tarea específica en un ordenador.

Sistemas Biológicos: Sistemas abiertos que operan en condiciones alejadas del equilibrio termodinámico, con muchas y fuertes interacciones no lineales entre sus muchos elementos.

Simulación: Presentación de algo como real. Es un intento de modelar situaciones de la vida real por medio de un programa de computadora.

Stakeholders: Personas u organizaciones que están activamente implicadas en el negocio, ya sea porque participan en él o porque sus intereses se ven afectados con los resultados del proyecto.

TeX: Mezcla entre procesador de textos y lenguaje de programación usado fundamentalmente para escribir documentos de contenido científico y gran calidad de impresión.

W3C: World Wide Web Consortium. Desarrolla tecnologías ínter-operables (especificaciones, guías, software y herramientas) para maximizar el potencial de la web.