Universidad de las Ciencias Informáticas

FACULTAD 6

Título: “Biblioteca de clases para la resolución de Ecuaciones Diferenciales en Derivadas Parciales”

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

Autores: Yarisleydis González Rama

Nelson García Jorge

Tutores: MsC. Lázaro Maykel Sixto Camacho

Ing. Vilmavis La Rosa Sordo

Ing. Yunet González Mulet

Co-tutor: Ing. Edel Moreno Lemus

Consultante: DrC. Ángela Mireya León Mecías

La Habana. Junio, 2013.

“Año 55 de la Revolución”

"Su tiempo es limitado, así que no lo desperdicien viviendo la vida de otra persona. No se dejen atrapar por el dogma que es vivir según los resultados del pensamiento de otras personas. No dejes que el ruido de las opiniones de los demás ahogue vuestra propia voz interior y lo más importante, tengan el coraje de seguir su corazón e intuición."

Steve Jobs

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 _____.

Yarisleydis González Rama

_____

Firma del Autor

Nelson García Jorge

_____

Firma del Autor

MsC. Lázaro Maykel Sixto Camacho

_____

Firma del Tutor

Ing. Vilmavis La Rosa Sordo

_____

Firma del Tutor

Ing. Yunet González Mulet

_____

Firma del Tutor

DATOS DE CONTACTO

MSc. Lázaro Maykel Sixto Camacho
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: **sixto@uci.cu**

Ing. Vilmavis La Rosa Sordo
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: **vlarosa@uci.cu**

Ing. Yunet González Mulet
Universidad de las Ciencias Informáticas, Habana, Cuba.
Email: **ygonzalezmu@uci.cu**

AGRADECIMIENTOS

A mi mamá Marisleydis por haberme traído al mundo, por todo su cariño y comprensión, por siempre estar y por todas las noches y los días malos que pasó conmigo.

A mi papá Arturo por su apoyo y cariño incondicional, por demostrarme que lo que me proponga lo puedo lograr.

A mi hermano Noel por demostrarme que el que persevera triunfa y que puedo ser una mejor persona.

A mi abuela Mey por su ternura, cariño y tantas enseñanzas que me ha dado y a mi familia por siempre cuidarme y quererme.

A mi novio Nelson por soportarme en los momentos malos y por siempre darme su amor y cariño.

A mis amigas y amigos: Rosa, Liena, Ailin Guerra, Aylin Muiño, Yaima, Irina, Arianna, Yeiser, Dimel, Yan, El gordo, Adrian, Reisel, Tomas, Alexei, Fernando por haberme dado fuerzas y ánimos en los momentos más difíciles de la carrera.

A los profes Vilmavis, Sixto, Niurka, Alberto, Yunet, Yudisney y Adonis por su ayuda.

Quisiera dar especiales agradecimientos:

A mis padres.

A mi hermano.

A toda mi familia.

A Yari.

A Sixto.

A Vilmavis.

A la profesora Ángela Mireya León de la Universidad de la Habana.

A todos mis amigos.

A la revolución en especial a su líder Fidel Castro Ruz.

En general les agradezco a todos los que de una manera u otra me han brindado su apoyo incondicional para hacer posible este momento. Gracias a todos.

DEDICATORIA

Le dedico este trabajo de diploma

A mi mamá Marisleydis por ser mi ejemplo de mujer a seguir.

A mi papá Arturo por ser mi ejemplo a seguir.

A mi hermano Noel Arturo por ser el mejor hermano del mundo.

A mi abuela Mey por enseñarme que todo lo que uno se proponga es posible de alcanzar.

A mi abuela y Juvencia que aunque ya no esté me enseño lo lindo que es tener el cariño de una abuela.

A mi familia por brindarme las cosas más lindas de este mundo, una familia unida y mucho amor.

A mi novio Nelson porque sin su apoyo y comprensión no hubiera llegado hasta aquí.

Le quiero dedicar este trabajo de diploma principalmente a mis padres, que tan duramente se han esforzado para que este día llegara, estoy seguro que sin su apoyo y cariño no me hubiera sido posible alcanzar esta meta.

A mi hermano.

A mis abuelas, primos, tíos, a toda mi familia que han sido mi fuente de confianza, su amor ha sido mi mayor bendición.

A mi compañera de vida y de tesis Yarisleydis por creer en mí y por apoyarme en momentos críticos.

RESUMEN

Actualmente en el área de la Biología de Sistemas, existen problemas cuando se trabaja con entidades biológicas, por la demanda de conocimiento, recursos y tiempo que trae consigo. La simulación del comportamiento de sistemas biológicos apresura la obtención de resultados, haciéndolos más confiables y permitiendo ahorrar valiosos recursos. Existen herramientas informáticas que realizan las simulaciones de sistemas biológicos utilizando modelos matemáticos, sin embargo algunos de estos software no pueden simular todos los tipos de modelos, por ejemplo la plataforma de análisis y simulación de sistemas biológicos BioSyS, desarrollado por la Universidad de las Ciencias Informáticas no es capaz de simular sistemas biológicos que se modelen con ecuaciones en derivadas parciales. El trabajo que a continuación se presenta tuvo como objetivo la creación de una biblioteca de clases, capaz de resolver ecuaciones diferenciales en derivadas parciales, empleando el Método de Elementos Finitos, utilizando el asistente matemático MatLab o el lenguaje de programación Java. Puede ser utilizada por otros sistemas o software que se encarguen de simular problemas similares o de distintas naturalezas que se modelen con este tipo de objeto matemático.

PALABRAS CLAVE

Biblioteca se clases, Método de Elementos Finitos, Sistemas biológicos, Sistemas de ecuaciones diferenciales en derivadas parciales.

Índice

Índice de Figuras

Índice de Tablas

Introducción

Los estudios de la Biotecnología han alcanzado avances significativos en la última década, varias son las ramas que esta ciencia moderna emplea para su fundamento, siendo la más joven: La Biología de Sistemas (BS), dicha ciencia es un campo de investigación multidisciplinario (1), que vincula ciencias como la Matemática, la Física, la Biología y la Informática. Tiene como objeto de estudio los organismos o sistemas biológicos, que son tratados como un todo, centrándose en las relaciones que existen entre sus componentes. La BS representa los sistemas biológicos de forma gráfica y/o como modelos matemáticos utilizando Sistemas de Ecuaciones Diferenciales Ordinarias (EDO), Sistemas de Ecuaciones Diferenciales en Derivadas Parciales (EDP), Autómatas Celulares y Ecuaciones en Diferencias, que son de conocimiento por parte de los científicos debido a investigaciones realizadas con anterioridad. De estos modelos matemáticos pueden obtenerse soluciones, que se alcanzan cuando se realizan simulaciones al sistema en estudio para predecir algunos procesos biológicos. Una simulación permite evaluar el comportamiento de un sistema bajo condiciones determinadas en un lapso de tiempo dado.

La BS maneja gran cantidad de datos debido a los numerosos estudios experimentales que pueden ser realizados. La manipulación de mucha información puede ser engorrosa, pues su procesamiento se realiza de forma manual, por lo que el uso de herramientas informáticas que posibiliten la manipulación de este tipo de información es de gran importancia.

En la actualidad existen varias herramientas informáticas para la simulación de sistemas biológicos, como por ejemplo: Cellware, Biogrid, ImmunoGrid y entre otros BioSyS; este último desarrollado por el departamento de Bioinformática perteneciente al Centro de Tecnología y Gestión de Datos (DATEC) de la Universidad de Ciencias Informáticas (UCI) en conjunto con el Centro de Inmunología Molecular (CIM) para la simulación y análisis de sistemas biológicos. Estos programas posibilitan realizar simulaciones a modelos biológicos de forma rápida y eficaz, además permiten mostrar el comportamiento de determinados elementos, de los modelos a los cuales se les realizan las simulaciones, de esta forma los investigadores ahorran tiempo realizando los experimentos in-vitro. Las herramientas son capaces de realizar las simulaciones de los modelos biológicos que se describen a partir de alguno de los modelos matemáticos antes descritos, devolviendo una serie de resultados para que sean interpretados por los biólogos de sistemas, estas interpretaciones son útiles para la comunidad científica. Sin embargo algunas de estas aplicaciones entre las que se encuentra BioSyS no pueden realizar simulaciones de fenómenos que no solo se representen mediante ecuaciones diferenciales ordinarias, ejemplo de esto son los

sistemas de ecuaciones diferenciales en derivadas parciales que puede representar a varios fenómenos entre los que se encuentra el conocido como Quimiotaxis, que está presente en procesos biológicos de distinta naturaleza. Entre los más relevantes está la Angiogénesis que es de los procesos clave en el desarrollo del cáncer. La Quimiotaxis también está presente en otros procesos, como el movimiento de bacterias o movimientos celulares en el Sistema Inmune. La simulación de este proceso de manera in-sílico[1] aportaría valiosos conocimientos para el estudio del mismo y en consecuencia apoyaría las investigaciones que en el ámbito de los estudios del cáncer se puedan realizar.

De lo antes descrito se plantea como **problema a resolver** ¿Cómo estandarizar la simulación de los sistemas biológicos que se representan a partir de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales?

Se define como **objeto de estudio** Simulación de Sistemas Biológicos.

Para dar solución al problema planteado se define como **objetivo general** Desarrollar una biblioteca de clases que permita estandarizar la simulación de sistemas biológicos que son descritos mediante Sistemas de Ecuaciones Diferenciales en Derivadas Parciales, desglosado en los siguientes **objetivos específicos:**

1. Seleccionar métodos numéricos para la resolución de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
2. Diseñar las funcionalidades de la biblioteca de clases.
3. Implementar las funcionalidades de la biblioteca de clases.
4. Verificar el correcto funcionamiento de la biblioteca de clases implementada.

Se define como **campo de acción** Simulación de Sistemas Biológicos a partir de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.

Para resolver el problema planteado y lograr el cumplimiento de los objetivos, se establecieron las siguientes **tareas a cumplir:**

1. Análisis de los Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
2. Análisis de métodos numéricos para dar solución a los Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.

[1] **In-sílico:** Hecho por computadora o vía simulación computacional. Fue usada por primera vez en público en 1989 en el taller *Autómata Celular: Teoría y Aplicaciones*, en Los Álamos, Nuevo México por Pedro Miramontes, un matemático de la Universidad Nacional Autónoma de México.

3. Selección de los métodos numéricos para dar solución a los Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
4. Definición de las funcionalidades de la biblioteca de clases.
5. Realización del diseño de la biblioteca de clases.
6. Implementación de las funcionalidades de la biblioteca de clases para realizar la simulación utilizando el asistente matemático MatLab.
7. Implementación de las funcionalidades de la biblioteca de clases para realizar la simulación utilizando el lenguaje de programación Java.
8. Selección de los métodos y técnicas para probar las funcionalidades implementadas.
9. Realización de pruebas de caja blanca y caja negra para validar el correcto funcionamiento de la biblioteca de clases.

Para el desarrollo de la biblioteca de clases se utilizaron varios métodos investigación, estos métodos son los Métodos empíricos y los Métodos particulares.

Los **Métodos Empíricos** describen y explican las características fenomenológicas del objeto, representan un nivel de la investigación cuyo contenido procede de la experiencia y es sometido a cierta elaboración racional. Los métodos empíricos generales son: la observación, la medición y la experimentación. (2) Para el desarrollo de la presente investigación se seleccionó el método Observación, ya que es la percepción planificada dirigida a un fin y relativamente prolongada de un hecho o fenómeno. Es el instrumento universal del científico, se realiza de forma consciente y orientada a un objetivo determinado.

Los **Métodos Particulares** son más específicos y están desarrollados en base a las características propias de cada ciencia y para su aplicación están vinculados a técnicas de recolección de datos característicos de ese tipo de investigación. Los métodos particulares son la entrevista y la encuesta. (2) En este caso se utilizó la Entrevista para obtener información acerca de los métodos de resolución de sistemas de ecuaciones diferenciales en derivadas parciales, saber qué dificultades estaban enfrentando los científicos del CIM y para obtener los requisitos funcionales y no funcionales que debe tener el software a desarrollar.

La estructura del documento se resume en los siguientes acápites:

Capítulo 1: Fundamentos Teóricos: En este capítulo se da a conocer el marco teórico estudiado y seleccionado para desarrollar el presente trabajo, donde se encuentran los métodos matemáticos

estudiados y las tecnologías escogidas, todo esto orientado al desarrollo de un software en el campo de la Biología de Sistemas.

Capítulo 2: Descripción y diseño de la biblioteca de clases: En este capítulo se definen los requisitos funcionales de la biblioteca de clases y los requisitos mínimos que debe tener el sistema de cómputo donde se vaya a utilizar. Se lleva a cabo el diseño de la biblioteca donde se realiza el diagrama de clases del diseño con su respectiva descripción de clases. Se describe la representación arquitectónica de la biblioteca, haciendo énfasis en el estilo de arquitectura utilizado.

Capítulo 3: Implementación y Pruebas de Software: En este capítulo se muestran fragmentos del código fuente de la biblioteca. Además de los casos de prueba de caja blanca y caja negra, con el objetivo de mostrar la verificación que se realizó.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

Introducción

En este capítulo se abordan aspectos relacionados con la BS, la simulación de sistemas biológicos, los métodos matemáticos para la resolución de sistemas de ecuaciones diferenciales en derivadas parciales, además de la metodología y diferentes tecnologías y lenguajes que fueron utilizados para la construcción de la biblioteca de clases.

1.1. Biología de sistemas

La Biología de Sistemas se define como: “la parte de la ciencia que se encarga del estudio de los mecanismos que describen procesos biológicos complejos en un organismo, representándolos como sistemas integrados de múltiples componentes interactuando entre sí”. (3)

A menudo se suelen confundir los términos Biología de Sistemas y Sistemas Biológicos, por lo cual se hace necesario distinguir que los Sistemas Biológicos son sistemas abiertos que operan en condiciones alejadas del equilibrio termodinámico, con muchas y fuertes interacciones no lineales entre sus muchos elementos. (4)

Con el uso de la BS los científicos son capaces de hacer predicciones, a partir de modelos matemáticos para conocer cómo se comporta un ser vivo o un conjunto de células ante una situación dada, lo que permite crear fármacos contra innumerables enfermedades.

1.2. Modelo matemático. Modelación de sistemas biológicos

La biología de sistemas utiliza los modelos matemáticos para la representación de sistemas biológicos. Estos modelos son una de las herramientas que se utilizan para el estudio de problemas relacionados con la medicina como: la Biología, la Fisiología, la Bioquímica y la Farmacocinética; sus objetivos primordiales son de demostración, enumeración, representación, explicación y predicción de fenómenos en dichas áreas. El modelo matemático permite representar un problema médico o biológico de una manera objetiva en que se definen una serie de relaciones matemáticas entre las mediciones cuantitativas (del problema) y sus propiedades. (5)

Se puede hacer uso de distintos tipos de modelos para describir sistemas biológicos. Estos modelos matemáticos son representados mediante los siguientes objetos matemáticos:

- Sistemas de Ecuaciones Diferenciales Ordinarias.

En estas ecuaciones intervienen una variable dependiente y sus derivadas con respecto a una o más variables independientes, es decir: z = f(x) (6)

- Ecuaciones en diferencias o mapas (ecuaciones recursivas).

Es una expresión que relaciona distintas sucesiones, siendo una de ellas una sucesión desconocida. Son similares a las ecuaciones diferenciales, sustituyendo las funciones por sucesiones. Los sistemas de ecuaciones en diferencias intentan simular un fenómeno en un modo discreto (esto es como verlo a intervalos iguales de tiempo). Esto es coherente con la realidad, ya que normalmente se toman una serie de medidas espaciadas en el tiempo, una vez al día, o por semana, o al mes por ejemplo (siempre a la misma hora si se confía que sirva para detectar un patrón). (7)

- Autómatas Celulares.

Son redes de autómatas simples conectados localmente. Cada autómata simple produce una salida a partir de varias entradas, modificando en el proceso su estado según una función de transición. Por lo general, en un autómata celular, el estado de una célula en una generación determinada depende única y exclusivamente de los estados de las células vecinas y de su propio estado en la generación anterior.

Los autómatas celulares son herramientas útiles para modelar cualquier sistema en el universo. Pueden considerarse como una buena alternativa a las ecuaciones diferenciales y han sido utilizados para modelar sistemas físicos, como interacciones entre partículas, formación de galaxias, cinética de sistemas moleculares y crecimiento de cristales, así como diversos sistemas biológicos a nivel celular, multicelular y poblacional. (8) (9)

- Ecuaciones diferenciales en derivadas parciales.

Son ecuaciones donde las funciones y sus derivadas dependen de más de una variable independiente, es decir: z = f(x, y)

Son expresados en términos de una EDP los problemas de mecánica de los fluidos, transmisión de calor, cálculo de tensiones, deformaciones en sólidos, fenómenos biológicos como la Quimiotaxis. Del conjunto de todas las posibles EDP, destacan las de segundo orden, por modelar un grupo importante de fenómenos. Su forma general es (en el caso bidimensional):

$$\frac{\partial}{\partial x}\left(A\frac{\partial \phi}{\partial x}\right)+\frac{\partial}{\partial y}\left(C\frac{\partial \phi}{\partial y}\right)+\frac{\partial}{\partial x}\left(B\frac{\partial \phi}{\partial y}\right)+\frac{\partial}{\partial y}\left(D\frac{\partial \phi}{\partial x}\right)+E\frac{\partial \phi}{\partial x}+F\frac{\partial \phi}{\partial y}+G\phi+Fu(x,y)=H\frac{\partial \phi}{\partial t}+\frac{\partial}{\partial t}\left(H1\frac{\partial \phi}{\partial t}\right)$$

En caso de que los coeficientes A, B, C, D, E, F, G, Fu, H y H1 sean constantes su forma general sería:

$$A\frac{\partial^2\phi}{\partial x^2}+B\frac{\partial^2\phi}{\partial x\,\partial y}+C\frac{\partial^2\phi}{\partial y^2}+D\frac{\partial\phi}{\partial x}+E\frac{\partial\phi}{\partial y}+F\phi=S$$

Complementariamente, las EDPs se califican de acuerdo a relaciones entre sus coeficientes,
Si

$B^2-4AC<0\rightarrow \text{elíptica}$

$B^2-4AC=0\rightarrow \text{parabólica}$

$B^2-4AC>0\rightarrow \text{hiperbólica}$

Las EDPs elípticas aparecen en problemas estacionarios de transmisión de calor, difusión de partículas o vibración de una membrana. Las ecuaciones parabólicas típicamente resultan de analizar la evolución temporal de los problemas mencionados para ecuaciones elípticas. Las ecuaciones de tipo hiperbólico son comunes en el análisis de transporte de masa en fluidos, fenómenos ondulatorios y otros procesos. (10)

Para el desarrollo de la biblioteca de clases se decidió utilizar la modelación mediante EDP que permite representar varios problemas y fenómenos que pueden ser de mucho interés para los científicos que desarrollan investigaciones en esta rama de la Biotecnología.

1.3. Métodos de resolución de sistemas de ecuaciones diferenciales en derivadas parciales

Para la solución de las EDPs pueden utilizarse varias técnicas, entre las que se encuentran los métodos numéricos, los cuales son un procedimiento mediante el cual se obtiene, casi siempre de manera aproximada, la solución de ciertos problemas realizando cálculos puramente analíticos y lógicos (operaciones aritméticas elementales, cálculo de funciones, consulta de una tabla de valores, cálculo preposicional, entre otras operaciones). Un procedimiento consiste en una lista finita de instrucciones precisas que especifican una secuencia de operaciones algebraicas y lógicas (algoritmo), que producen o bien una aproximación de la solución del problema (solución numérica) o bien un mensaje. De estos métodos los que más destacan son los Métodos de Homogeneización Asintótica, los Métodos de Diferencias Finitas y los Métodos de Elementos Finitos. (11)

1.3.1 Método de homogeneización asintótica (MHA)

Se basa en un desarrollo asintótico a doble escala, es una rigurosa técnica matemática para el cálculo de los coeficientes efectivos de medios heterogéneos y periódicos. Desde un punto de vista matemático, este

método garantiza que la solución del problema original con una micro-estructura periódica converge a la solución del problema homogeneizado cuando el período de la micro-estructura tiende hacia cero. De esta forma, el estudio del problema inicial sobre un medio heterogéneo se traslada al de un problema equivalente, sobre un medio homogéneo, que requiere de la solución de los llamados problemas locales sobre la celda periódica. Estos problemas locales suelen estar dados por sistemas de ecuaciones diferenciales, ordinarias o parciales, con condiciones de conjugación en las intercaras, y condiciones de periodicidad en las caras opuestas de la celda periódica. Por tanto, desde un punto de vista práctico, esta teoría asintótica no proporciona directamente algoritmos numéricos para dar respuestas a problemas ingenieriles. (12)

1.3.2 Método de Diferencias Finitas (MDF)

El método consiste en una aproximación de las derivadas parciales por expresiones algebraicas con los valores de la variable dependiente en un limitado número de puntos seleccionados. Como resultado de la aproximación, la ecuación diferencial parcial que describe el problema es reemplazada por un número finito de ecuaciones algebraicas, en términos de los valores de la variable dependiente en puntos seleccionados. El valor de los puntos seleccionados se convierte en las incógnitas. El sistema de ecuaciones algebraicas debe ser resuelto y puede llevar un número largo de operaciones aritméticas. (13)
El MDF es usado por otros métodos como el Método de Elementos Finitos para solucionar problemas con trascendentes temporales.

1.3.3 Método de Elementos Finitos (MEF)

Es un método numérico para resolver ecuaciones diferenciales por medio de aproximaciones discretas. A diferencia del MDF, en el cual la zona de solución es un conjunto de puntos discretos, el método de elementos finitos supone que la zona de solución está compuesta de muchas subzonas interconectadas, las que se denominan "elementos finitos". Estos elementos, pueden tomar formas simples (por ejemplo, líneas, triángulos, rectángulos, paralelepípedos y/o polígonos incluso circulares) se ensamblan de diferentes maneras para representar la solución sobre una región cualquiera. (14)
El MEF permite realizar un modelo matemático de cálculo del sistema real, más fácil y económico de modificar que un prototipo. Sin embargo no deja de ser un método aproximado de cálculo debido a las hipótesis básicas del método. Los prototipos, por lo tanto, siguen siendo necesarios, pero en menor número, ya que el primero puede acercarse bastante más al diseño óptimo.

El MEF se ha convertido en una tecnología indispensable en la modelación y simulación de sistemas avanzados y complejos, en diversos campos de la ingeniería y la ciencia, como el transporte, las comunicaciones, las construcciones, la aeronáutica, la medicina, entre otros. Las potencialidades de dicho método son increíbles e incalculables; es posible diseñar en la computadora, las funcionalidades y características de piezas que serán usadas en sistemas ingenieriles avanzados y de comprobar cómo se comportarán en su uso real posterior.

Luego de haber realizado un análisis de los métodos de resolución de EDP se llegó a la conclusión que el MHA es rigurosamente establecido desde el punto de vista matemático. Sin embargo, su implementación numérica para compuestos tridimensionales se ha visto limitada por la complejidad de la implementación computacional; el MDF trae consigo un alto número de operaciones aritméticas, por lo que la realización de los cálculos con él, demandaría más tiempo, decidiendo entonces utilizar el MEF que es clave en la modelación y simulación de sistemas avanzados y complejos, no requiere gran número de operaciones aritméticas y no necesita utilizar muchos recursos de cómputo. Para conocer mejor el MEF y tener una idea clara de los elementos que de este se utilizaron para el desarrollo de la biblioteca, consultar el documento Método de Elementos Finitos en el Expediente de Proyecto de la Biblioteca para la simulación de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.

1.4. Simulación de Sistemas biológicos.

Una vez seleccionado el método numérico con el que se va a resolver los modelos matemáticos, se puede realizar las simulaciones de los sistemas biológicos. La simulación de sistemas biológicos “es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso, y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar estrategias con las cuales se puede operar sobre él”. (15)

La simulación es ampliamente utilizada por los científicos que estudian sistemas complejos. Esta técnica incluye, tanto la construcción del modelo como su uso analítico. Ofrece la posibilidad de comprimir el tiempo, esfuerzo y cantidad de recursos necesarios para la toma de decisiones.

La simulación de sistemas biológicos es utilizada para simular cómo funciona un sistema biológico de manera integral. “Permite a las compañías farmacéuticas reducir de manera significativa el número de experimentos de laboratorio requeridos para identificar posibles objetivos de los fármacos. El modelado “in- sílico” (en oposición al “in vitro” [en vidrio] o “in vivo” [en vivo]) también permite que los investigadores predigan los efectos de los fármacos en el cuerpo humano, incluyendo su eficacia y seguridad.” (16)

Aunque las simulaciones pueden ser utilizadas para analizar y sintetizar complejas y extensas situaciones no siempre generan soluciones óptimas dado que siempre quedarán variables fuera que pueden ser vitales para el funcionamiento del sistema y dar un resultado que la simulación no contempló.

1.4.1 Desventajas de las simulaciones biológicas

Un buen modelo de simulación puede resultar bastante costoso; a menudo el proceso de desarrollar un modelo es largo y complicado. El modelo de simulación no produce respuestas por sí mismo. Las soluciones e inferencias no son usualmente transferibles a otros problemas. Siempre quedarán variables por fuera y esas variables pueden cambiar completamente los resultados en la vida real que la simulación no previó, ya que el arte de escoger un modelo está en tomar los parámetros necesarios, no muchos porque no se va ver claro el fenómeno, ni pocos porque no se va a representar en plenitud lo que pasa. (17)

1.4.2 Ventajas de las simulaciones biológicas

La simulación biológica es un proceso relativamente eficiente y flexible, que puede ser usada para analizar y sintetizar una compleja y extensa situación real, pero no puede ser empleada para solucionar un modelo de análisis cuantitativo convencional. En algunos casos es el único método disponible. No interfiere en sistemas del mundo real, pero permite la inclusión de complicaciones reales. También permite estudiar los efectos interactivos de los componentes individuales o variable para determinar las más importantes. (17)

Las simulaciones biológicas han sido de interés para muchas compañías que a nivel mundial se dedican al estudio de las mismas. Estas compañías se han visto en la necesidad de desarrollar software que permitan agilizar los cálculos que se deben realizar para la simulación.

1.5. Aplicaciones informáticas y Biblioteca de clases.

Se pueden realizar simulaciones a sistemas biológicos tanto con aplicaciones informáticas como con bibliotecas de clases, que permiten estandarizar una solución posibilitando ser reutilizada en un nuevo contexto.

Una aplicación informática es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos. Suelen ser la solución informática para la automatización de ciertas tareas complicadas como pueden ser la contabilidad, la redacción de documentos o la gestión de un almacén. Ciertas aplicaciones desarrolladas “a medida” suelen ofrecer una

gran potencia ya que están exclusivamente diseñadas para resolver un problema específico. (18) Sin embargo las aplicaciones informaticas dificultan ser reutilizadas o adaptadas a otras aplicaciones, puesto que para ello es necesario realizar modificaciones en una o en las dos si no son compatibles para que puedan ser acopladas, de modo que se hace factible crear soluciones informáticas que puedan ser usadas por otros sistemas de manera fácil como es el caso de las biblioetcas de clases.

Las bibliotecas de clases son una colección de clases o plantillas de código escrito previamente, cualquiera de los cuales se pueden especificar y utilizar por un programador cuando desarrolla una aplicación. Las bibliotecas de clases incluyen todas las clases esenciales en un formato codificado, que no sólo simplifica la programación, sino también aumenta la calidad y reutilización del código, proporcionando implementaciones de trabajos repetitivos.

Para dar solución al problema planteado se decidió realizar una biblioteca de clases ya que puede resolver el mismo problema que una aplicación. Su mantenimiento, ampliación y modificación es más sencilla, pues solo se modifican las clases o métodos necesarios sin necesidad de afectar todo el sistema. Puede ser utilizada por cualquier aplicación informática que necesite las funcionalidades implementadas en ella, con solo añadirla a su código fuente y utilizar sus algoritmos sin necesidad de modificar los códigos que estén implementados.

1.6. Bibliotecas para la resolución de ecuaciones diferenciales en derivadas parciales

En la actualidad existen bibliotecas de clases que resuelven ecuaciones diferenciales en derivadas parciales, entre las más importantes:

PETSc (*Portable Extensible Toolkit for Scientific Computation*): destinada a la resolución de aplicaciones científicas que se modelen con este tipo de ecuaciones, es programada en C y puede ser usada desde lenguajes de programación como C, C++, FORTRAN y Python, es un proyecto de código abierto por lo que su uso, reproducción y redistribución a otros usuarios es permitida; sin embargo los desarrolladores de esta herramienta no se responsabilizan por los errores que esta pueda tener, así como no dan mantenimiento ni soporte una vez esté en uso.

OpenCurrent: de código abierto, programada en C++, usa la plataforma de computación en paralelo CUDA creada por NVIDIA para el incremento del rendimiento de la computadora usando la *Graphics Processing Unit* (GPU), puede ser usada en los sistemas operativos Linux y Windows, es necesario tener instalado cmake (plataforma de código abierto para controlar el proceso de compilación del software), además posee dependencia con NetCDF (biblioteca de clases usada por las comunidades científicas

atmosféricas y oceánicas para el almacenamiento de datos). El uso de OpenCurrent conlleva una elevada dependencia tecnológica, pues necesita el uso una tarjeta gráfica NVIDIA, tener instalado cmake que para su mantenimiento necesita un contrato de 3500.00 euros, el uso de CUDA y de la biblioteca NetCDF.
GPDE (GRASS *Partial Differential Ecuation*): usada por GRASS GIS (*Geographic Resources Analysis Support System*), software libre de *Geographical Information System* (GIS) para el análisis de recursos geográficos, esta biblioteca provee funciones para crear y resolver sistemas de ecuaciones lineales y ecuaciones en derivadas parciales a las que puede dar solución por medio del paralelismo multi-núcleo, está diseñada a partir de la arquitectura de GRASS por lo que su uso en otras aplicaciones conlleva gran cantidad de modificaciones a su código fuente para que pueda ser adaptada a otra aplicación.
LibMesh: framework de C++ para la simulación numérica de EDPs en serie y en plataformas paralelas usando el método de elementos finitos, sin embargo esta biblioteca hace uso de otras herramientas para la resolución de sistemas de ecuaciones lineales como son PETSc, que por los problemas antes planteados no es conveniente usar.
Dado los inconvenientes encontrados en las principales bibliotecas para la resolución de EDPs se decidió crear una de código abierto, en lenguaje de programación java, que permita resolver este tipo de ecuaciones sin tener dependencia de herramientas externas al sistema, que agiliza su funcionamiento, suficientemente genérica como para que sea usada por cualquier herramienta que necesite sus funcionalidades sin tener que hacer cambios en sus códigos fuente y en caso de cualquier error sus creadores sean capaces de brindarle soporte.

1.7. Herramientas y metodología usadas para el desarrollo de la biblioteca de clases

Para la realización de una aplicación siempre se deben definir los tipos de tecnologías a utilizar, así como las herramientas y la metodología que serán de mayor utilidad para su implementación. A continuación se explica en detalle cada una de las tecnologías que fueron seleccionadas para llevar a cabo la implementación y documentación de la biblioteca desarrollada.

1.7.1 Metodología de desarrollo.

Para el desarrollo de un software es necesario un conjunto de procedimientos y técnicas que indiquen paso a paso todas las actividades a realizar, además de qué personas deben participar en el desarrollo de las actividades y qué papel deben tener.

Aunque la biblioteca de clases no es una aplicación informática resulta de gran importancia detallar y documentar su proceso de desarrollo, por lo que es necesaria la creación de artefactos que describan su funcionamiento generados por una metodología.

OpenUP

Es un proceso unificado (de aplicación general), ágil (se centra en el desarrollo rápido de sistemas) y extensible, dirigido a la gestión y desarrollo de proyectos de software en su mayoría para proyectos pequeños y de bajos recursos. Es aplicable a un conjunto amplio de plataformas y aplicaciones de desarrollo. Plantea que se debe tener un software ya funcional o lo que es lo mismo un proyecto ejecutable en un lapso de tiempo relativamente corto. Propone principalmente que se debe utilizar solo los procesos que sean necesarios, en este caso se necesita de personas y profesionales que sean capaces de distinguir entre lo necesario y lo que no lo es para que el proyecto no tenga errores y con eso evitar entregar al usuario final un producto de mala calidad. Este tipo de método plantea además que no se deben utilizar demasiados artefactos y sobre todo que el proyecto debe acoplarse a las necesidades del usuario pudiendo ser este modificado, mejorado y extendido. (19)

OpenUP tiene dos ventajas importantes, la primera es que este tipo de método disminuye los riesgos y la segunda ventaja es que puede utilizarse tanto en proyectos pequeños como en proyectos grandes. Si se maneja con cuidado y con profesionalismo se puede desarrollar un software de gran calidad a pesar de que se diseñe en poco tiempo y con poca documentación. Una de las finalidades que se busca con el desarrollo del presente trabajo es tener un software funcional en un lapso de tiempo relativamente corto, disminuyendo los riesgos que eso pueda implicar. Por lo antes expuesto se ha decidido utilizar esta metodología en el proceso de desarrollo del software.

1.7.2 Herramienta Case

Las herramientas CASE (*Computer Aided Software Engineering***,** Ingeniería de Software Asistida por Computador) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas ayudan en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras. (20)

Visual Paradigm para UML 8.0

Visual Paradigm para UML 8.0 proporciona un conjunto de ayudas para el desarrollo de programas informáticos, desde la planificación, pasando por el análisis y el diseño, hasta la generación del código fuente de los programas y la documentación, ha sido concebida para soportar el ciclo de vida completo del proceso de desarrollo del software a través de la representación de todo tipo de diagramas, es una herramienta potente para visualizar y diseñar elementos de software, para ello utiliza el lenguaje UML y proporciona a los desarrolladores una plataforma que les permite diseñar un producto con calidad de una forma rápida. (21)

Proporciona soporte a varios lenguajes en generación de código e ingeniería inversa a través de plataformas Java. Poderoso generador de documentación y reportes UML PDF/HTML/MS Word. Además posee una alta interoperabilidad, para maximizar la interoperabilidad de los productos de Visual Paradigm con otras aplicaciones, Visual introdujo la importación y exportación de modelos de proyecto desde o hasta un formato XML. Los usuarios y proveedores de tecnología pueden integrar Visual Paradigm en cada uno de sus modelos para utilizarlos en sus soluciones con un mínimo esfuerzo.

1.7.3 Lenguaje de Modelado

Utilizar algún lenguaje de modelado representa un pilar fundamental en cualquier proyecto de desarrollo, por su universalidad, sencillez y potencia.

UML (*Unified Modeling Language*, Lenguaje Unificado de Modelado) 2.1

Es un lenguaje gráfico para visualizar, especificar, construir y documentar cada una de las partes que comprende el desarrollo de software. UML entrega una forma de modelar cosas conceptuales, como lo son procesos de negocio y funciones de sistema, además de cosas concretas, como lo son escribir clases en un lenguaje determinado, esquemas de base de datos y componentes de software reusables. (22) Contribuye a una rápida construcción de aplicaciones de calidad, y a un menor coste.

El UML estándar está compuesto por tres partes: bloques de construcción (tales como clases, objetos, mensajes), relaciones entre los bloques (tales como asociación, generalización) y diagramas (por ejemplo, diagrama de actividad) Se utiliza para definir un sistema, para detallar los artefactos en el sistema y para documentar y construir.

1.7.4 Lenguaje de programación

Es un idioma artificial diseñado para expresar computaciones que pueden ser llevadas a cabo por máquinas como las computadoras. Pueden usarse para crear programas que controlen el comportamiento

físico y lógico de una máquina, para expresar algoritmos con precisión, o como modo de comunicación humana. Está formado de un conjunto de símbolos y reglas sintácticas y semánticas que definen su estructura y el significado de sus elementos y expresiones. Una característica relevante de los lenguajes de programación es precisamente que más de un programador pueda usar un conjunto común de instrucciones que sean comprendidas entre ellos para realizar la construcción del programa de forma colaborativa. (23)

Java 1.6

Es el lenguaje de programación utilizado ya que es un lenguaje de alto nivel, de propósito general, concurrente, basado en clases y orientado a objetos, tiene un modelo de objetos simple y elimina herramientas de bajo nivel. La memoria es gestionada mediante un recolector de basura. Soporta las características propias del paradigma de la programación orientada a objetos: abstracción, encapsulamiento, herencia y polimorfismo. Proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando. Es robusto, realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. Es *multithreaded* (multihilos), permite muchas actividades simultáneas en un programa. Los *threads* son básicamente pequeños procesos o piezas independientes de un gran proceso. (17)

Los programas desarrollados con Java son independientes de la arquitectura de la computadora donde se ejecutan, así como que también son independientes del sistema operativo que los ejecuta ya que el código Java no se ejecuta sobre el sistema operativo sino sobre una máquina virtual (JDK). (24)

1.7.5 Entornos de Desarrollo Integrado (IDE)

Un Entorno de Desarrollo Integrado, llamado también IDE por sus siglas en inglés *Integrated Development Environment*, es un programa informático compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un solo lenguaje de programación o bien puede utilizarse para varios. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes.

NetBeans 7.1

Es “una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y

gratuito sin restricciones de uso. ” (25). La plataforma NetBeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes de software llamados módulos. Un módulo es un archivo Java que contiene clases de java escritas para interactuar con la Interfaz de Programación de Aplicaciones (IPA) o API[2] (del inglés *Application Programming Interface*) de NetBeans y un archivo especial (*manifest file*) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma NetBeans pueden ser extendidas fácilmente por otros desarrolladores de software. Además es un proyecto GNU[3] (libre), tiene un excelente diseñador de interfaces integrado, es muy rápido y fácil de usar.

Eclipse

“Eclipse es un entorno de desarrollo integrado (IDE, por sus siglas en inglés), que facilita las tareas de edición, compilación y ejecución de programas durante su fase de desarrollo. Aunque Eclipse, pretende ser un entorno versátil soportando varios lenguajes de programación, es con el lenguaje Java con el que mejor se integra y con el que ha ganado su popularidad.” (26)

Es además, una aplicación gratuita, de código abierto, multiplataforma y extensible para integrar herramientas, disponible en la red para su descarga, e incluida ya en muchas distribuciones de Linux. Proporciona el entorno de desarrollo solamente, siendo necesario además, para el caso del lenguaje Java, disponer del *Java Development Kit* (JDK, por sus siglas en inglés), para poder compilar y ejecutar las aplicaciones desarrolladas. Si bien las funciones de Eclipse son más bien de carácter general, las características del programa se pueden ampliar y mejorar mediante el uso de plug-ins. (27)

A pesar de estas ventajas, implementar un plug-in para eclipse es engorroso, pues la curva de aprendizaje es muy lenta, por lo que su desarrollo demoraría un tiempo considerable. Además, desarrollar interfaces visuales resulta un poco complicado.

Se usará NetBeans 7.1 como entorno de desarrollo por ser multiplataforma, integrarse con UML, sin restricciones de uso y soportar varios sistemas de control de versiones.

[2] **API:** Es el conjunto de funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción. Son usadas generalmente en las bibliotecas

[3] GNU: El autor permite que cualquier persona pueda ejecutar, copiar, modificar y distribuir libremente el software.

1.7.6 MatLab 7.11.0

Es un software matemático muy potente, con un entorno agradable, que incluye herramientas de cálculo científico-técnico y de visualización gráfica. Ofrece un entorno de desarrollo integrado (IDE) para llevar a cabo proyectos en donde se encuentren implicados elevados cálculos matemáticos, con un lenguaje de programación propio (lenguaje M). Está disponible para las plataformas Unix, Windows y Mac OS X. MatLab integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional. El paquete MATLAB dispone de dos herramientas adicionales que expanden sus prestaciones, a saber, *Simulink* (plataforma de simulación multidominio) y GUIDE (editor de interfaces de usuario - GUI). Se pueden ampliar las capacidades de MATLAB con las cajas de herramientas (*Toolboxes*); y las de *Simulink* con los paquetes de bloques (*blocksets*). (28)

1.8. Conclusiones

En este capítulo se realizó un estudio sobre las herramientas, métodos y metodologías a aplicar en el desarrollo de la biblioteca de clases, donde se llegó a la conclusión que, el Método de Elementos Finitos es el más adecuado para la resolución de EDPs por los beneficios que aporta. Como guía en el proceso de desarrollo del software, la metodología OpenUP. Además fueron estudiadas las herramientas y tecnologías existentes con el propósito de seleccionar las más adecuadas para la solución de la investigación, se determinó utilizar como herramienta CASE, Visual Paradigm para UML 8.0 con el lenguaje de modelado UML 2.1, el IDE de desarrollo NetBeans 7.1, empleando la máquina virtual de Java 1.6, además se utilizó el asistente matemático MatLab 7.11.0 debido a las funcionalidades que tiene implementado.

CAPÍTULO 2: DESCRIPCIÓN Y DISEÑO DE LA BIBLIOTECA DE CLASES

Introducción

En el presente capítulo se da a conocer la descripción de la biblioteca de clases, el modelo conceptual en el cual se representan las principales clases y conceptos dentro del contexto del sistema, además de definir qué debe hacer el software a través de los requisitos funcionales y no funcionales.

2.1. Modelo conceptual

Para comprender mejor el entorno en el cual se va a desarrollar el software, se creó un modelo conceptual donde se explica cuáles son y cómo se relacionan los conceptos relevantes en la descripción del problema. (29)

En el siguiente modelo conceptual se especifican los conceptos fundamentales para un mejor entendimiento de la biblioteca de clases que se desarrolló, en este modelo conceptual un Modelo está compuesto por Parámetros, una o un conjunto de Ecuaciones Diferenciales y Variables, a un Modelo se le puede realizar una o muchas Simulación. La Simulación se le realiza a los Sistemas de Ecuaciones Diferenciales Ordinarias (Simulación de SEDO) mediante el uso del paquete de software ODEToJava, el asistente matemático Octave, la herramienta de cálculo numérico y simbólico Mathematica, la librería ODESolver y el asistente matemático MatLab, además se quiere que la Simulación se le realice a los Sistemas de Ecuaciones Diferenciales en Derivadas Parciales (Simulación de SEDDP) mediante el asistente matemático MatLab y el lenguaje de programación Java y dichas simulaciones van a tener un Resultado Simulación .

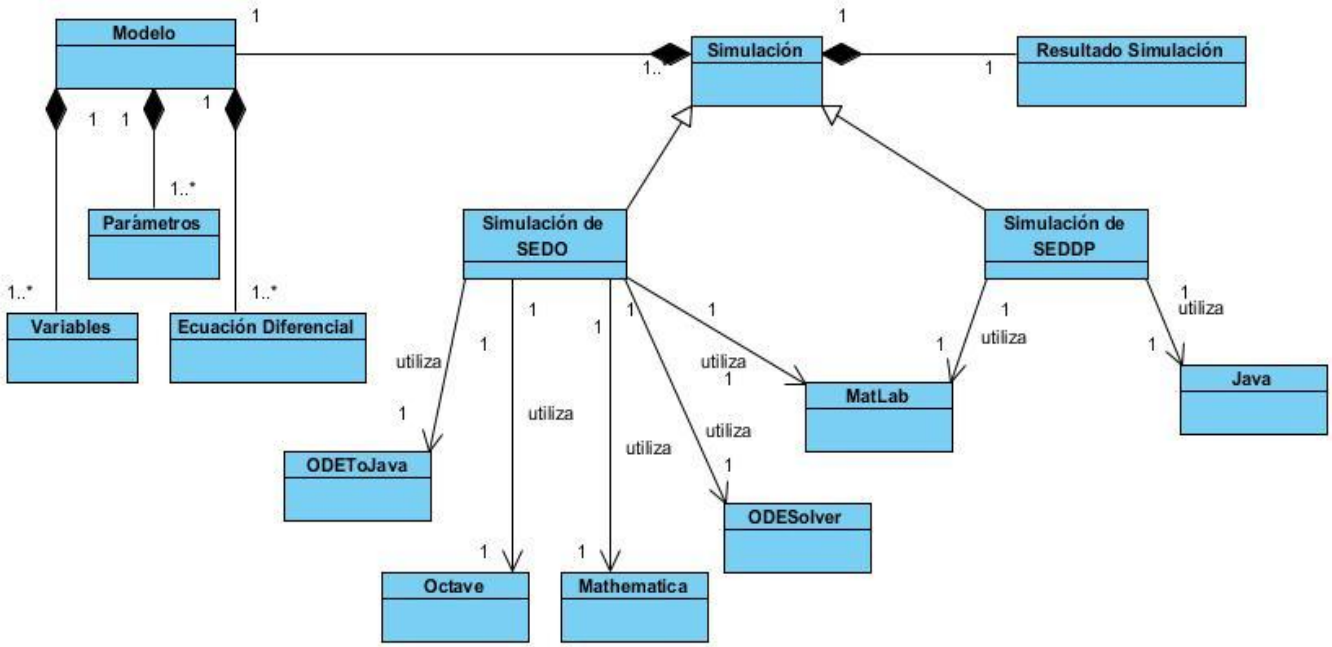

Figura 1: Modelo Conceptual

Descripción de las clases del modelo conceptual

- Parámetros: Concepto que abarca los parámetros necesarios para resolver los SED.
- Ecuación Diferencial: Concepto que representa a los Sistemas de Ecuaciones Diferenciales (SED).
- Variables**:** Concepto que representa las variables necesarias para dar solución al SED.
- Modelo: Es el concepto que representa la descripción matemática que define el problema.
- Simulación: Concepto que se asocia al proceso resultante de dado un modelo matemático, y a través de una herramienta matemática, resolver los SED Ordinarias y SED en Derivadas Parciales con de los cuales se obtienen resultados.
- Simulación de SEDO: Este concepto representa a uno de los tipos de simulación, la simulación de Sistemas de Ecuaciones Diferenciales Ordinarias.
- Simulación de SEDDP: Este concepto representa a uno de los tipos de simulación, la simulación de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
- ODEtoJava: Se asocia a al paquete de software para la resolución de los SED Ordinarias.
- Octave: Es el concepto que representa a el asistente matemático Octave para la resolución de EDO.

- Mathematica: Concepto que representa a la herramienta de cálculo numérico y simbólico Mathematica para la resolución de EDO.
- ODESolver: Representa la librería ODESolver para la resolución de EDO.
- MatLab: Se asocia con el asistente matemático MatLab para la resolución de EDO y EDP.
- Java: Clase encargada de realizar simulaciones con el lenguaje de programación Java para la resolución de EDP.
- Resultado Simulación: Son los resultados que se obtienen al realizar simulaciones a un modelo matemático.

2.2. Breve descripción del sistema

Mediante la realización del presente trabajo de diploma se desarrollará una biblioteca de clases que constituirá una herramienta computacional, capaz de simular los modelos matemáticos que se describan a través de EDPs, utilizando el asistente matemático MatLab o el lenguaje de programación Java. Esta biblioteca permitirá ser adoptada por sistemas informáticos que se dediquen a la simulación de fenómenos que se representen con este tipo de ecuaciones.

2.3. Especificación de los requisitos del sistema (ERS)

Es una descripción completa del comportamiento del software que se va a desarrollar. Incluye un conjunto de requisitos funcionales, los cuales son las funcionalidades que se necesita que efectúe la biblioteca de clases y requisitos no funcionales (o complementarios), los que imponen restricciones en el diseño o la implementación.

2.3.1 Requisitos Funcionales (RF)

Son capacidades o condiciones que el sistema debe cumplir, describen su funcionalidad, los servicios que de él se esperan, o los que proveerá, entre ellos: sus entradas, salidas y excepciones. A continuación se muestran los requisitos funcionales identificados para la biblioteca de clases a implementar:

RF1 Cargar parámetros de la simulación.

RF2 Realizar simulación utilizando el lenguaje de programación Java.

RF3 Realizar simulación utilizando el asistente matemático MatLab.

RF4 Obtener valor para un punto determinado.

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

El **RF1 Cargar parámetros de la simulación** permite cargar los parámetros de un fichero especificado por el usuario donde se encontrarán los datos necesarios para llevar a cabo la simulación y verifica que los parámetros estén correctos. El fichero especificado tiene que contar con el siguiente formato:

```
(a) * u,xx + (b) * u,yy + (c) * u,xy + (d) * u,yx + (e) * u,x + (f) * u,y + (g) * u + (fu) = (h) * u,t + (h1) * u,tt
a, b, c, d , e, f, g, fu, h , h1 son coeficientes de la ecuación, pueden ser constantes o funciones dependientes de X y de Y
x=[xIni,xFin] Intervalo de X
y=[yIni,yFin] Intervalo de Y
nel=nel Numero de elementos
nnel=nnel Numero de nodos por elementos
nnode=nnode Numero total de nodos
deltt=deltt Paso del tiempo
stime=stime Tiempo inicial
ftime=ftime Tiempo final
u(x,yIni)=val Condición de frontera
u(x,yFin)=val Condición de frontera
u(xIni,y)=val Condición de frontera
u(xFIn,y)=val Condición de frontera
```

Figura 2: Parámetros de entrada

El **RF2 Realizar simulación utilizando el lenguaje de programación Java** posibilita luego de haber creado un modelo matemático, realizar una o varias simulaciones usando el lenguaje de programación Java.

El **RF3 Realizar simulación utilizando el asistente matemático MatLab** permite luego de haber creado un modelo matemático, realizar una o varias simulaciones usando asistente matemático MatLab.

El **RF4 Obtener valor para un punto determinado** permite a partir de la solución obtenida por el Método de Elementos Finitos, evaluar la ecuación en un punto determinado y obtener la solución para ese punto.

2.3.2 Requisitos No Funcionales

Son propiedades o cualidades que el producto debe tener. Estas propiedades son las características que hacen al producto atractivo, usable, rápido o confiable. Por lo general, están vinculados a requisitos funcionales, es decir, una vez que se conozca lo que el sistema debe hacer, se puede determinar cómo ha de comportarse, qué cualidades debe tener o cuán rápido o grande debe ser. En muchos casos los requisitos no funcionales son fundamentales en el éxito del producto. A continuación se muestran los requisitos no funcionales identificados:

Software

- Se debe tener instalado en las computadoras cualquier distribución de los sistemas operativos Linux o Windows.
- Tiene que estar instalada la máquina virtual de Java 1.6.

- Tiene que estar instalado en la máquina donde se vaya a ejecutar la biblioteca de clases el asistente matemático MatLab.

Hardware

- Se debe contar con 512 MB de memoria RAM como mínimo.
- Procesadores Pentium IV o superior.
- Espacio libre en disco duro de 25 MB.

Requerimientos en el diseño y la implementación

- Lenguaje de programación Java 1.6.
- NetBeans 7.1 como IDE de desarrollo.
- Visual Paradigm para UML 8.0 como herramienta CASE.

Portabilidad

- El software deberá funcionar en los sistemas operativos Windows o cualquier distribución de Linux sobre el cual se haya instalado la máquina virtual de Java 1.6.

Soporte

- Mantenimiento: La biblioteca de clases contará con el documento Javadoc que facilitará el mantenimiento de las funcionalidades implementadas.

Luego que se identifican los requisitos funcionales se agrupan en los casos de uso, se define como caso de uso a una secuencia de interacciones que se desarrollarán entre un sistema y sus actores, en respuesta a un evento que inicia un actor principal sobre el propio sistema. Por las características de la biblioteca de clases a implementar no se determinan los Casos de Uso, ya que no existe un actor específico que juegue un determinado rol sobre una serie de responsabilidades en el tiempo de ejecución de la biblioteca de clases, derivando que no se realice la correspondiente especificación, ni el diagrama.

2.4. Diseño de la biblioteca

En la etapa de diseño se define y confecciona la arquitectura del sistema para que pueda soportar todos los requisitos funcionales y no funcionales. El propósito del diseño se puede resumir en transformar los requerimientos en un diseño que describa cómo el sistema debe ser, desarrollar una robusta arquitectura del sistema y adaptar el diseño para que se corresponda con el entorno de implementación, diseñando

sus funcionalidades. (30) En el diseño es necesario tener en cuenta los detalles concretos de la implementación del sistema por lo que sería de gran utilidad un diagrama de clases del diseño.

2.4.1 Diagrama de clases del diseño

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 encargarán del funcionamiento y la relación entre uno y otro. A continuación se presenta el diagrama de clases del diseño en donde se muestran las clases y las relaciones entre ellos de la biblioteca de clases:

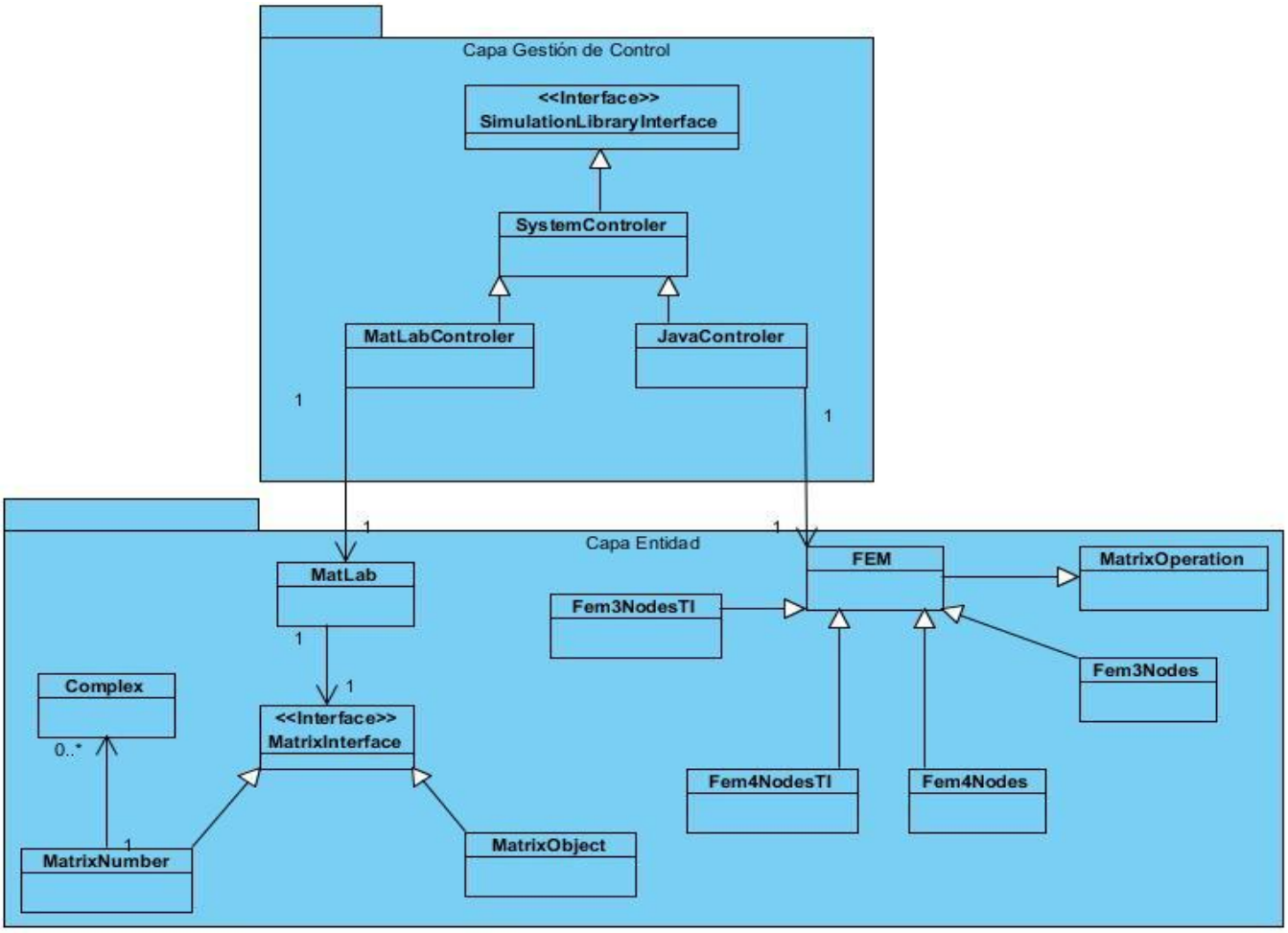

Figura 3: **Diagrama de Clases del Diseño**

Descripción de las clases del diseño

Para tener un mejor entendimiento de las clases representadas en el diagrama antes expuesto, se realiza la descripción de las clases del diseño, se muestran las clases con sus atributos y métodos y además que se explica qué función realiza cada clase

SimulationLibraryInterface: La implementación de esta interfaz permite realizar las simulaciones con el asistente matemático MatLab o con el lenguaje de programación Java.

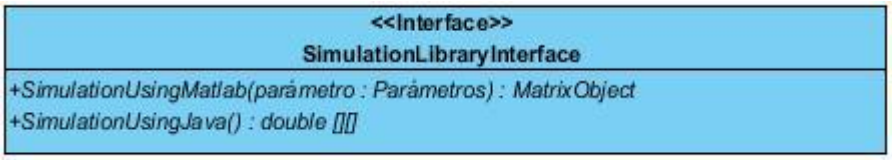

Figura 4: **Clase Interfaz SimulationLibraryInterface**

SystemControler: Es la clase encargada de implementar la Interfaz BibliotecaSimulacionInterface, permite cargar parámetros de la simulación, validar que la entrada sea correcta y contiene los métodos SimulationUsingMatlab () y SimulationUsingJava (), los cuales son los encargados de realizar la simulación.

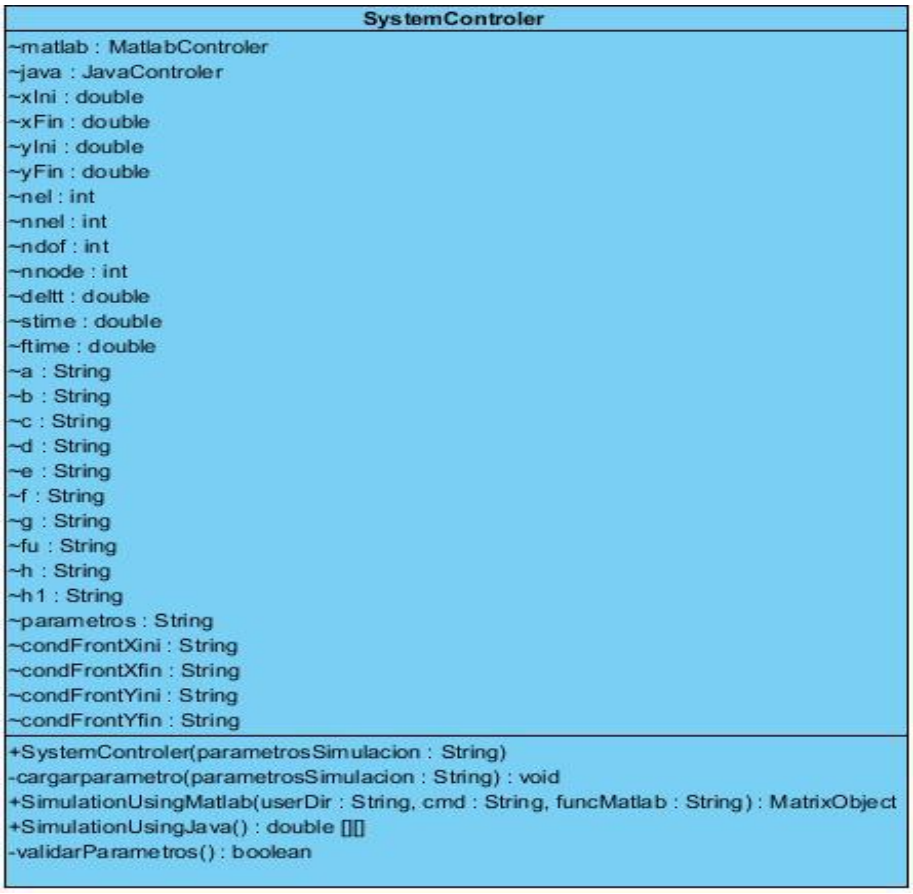

Figura 5: **Clase controladora SystemControler**

MatlabControler: Clase encargada de realizar las simulaciones usando MatLab, contiene el método ejecutar al que se le pasa por parámetros la dirección donde se va a crear un archivo temporal, la dirección donde está el ejecutable de MatLab, la dirección donde están los ficheros de MatLab y los parámetros necesarios para realizar la simulación, este método devuelve una matriz que representa la solución para cada nodo de la malla.

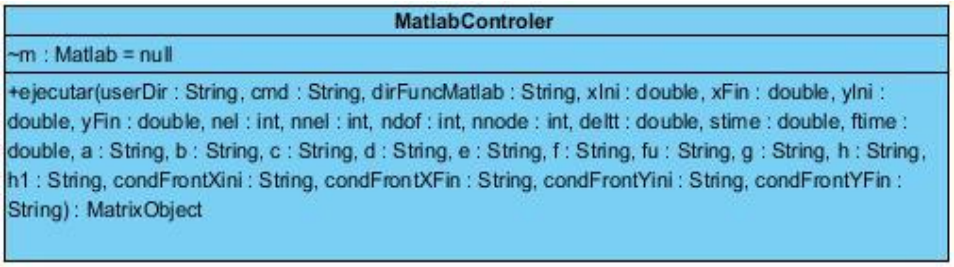

Figura 6: **Clase controladora MatLabControler**

MatLab: Clase encargada de establecer la conexión con el asistente matemático MatLab mediante el método start (), ejecutar el algoritmo a través de eval () y obtener el resultado con el método save ().

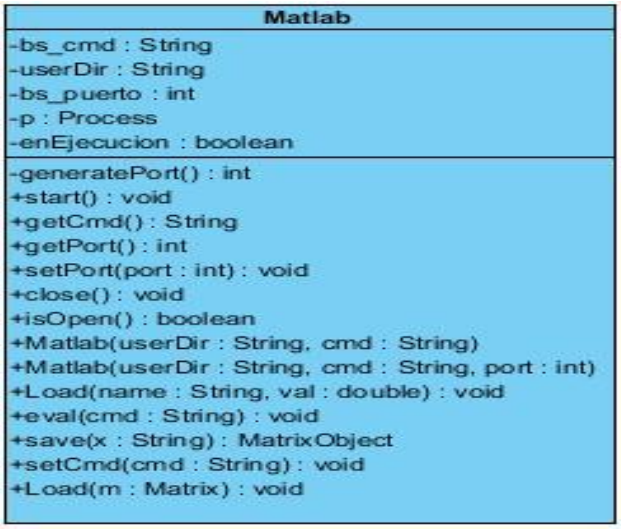

Figura 7: **Clase MatLab**

JavaControler: Clase encargada de realizar las simulaciones usando el lenguaje de programación Java, contiene el método ejecutar, al que se le pasan los parámetros de la simulación; devuelve una matriz que representa la solución para cada nodo de la malla.

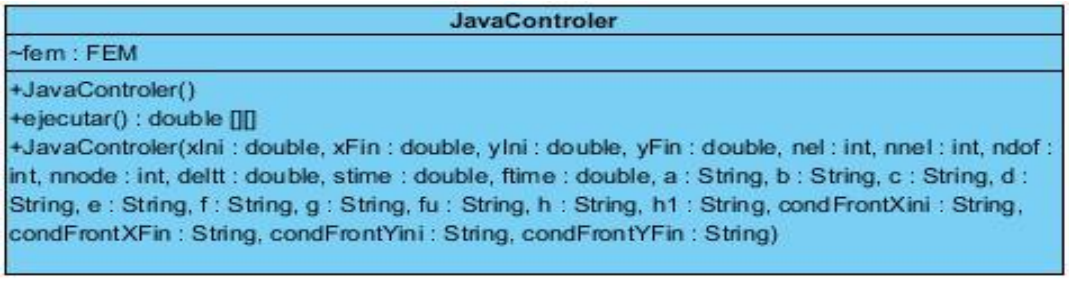

Figura 8: **Clase controladora JavaControler**

FEM: Da solución a los sistemas de ecuaciones diferenciales en derivadas parciales usando el método numérico de Elementos Finitos, tiene cuatro clases hijas la clase FEM3Nodes para resolver ecuaciones diferenciales en derivadas parciales de tipo elíptica usando elementos de tres nodos, FEM3NodesTI para resolver ecuaciones diferenciales en derivadas parciales de tipo hiperbólicas y parabólicas usando elementos de tres nodos, FEM4Nodes para resolver ecuaciones diferenciales en derivadas parciales de tipo elíptica usando elementos de cuatro nodos, FEM4NodesTI para resolver ecuaciones diferenciales en derivadas parciales de tipo hiperbólicas y parabólicas usando elementos de cuatro nodos.

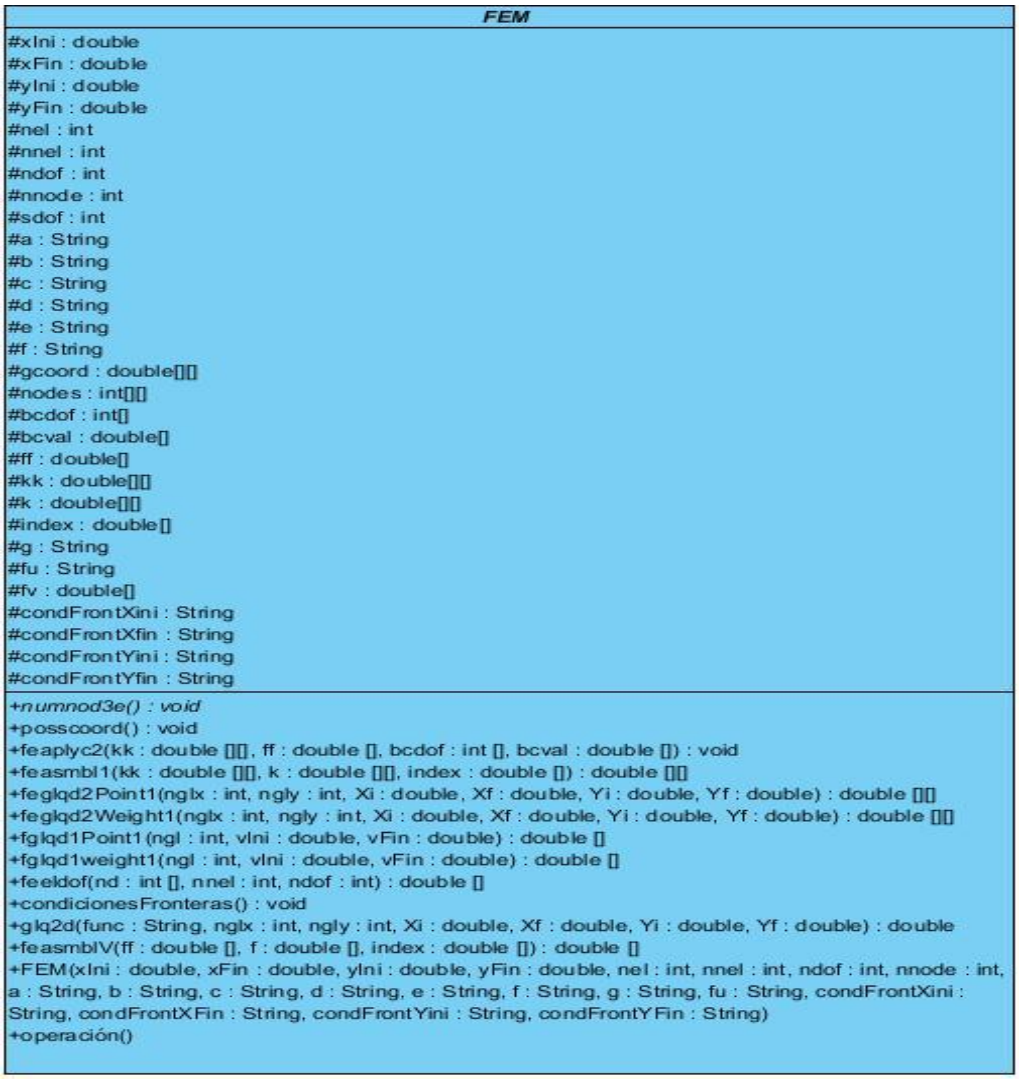

Figura 9: **Clase FEM**

MatrixOperations: Clase que contiene las operaciones entre matrices usadas por el algoritmo implementado como son: multiplicar matriz por un vector, matriz por un escalar, hallar la inversa de una matriz, entre otras operaciones.

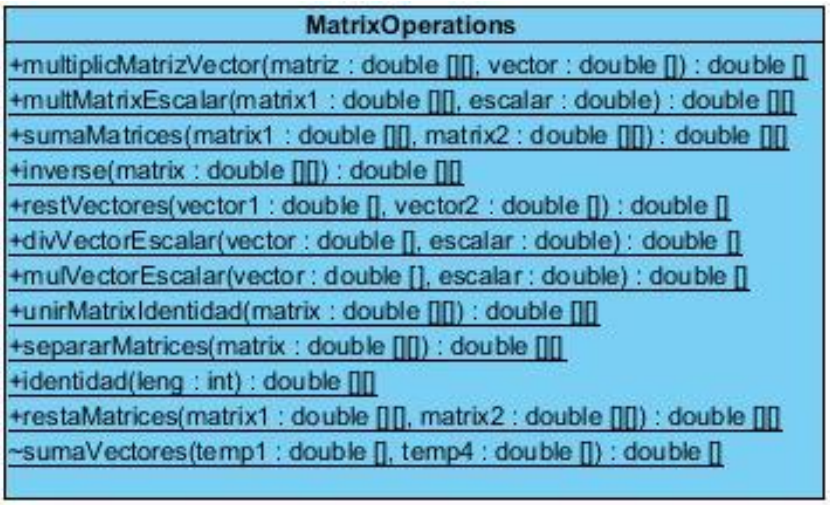

Figura 10: **Clase MatrixOperations**

Se describieron las clases más significativas dentro de la biblioteca de clases, para ver la descripción de las restantes clases ir a **Anexo 1.**

2.5. Arquitectura propuesta

Una vez realizado el diagrama del diseño, se establece la arquitectura del sistema. La arquitectura de un software indica la estructura, el funcionamiento e interacción entre las partes del software. Se define que la Arquitectura es un nivel de diseño que se centra en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema". (31)

El objetivo de la arquitectura es identificar los requisitos que producen un impacto en la estructura del sistema y reducir los riesgos asociados con la construcción del mismo. La arquitectura debe soportar los cambios futuros del software, del hardware y de las funcionalidades demandadas por los clientes.

2.5.1 Estilos arquitectónicos

Un estilo arquitectónico se define como familia de sistemas de software en términos de su organización estructural. Expresa componentes y las relaciones entre estos, con las restricciones de su aplicación y la composición asociada, así como también las reglas para su construcción. Se consideran como un tipo particular de estructura fundamental para un sistema de software, conjuntamente con un método asociado que especifica cómo construirlo incluyendo información acerca de cuándo usar la arquitectura que describe, sus invariantes y especializaciones, así como las consecuencias de su aplicación. (32)

Existen varios tipos de estilos arquitectónicos entre los que se encuentran el Flujo de datos, Llamada y Retorno, Centrado de datos, Código Móvil y Peer To Peer.

Llamada y Retorno

Dentro de estos estilos arquitectónicos se decidió utilizar el de Llamada y Retorno, que permite construir una estructura de programa relativamente fácil de modificar y ajustar. Esta familia de estilos enfatiza la modificabilidad y la escalabilidad. (33) Son los estilos más generalizados en sistemas a gran escala. Miembros de este estilo son los patrones de Orientado a Objetos y Estructura Jerárquica en Capas.

Arquitectura en capas

Dentro del estilo de Llamada y Retorno se escogió el patrón arquitectónico Arquitectura en Capas, un patrón es la descripción de un problema particular y recurrente del diseño, que aparece en contexto de diseño específico y representa un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí. (34) La Arquitectura en Capas define una organización jerárquica en la que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. Al dividir un sistema en capas, cada capa puede tratarse de forma independiente, sin tener que conocer los detalles de las demás. La división de un sistema en capas facilita el diseño modular, cada capa encapsula un aspecto concreto del sistema y permite además la construcción de sistemas débilmente acoplados. Resulta más fácil sustituir la implementación de una capa sin afectar el resto del sistema.

- **Ventajas**
  - Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
  - Admite muy naturalmente optimizaciones y refinamientos.
  - Proporciona amplia reutilización.
- **Desventajas**
  - Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de rendimiento pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.
  - Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada; también se admite que la arquitectura en capas ayuda

a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

2.5.2 Vista Lógica

Esta vista sirve para soportar los requisitos funcionales del sistema, o sea, los servicios que el sistema debe proporcionar y comprende las clases, interfaces y colaboraciones (conjuntos de clases) que forman el vocabulario del problema y de la solución. Dentro de la vista lógica se pueden definir diferentes paquetes lógicos que sirven para dividir esta vista en diferentes subvistas o subsistemas.

Para la biblioteca de clases a implementar se definieron dos paquetes lógicos, Capa Gestión de Control y la Capa Entidad, en la primera se encuentran las clases encargadas de controlar y dirigir el flujo de datos y en la segunda se encuentran las restantes clases como se muestra en la Figura 3.

Capa de Gestión de Control: Esta es la capa que media entre el Beneficiario que utiliza la biblioteca y las funcionalidades que esta posee, contiene las clases encargadas de gestionar todas la solicitudes hechas a la biblioteca y de dar respuesta auxiliándose de la Capa Entidad para acceder a las funcionalidades que las clases de esta capa brindan. Las clases que se encuentran en esta capa son la interfaz BibliotecaSimulacion.java y las clases SystemControler.java, MatlabControler.java y JavaControler.java ya que estas contienen los métodos y atributos necesarios para controlar el flujo de eventos en el sistema.

Capa Entidad: Es en esta capa donde se encuentran las demás clases usadas para la implementación de la biblioteca, que son las encargadas de ejecutar la lógica de la biblioteca de clases. En esta capa se encuentran clases como: FEM.java y MatLab.java

2.6. Patrones de diseño

Proveen un esquema para refinar los subsistemas componentes de un sistema de software, o las relaciones entre ellos. Describen la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. (32)

Los patrones de diseño son menores en escala que los patrones arquitectónicos, están contenidos dentro de los patrones de arquitectura y tienden a ser independiente de los lenguajes y paradigmas de programación. Su aplicación no tiene efectos en la estructura fundamental del sistema, pero si sobre la de un subsistema, debido a que especifican a un mayor nivel de detalle, sin llegar a la implementación, el comportamiento de los componentes del subsistema. (32)

Entre los patrones de diseño se encuentran los patrones de asignación de responsabilidades GRASP.

2.6.1 Patrones GRASP (asignación de responsabilidades)

Describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. (35)

Entre los patrones de asignación de responsabilidades más usados y que se usaron en la biblioteca de clase se encuentran:

- **Experto:** Se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele ser útil en el diseño orientado a objetos. El patrón Experto asigna una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad. (36) Con este patrón se conserva el encapsulamiento, ya que los objetos se valen de su propia información para hacer lo que se les pide, además que provee un bajo nivel de acoplamiento. En la biblioteca de clases a implementar se ve reflejado este patrón en la clase JavaControler ya que esta es la clase que cuenta con los atributos y métodos necesarios para realizar una simulación usando el lenguaje de programación Java y MatLabControler ya que esta clase que cuenta con los atributos y métodos necesarios para realizar una simulación usando el asistente matemático MatLab, el método que evidencia este patrón es el método ejecutar ().

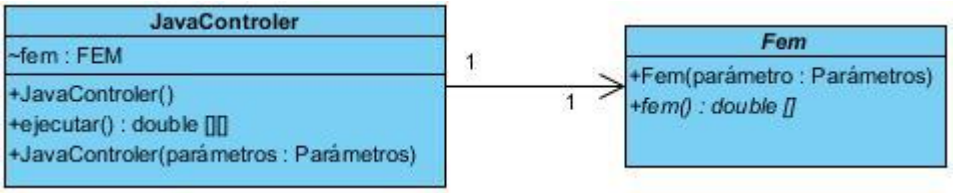

Figura 11: **Patrón Experto**

- **Creador:** El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento. (37) Al utilizar este patrón se da soporte al bajo acoplamiento. En la biblioteca de clases este patrón se ve reflejado en la clase MatlabControler , la cual crea instancias de la clase MatrixObject

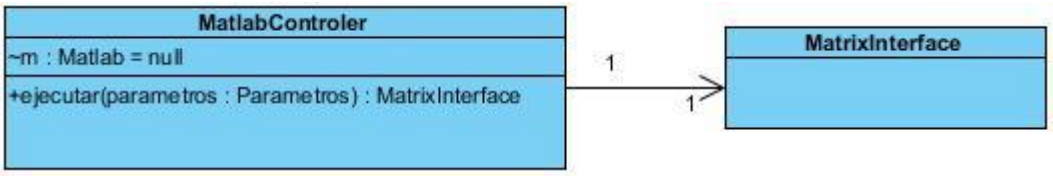

Figura 12: **Patrón Creador**

- **Controlador:** Asignar la responsabilidad del manejo de mensajes de los eventos de un sistema a una clase que represente un sistema global. Este patrón se evidencia en la biblioteca de clases a implementar en la clase SystemControler, la cual contiene los métodos necesarios para manejar los mensajes de eventos estos métodos son SimulationUsingMatlab () y SimulationUsingJava ().

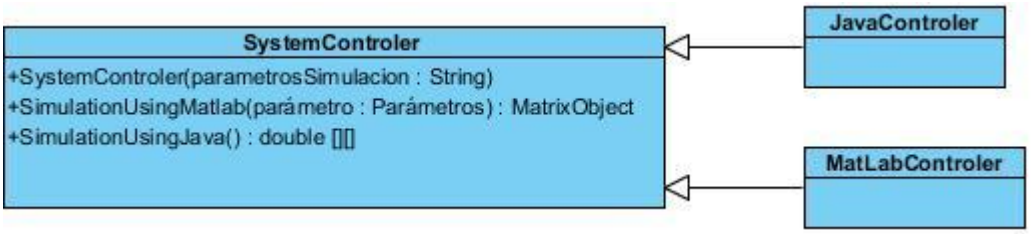

Figura 13: **Patrón Controlador**

- **Bajo Acoplamiento:** Con el uso de este patrón se garantiza tener las clases lo menos ligadas entre sí, de tal forma que en caso de producirse una modificación en alguna de ellas, se tenga la mínima repercusión posible en el resto de clases, potenciando la reutilización, y disminuyendo la dependencia entre las clases. El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases. (36)

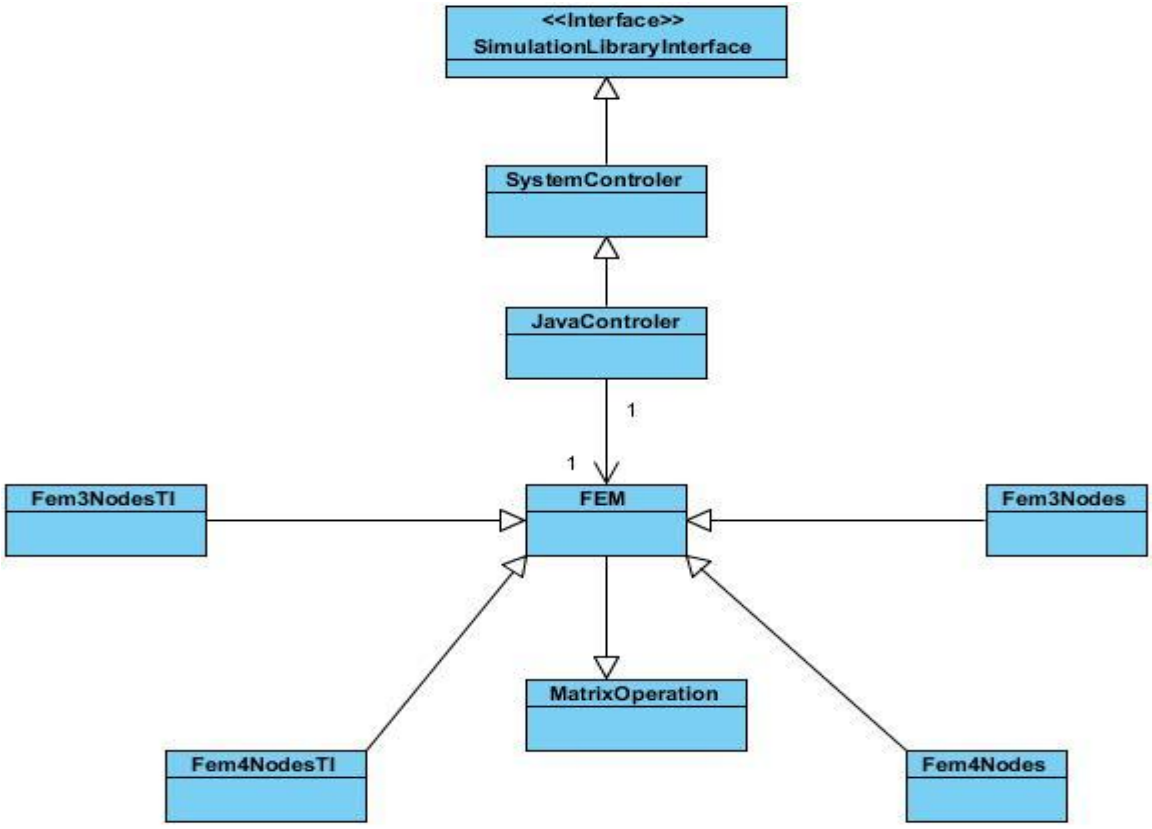

Figura 14: Patrón Bajo Acoplamiento

- **Alta Cohesión:** Este patrón mantiene la complejidad dentro de límites manejables, en otras palabras, asignar responsabilidades de modo que la cohesión siga siendo alta. La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme, lo cual mejora la claridad y facilidad con que se entiende el diseño, se simplifica el mantenimiento y las mejoras de funcionalidad, se puede generar un bajo acoplamiento y soporta mayor capacidad de reutilización. (36)

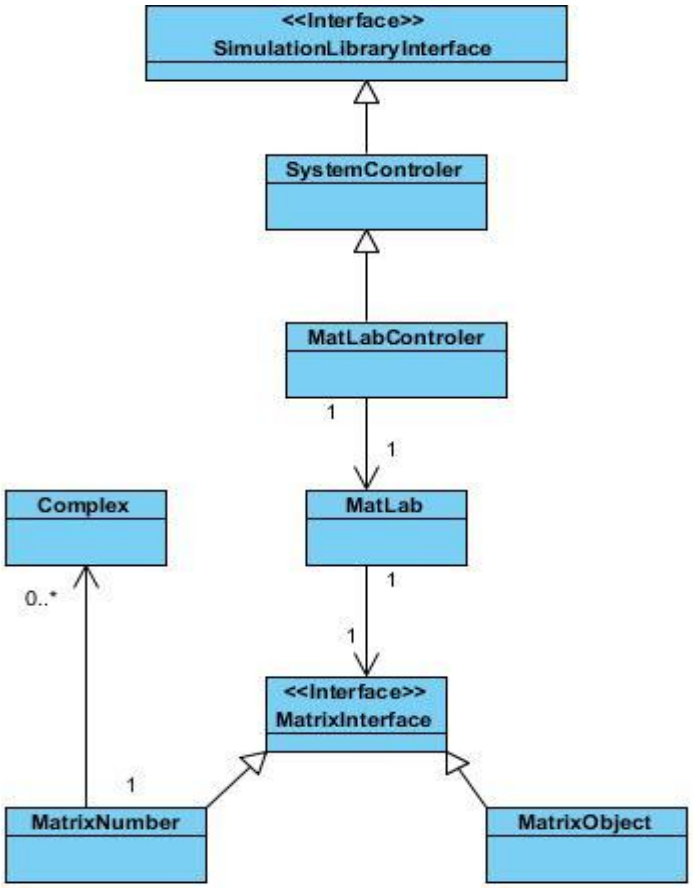

Figura 15: Patrón Alta Cohesión

2.7. Conclusiones

En el presente capítulo se definieron los artefactos correspondientes a los requisitos de la metodología de desarrollo OpenUP, y aprovechando su extensibilidad, se creó el modelo conceptual del sistema y se describieron los objetos del dominio del problema. Se identificaron 4 requisitos funcionales y 11 requisitos no funcionales. Se dio a conocer una breve descripción de la biblioteca de clases a implementar, además se realizó una propuesta de diseño de la biblioteca donde se elaboró el diagrama de clases del diseño, haciendo una descripción detallada de las principales clases. Se propuso una estructura arquitectónica donde se escogió como estilo arquitectónico el de Llamada y Retorno dentro de este, el patrón Arquitectura en Capas, donde las capas identificadas fueron la capa de Gestión de Control donde se encuentran las clases controladoras y la Capa Entidad que contiene las demás clases utilizadas para el desarrollo de la biblioteca de clases.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBAS DE SOFTWARE

Introducción

Una vez que se concluye el modelo del diseño, se dispone de los detalles suficientes para proceder a la construcción del sistema, luego se realiza la verificación del cumplimiento de los requisitos funcionales mediante las pruebas de software. En el presente capítulo se explica brevemente la implementación de la biblioteca de clases y se determinan los tipos de pruebas a realizar y los casos de pruebas que serán aplicados al sistema.

3.1 Implementación de la biblioteca de clases

En el proceso de desarrollo de un software es necesario tener en cuenta un conjunto de reglas y normas que rijan la implementación del código, permitiendo que el mismo sea legible y entendible para otros desarrolladores, esto se logra a partir de los estándares de codificación.

3.1.1 Estándares de codificación

Un estándar de codificación son reglas que se siguen para la escritura del código fuente. De tal manera que a otros programadores se les facilite el entendimiento del código (identificar las variables, las funciones o métodos).

Clases

Los nombres de clases deben ser mezclas de mayúsculas y minúsculas, con la primera letra de cada palabra interna en mayúsculas (*CamelCase*[4]). Debemos intentar mantener los nombres de clases simples y descriptivos.

```
public class ClaseEjemplo extends Object {
 int variable1;
 int variable2;
 public ClaseEjemplo() {
  variable1 = 0;
  variable2 = 1;
 }
 ...
}
```

[4] **CamelCase:** es un estilo de escritura que se aplica a frases o palabras compuestas. El nombre se debe a que las mayúsculas a lo largo de una palabra en CamelCase se asemejan a las jorobas de un camello.

Métodos

Los métodos deberán ser verbos (en infinitivo), en mayúsculas y minúsculas con la primera letra del nombre en minúsculas, y con la primera letra de cada palabra interna en mayúsculas (*lowerCamelCase*[5]). No se permiten caracteres especiales. El nombre ha de ser lo suficientemente descriptivo, no importando la longitud del mismo.

```
public void insertaUnidad(Unidad unidad);
public void eliminaAgenda(Agenda agenda);
public void actualizaTramite(Tramite tramite);
```

Variables

Los nombres de las variables tanto de instancia como estáticas reciben el mismo tratamiento que para los métodos, con la salvedad de que aquí importa su longitud y que no sea una palabra reservada del sistema. Se evitará en la medida de lo posible la utilización de caracteres especiales, así como nombre sin ningún tipo de significado funcional. Las excepciones son las variables utilizadas en bucles *for*, para esos casos se permite utilizar i, j, k, l y siempre en ese orden de anidamiento. El primer bucle siempre será el que tenga la variable i como iterador.

```
Unidad unidad;
Agenda agenda;
Tramite tramite;
```

Sentencias

Cada línea debe contener como máximo una sentencia. Ejemplo,

```
int contador++;
int variable--;
```

Las sentencias pertenecientes a un bloque de código estarán tabuladas un nivel más a la derecha con respecto a la sentencia que las contiene. El caracter inicio de bloque "{" debe situarse al final de la línea que inicia el bloque. El caracter final de bloque "}" debe situarse en una nueva línea tras la última línea del bloque y alineada con respecto al primer caracter de dicho bloque. Todas las sentencias de un bloque

[5] **lowerCamelCase:** Se refiere a que sólo la primera letra del nombre de la entidad es en minúscula.

deben encerrarse entre llaves "{...}", aunque el bloque conste de una única sentencia. Esta práctica permite añadir código sin cometer errores accidentalmente al olvidar añadir las llaves. Ejemplo,

```
if (condicion) {
 variable++;
}
```

La sentencia "try/catch" siempre debe tener el formato siguiente,

```
try {
    sentencias;
} catch (ClaseException e) {
    sentencias;
}
```

En el bloque "catch" siempre se imprimirá una traza de error indicando el tipo de excepción generada y posteriormente se elevará dicha excepción al código que lo invoca, salvo que la lógica de ejecución de la aplicación no lo requiera.

La declaración de los bucles *for* será usualmente de la forma:

```
for (inicialización; condición ; actualización)
```

Son obligatorias las tres condiciones del bucle *for*: inicialización, condición de finalización y actualización del valor de la variable de avance. La variable de avance del bucle nunca podrá ser modificada dentro del propio bucle

3.1.2 Principales métodos implementados

El modelo de implementación describe como los elementos del modelo de diseño, como las clases, se implementan en términos de ficheros de código fuente y ejecutables. El modelo de implementación describe también como se organizan los componentes de acuerdo con los mecanismos de estructuración y modularización disponibles en el entorno de implementación y en el lenguaje de programación utilizados, y como dependen los componentes unos de otros. (38)

Para que la biblioteca cumpla con sus funciones y se puedan obtener los resultados esperados, es necesario implementar varios métodos que son fundamentales para realizar las simulaciones:

- Método **fem** de la clase FEM3Nodes, resuelve las ecuaciones diferenciales en derivadas parciales de tipo elíptica utilizando el método de elementos finitos a partir del lenguaje de programación Java.
- Función **fem3gen** de MatLab, resuelve las ecuaciones diferenciales en derivadas parciales de tipo elíptica utilizando el método de elementos finitos a partir del asistente matemático MatLab.

Además de estos métodos se utilizó:

- Método **ejecutar** de la clase JavaControler, manda a ejecutar los métodos fem en dependencia de los tipos de ecuaciones diferenciales que sean haciendo uso del lenguaje de programación Java.
- Método **ejecutar** de la clase MatlabControler, manda a ejecutar los métodos fem en dependencia de los tipos de ecuaciones diferenciales que sean utilizando el asistente matemático MatLab.

A continuación se muestran varios ejemplos del código fuente de algunos métodos antes mencionados que fueron utilizados en la biblioteca de clases. Los restantes códigos fuentes se muestran en el **Anexo 2.**

Método fem

Este método pertenece a la clase FEM3Nodes que contiene los algoritmos necesarios para resolver las ecuaciones diferenciales en derivadas parciales de tipo elíptica; tiene los métodos como calmate3 que permite calcular las matrices y vectores de los elementos.

El método fem utiliza los valores de los intervalos de “x” y de “y” para crear una malla de elementos triangulares sobre la cual se resolverá la ecuación, después para cada elemento calculará sus matrices y vectores, al ensamblar dichas matrices quedará un sistema de ecuaciones lineales que será resuelto hallando la inversa de la matriz y multiplicándola por el vector. Devolviendo un vector que sería la solución para cada nodo de la malla.

Código fuente del método fem de la clase FEM3Nodes

```
public double[] fem() throws JepException {
 double[] result;
//Crear de la malla
 posscoord();
 numnod3e();
//Cargar las condiciones de fronteras
 condicionesFronteras();
//Cargar el flujo en las fronteras
condicionesFronterasflujo();
double[] flux;
```

```
 int[] nd = new int[4];
//Recorrer cada elemento
 for (int iel = 1; iel <= nel; iel++) {
//Obtener los nodos para cada elemento
 nd[1] = nodes[iel][1];
 nd[2] = nodes[iel][2];
 nd[3] = nodes[iel][3];
 //Obtener las coordenadas de cada nodo
 double x1 = gcoord[nd[1]][1];
 double y1 = gcoord[nd[1]][2];
 double x2 = gcoord[nd[2]][1];
 double y2 = gcoord[nd[2]][2];
 double x3 = gcoord[nd[3]][1];
 double y3 = gcoord[nd[3]][2];
 //Extraer los grados de libertad del sistema para el elemento
 index = feeldof(nd, nnel, ndof);
 //Calcular la matriz del elemento
 k = calmate(a, b, c, d, e, f, g, x1, x2, x3, y1, y2, y3);
 //Calcular el vector del elemento
 fv = fv3e(fu, x1, x2, x3, y1, y2, y3);
//Calcular el vector con el flujo del elemento
flux = aplyFluxboundary(nd);
//Unir Vector flujo con el vector del elemento
fv = MatrixOperations.sumaVectores(fv, flux);
 //Ensamblar las matrices y vectores de los elementos
 ff = feasmblV(ff, fv, index);
 kk = feasmbl1(kk, k, index);
 }
 //Aplicar las condiciones de fronteras
 feaplyc2(kk, ff, bcdof, bcval);
 //Resolver sistemas de ecuaciones lineales
 Matrix A = new Matrix(kk);
 Matrix inverse = A.inverse();
 double[][] array = inverse.getArray();
 result = MatrixOperations.multiplicMatrizVector(array, ff);
 return result;
 }
```

Tabla 1: Método fem de la clase FEM3Nodes

Método ejecutar

Este método pertenece a la clase JavaControler, la cual contiene una instancia de la clase FEM y los atributos necesarios para resolver la ecuación diferencial.

El algoritmo ejecutar en dependencia de los atributos ejecutará el método fem de las clases hijas de FEM realizando un casteo para acceder a dicho método.

Código fuente del método ejecutar de la clase JavaControler

```
public double[][] ejecutar() throws JepException {
 double[][] result = new double[1][1];
//Ejecutar el método fem de la clase correspondiente dependiendo del tipo de instancia
 if (fem instanceof FEM3Nodes) {
 result[0] = ((FEM3Nodes) fem).fem();
 return result;
 } else if (fem instanceof Fem4Nodes) {
 result[0] = ((Fem4Nodes) fem).fem();
 return result;
 } else if (fem instanceof FEM3NodesTI) {
 result = ((FEM3NodesTI) fem).fem();
 return result;
 } else if (fem instanceof FEM4NodesTI) {
 result = ((FEM4NodesTI) fem).fem();
 return result;
 }
 return null;
 }
```

Tabla 2: Método ejecutar de la clase JavaControler

3.2 Pruebas de Software

La prueba es el proceso de ejecución de un programa con la intención de descubrir errores previos a la entrega al usuario final. Es una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requisitos específicos, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente. (39)

Las pruebas son aplicadas para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Los niveles de pruebas más comunes son:

- **Nivel de Unidad**
  - Enfocada al código fuente de los componentes.
  - Se utiliza para verificar todos los flujos de control.
  - Primero pasa por la revisión del programador.
- **Nivel de Integración**
  - Prueba los componentes combinados para ejecutar un CU.

- Se utiliza para verificar, descubrir errores o que estén incompletas las especificaciones de las interfaces de las clases.

➢ **Nivel de Sistema**

- Prueba el software funcionando como un todo.
- Se utiliza cuando el software se encuentra en la Fase de Construcción.

➢ **Nivel de Aceptación**

- Prueba final antes del despliegue del sistema.
- Generalmente lo realizan los usuarios finales.

Para validar la biblioteca, de acuerdo a las características de los niveles de prueba presentados anteriormente se seleccionó el Nivel de Unidad y el Nivel de Sistema.

3.2.1 Pruebas de Unidad

Las pruebas de unidad están enfocadas al código fuente de los componentes, donde se verifican todos los flujos de control. Dentro de las pruebas de Unidad se escogió el método de prueba de Caja Blanca por las características propias de la biblioteca, pues no tiene una interfaz gráfica a probar y además este tipo de métodos de pruebas requieren del conocimiento de la estructura interna del programa y son derivadas a partir de las especificaciones internas de diseño o el código. Mediante los métodos de prueba de caja blanca, se puede obtener casos de prueba que garanticen que:

1) Se ejerciten por lo menos una vez todos los caminos independientes para cada módulo.
2) Se ejerciten todas las decisiones lógicas en sus vertientes verdaderas y falsa.
3) Ejecuten todos los bucles en sus límites y con sus límites operacionales.
4) Se ejerciten las estructuras internas de datos para asegurar su validez.

La prueba de caja blanca se realiza para comprobar los caminos lógicos proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles. Se puede examinar el estado del programa en varios puntos para determinar si el estado real coinciden con el esperado o mencionado.

Además de que se considera a la prueba de Caja Blanca como uno de los tipos de pruebas más importantes que se le aplican a los software, logrando como resultado que disminuya en un gran porciento el número de errores existentes en los sistemas y por ende una mayor calidad y confiabilidad. (39)

Dentro de las técnicas del método de prueba de Caja Blanca se encuentran:

✓ **La prueba del camino básico:** Permite obtener una medida de la complejidad de un diseño procedimental, y utilizar esta medida como guía para la definición de una serie de caminos básicos

de ejecución, diseñando casos de prueba que garanticen que cada camino se ejecuta al menos una vez. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática.

- ✓ **La prueba de condición**: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.
- ✓ **La prueba de flujo de datos:** Se selecciona caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.
- ✓ **La prueba de bucles:** Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

Aplicación de la técnica Camino Básico

Para comprobar el correcto funcionamiento de la biblioteca de clases se decidió utilizar la técnica de Camino Básico perteneciente al método de prueba Caja Blanca, esta técnica se usó sobre los métodos más significativos dentro de la biblioteca de clases. A continuación se muestra un ejemplo de la aplicación de esta técnica sobre el método fem de la clase FEM3Nodes descrito con anterioridad.

Para la aplicación de la técnica como tal se identificaron 5 sentencias procedimentales como muestra la Figura 16, a partir de las cuales se construyó el grafo de flujo que se muestra la Figura 17, que representa el flujo de control lógico de un programa y se utiliza para trazar fácilmente los caminos de éste.

```
public double[] fem() throws JepException {
    double[] result;
    posscoord();
    numnod3e();
    condicionesFronteras();
    int[] nd = new int[4];
    for (int iel = 1; iel <= nel; iel++) {
      nd[1] = nodes[iel][1];
      nd[2] = nodes[iel][2];
      nd[3] = nodes[iel][3];
      double x1 = gcoord[nd[1]][1];
      double y1 = gcoord[nd[1]][2];
      double x2 = gcoord[nd[2]][1];
      double y2 = gcoord[nd[2]][2];
      double x3 = gcoord[nd[3]][1];
      double y3 = gcoord[nd[3]][2];
      index = feeldof(nd, nnel, ndof);
      k = calmate(a, b, c, d, e, f, g, x1, x2, x3, y1, y2, y3);
      fv = fv3e(fu, x1, x2, x3, y1, y2, y3);
      ff = feasmblV(ff, fv, index);
      kk = feasmbl1(kk, k, index);
    }
    feaplyc2(kk, ff, bcdof, bcval);
    Matrix A = new Matrix(kk);
    Matrix inverse = A.inverse();
    double[][] array = inverse.getArray();
    result = MatrixOperations.multiplicMatrizVector(array, ff);
    return result;
  }
```


Figura 16: Código fuente del Método fem con las sentencias procedimentales identificadas

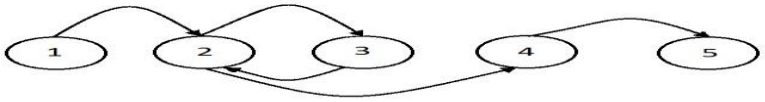

Figura 17: Grafo de Flujo del método fem de la clase FEM3Nodes

Luego de realizado el grafo de flujo correspondiente al método analizado, se identifican los nodos predicado que en el caso estudiado es el nodo 2 y se procede a calcular la complejidad ciclomática; para tener una medición cuantitativa de la complejidad lógica de la biblioteca de clases, definir el número de caminos independientes y obtener un límite superior para el número de pruebas que se deben realizar con el objetivo de asegurar que se ejecute cada sentencia al menos una vez. (39) Hay tres formas fundamentales de calcular la complejidad ciclomática:

- *V (G) = A - N + 2* donde: “A” es el número de aristas del grafo y “N” es el número de nodos.
- *V (G) = P* + 1 donde: “P” es el número de nodos predicado contenidos en el grafo.
- *V (G) = R* donde: “R” es la cantidad de regiones.
- *V (G) = A - N + 2 = 5 – 5 + 2 = 2*
- *V (G) = P + 1 = 1+1=2*
- *V (G) = R =2*

Al aplicar las tres fórmulas de calcular la complejidad ciclomática se obtuvo como resultado V (G) = 2. Fueron identificados dos caminos independientes:

- **Camino 1:** 1, 2, 3, 4, 5
- **Camino 2:** 1, 2, 4, 5

Derivación de los casos de prueba para cada camino independiente

Luego de tener elaborados los Grafos de Flujos y los caminos independientes a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de forma que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. Los casos de prueba que continuación se presentan fueron tomados del libro *The Finite Element Method using MatLab* de los autores Young W. Kwon, y Hyochoong Bang, del año 1997.

Número de camino	Casos de Prueba	Objetivo	Resultado esperado	Resultado prueba
1	(1) * u,xx + (1) * u,yy + (0) * u,xy + (0) * u,yx + (0) * u,x + (0) * u,y + (0) * u + (0) = (0) * u,t + (0) * u,tt x=[0,5] y=[0,10] nel=32 nnel=3 nnode=25 deltt=0 stime=0 ftime=10 u(x,yIni)=0 u(x,yFin)=100*sin((3.14*x)/10) u(xIni,y)=0 u(xFIn,y)=0	Probar el cálculo de ecuaciones diferenciales en derivadas parciales de tipo elíptica, con una malla de 32 elementos de 3 nodos.	0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.76 1.11 0.81 0.0 0.0 3.17 4.78 3.72 0.0 0.0 11.81 19.18 17.28 0.0 0.0 38.24 70.68 92.36 99.99	Satisfactorio
2	(1) * u,xx + (1) * u,yy + (0) * u,xy + (0) * u,yx + (0) * u,x + (0) * u,y + (0) * u + (0) = (0) * u,t + (0) * u,tt x=[0,5] y=[0,10] nel=0 nnel=3 nnode=0 deltt=0 stime=0 ftime=10 u(x,yIni)=0 u(x,yFin)=100*sin((3.14*x)/10) u(xIni,y)=0	Probar el cálculo de ecuaciones diferenciales en derivadas parciales de tipo elíptica, con una malla de 0 elementos de 3 nodos.	Mensaje de error	Satisfactorio

	u(xFIn,y)=0			

Tabla 3: Casos de prueba para cada camino independiente

Una vez que se tienen los casos de prueba se pasa a la ejecución forzada de cada uno de los caminos, para ello se utilizó la herramienta JUnit, que es conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente. La prueba con jUnit arrojó los resultados que se muestran en las siguientes figuras.

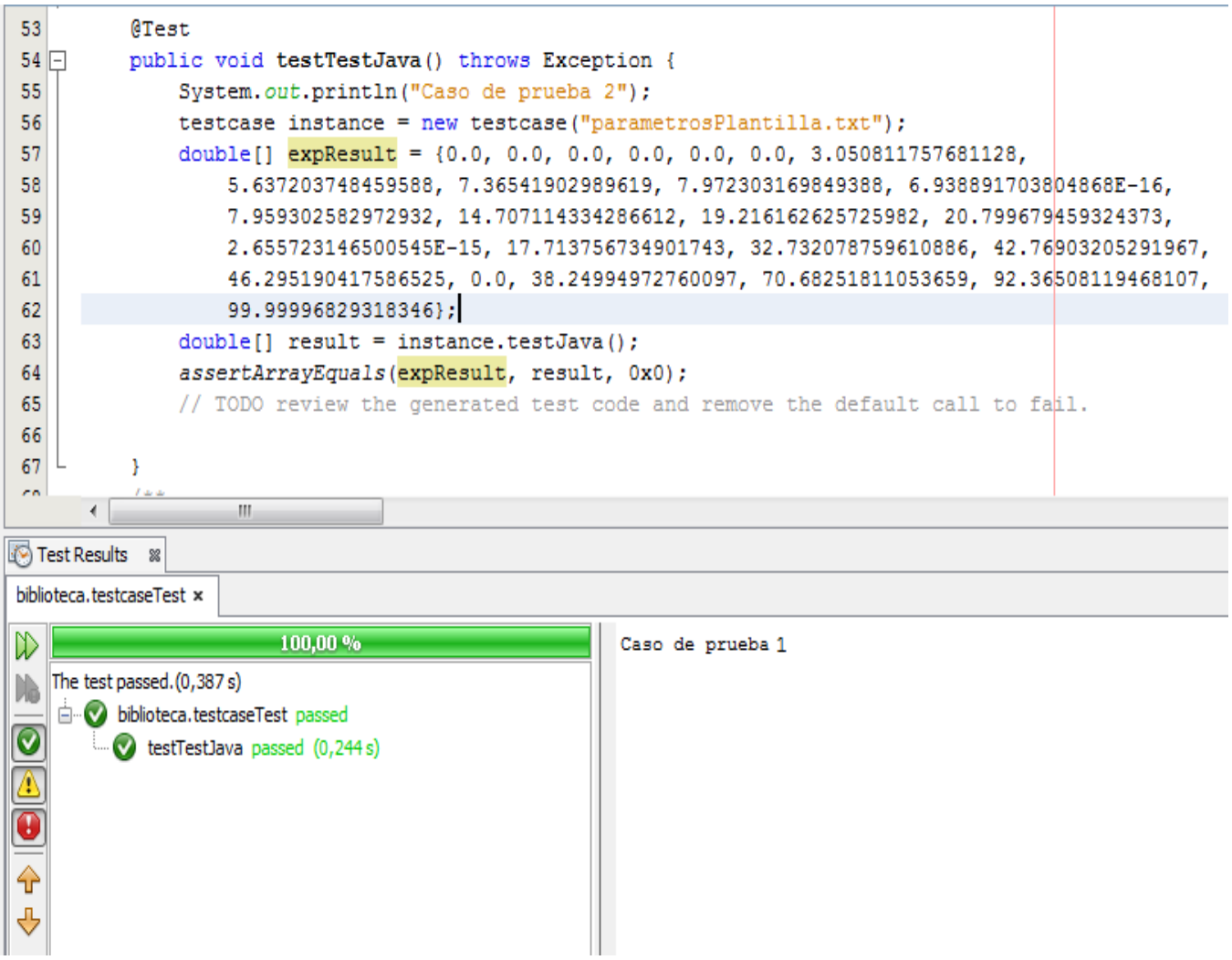

Figura 18: Caso de prueba 1

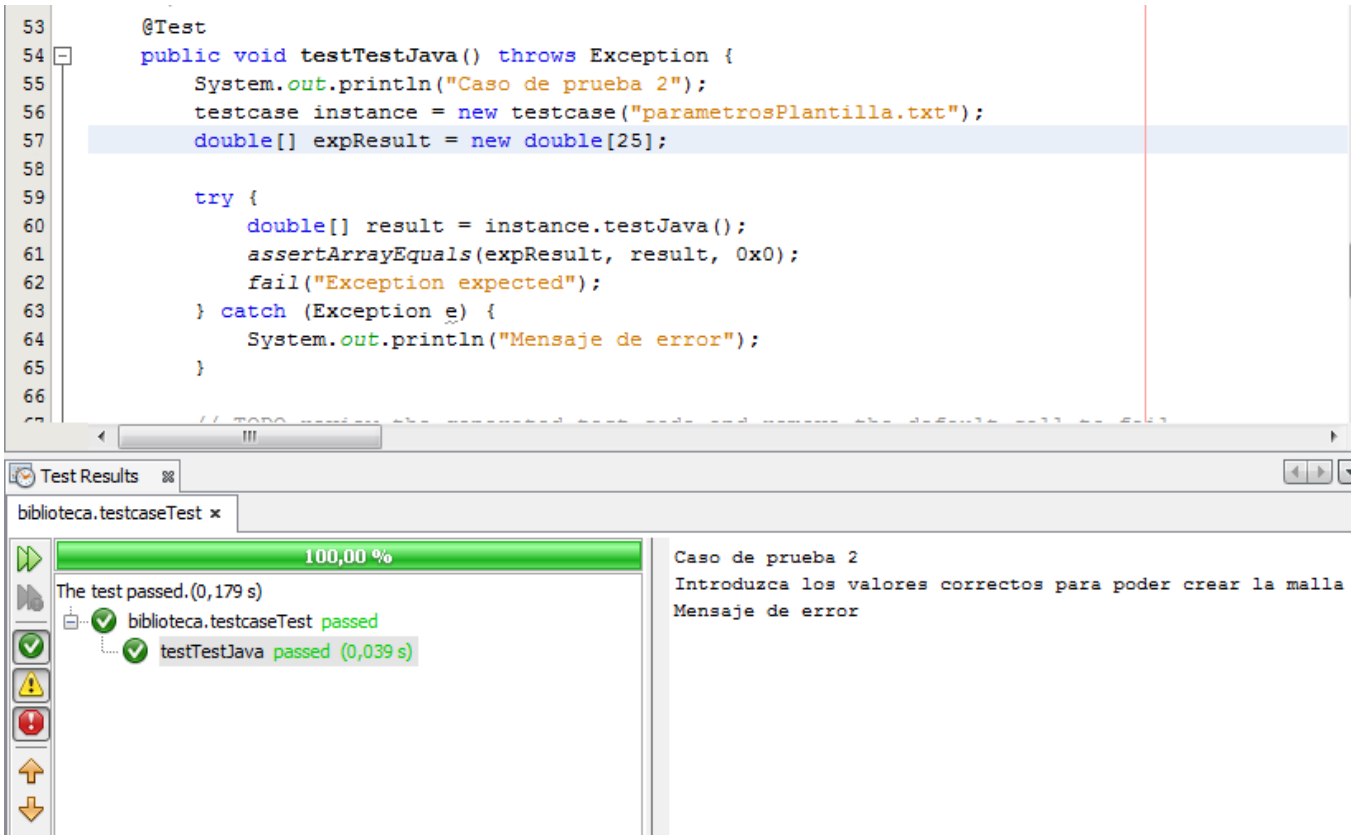

Figura 19: Caso de prueba 2

Resultado de las pruebas

Para aplicar las pruebas de unidad fueron creados 12 casos de pruebas que forzaron la ejecución de cada uno de los caminos independientes de las 4 funcionalidades más significativas, fueron ejecutados en 3 iteraciones obteniendo los resultados que se muestran en la siguiente figura:

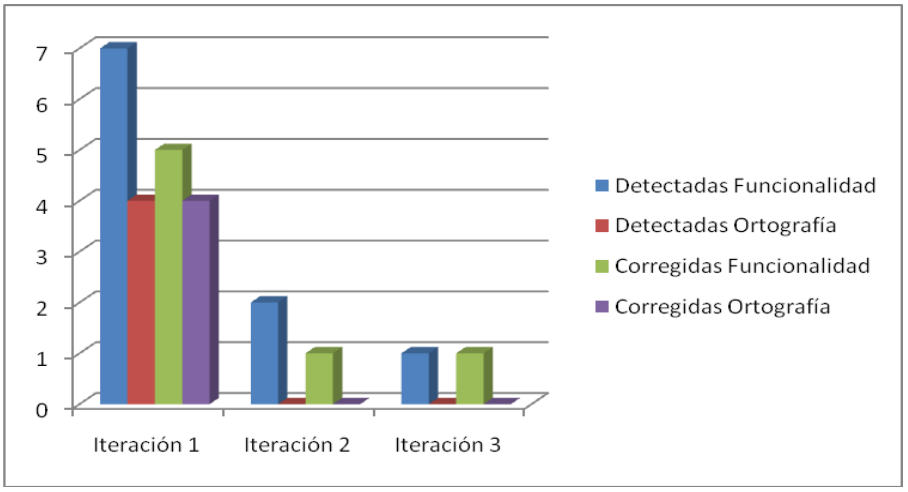

Figura 20: Resultado de las pruebas de Unidad

3.2.2 Pruebas de Sistemas

Las pruebas de sistemas se usan para probar el software funcionando como un todo y ver que se cumpla con los requisitos planteados. Dentro de estas pruebas, se escogió el método de prueba de Caja Negra para complementar las pruebas de unidades realizadas.

La prueba de caja negra se refiere a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene. A pesar que la biblioteca de clases no cuenta con una interfaz visual se utilizó la consola del IDE NetBeans para probar las funcionalidades.

Dentro del método de caja negra se escogió la técnica de prueba partición equivalente, que divide el campo de entrada de un programa en clases de datos de los que se pueden derivar casos de prueba. En otras palabras, este método intenta dividir el dominio de entrada de un programa en un número finito de clases de equivalencia. De tal modo que se pueda asumir razonablemente que una prueba realizada con un valor representativo de cada clase es equivalente, a una prueba realizada con cualquier otro valor de dicha clase. (39)

También se utilizó dentro de este método de prueba, la técnica análisis de valores límites para complementar la técnica de partición equivalente, ya que esta técnica se basa en los casos de prueba que exploran las condiciones límites, que produce mejor resultado que aquellas que no lo hacen. Las

condiciones límites son aquellas que se hallan en los márgenes de la clase de equivalencia, tanto de entrada como de salida.

Resultado de las pruebas

Para aplicar las pruebas de sistema fueron creados 17 casos de pruebas, que verificaron los 4 Requisitos Funcionales implementados. Los casos de pruebas fueron ejecutados en 3 iteraciones obteniendo los resultados que se muestran en la siguiente figura:

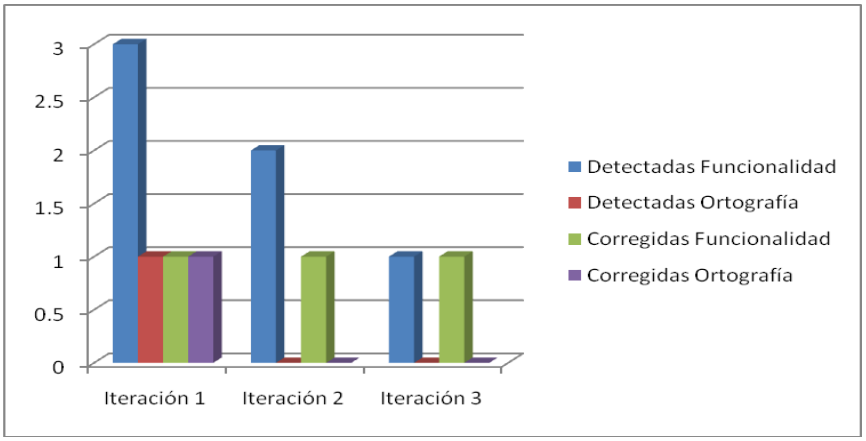

Figura 21: Resultado de las pruebas de Sistema

3.3 Conclusiones

En este capítulo fueron expuestos varios métodos fundamentales para resolver las EDPs entre los que se encuentran ejecutar () de la clase MatlabControler y el método fem () de la clase Fem3Nodes, se mostró parte del código fuente más significativo dentro de la biblioteca de clases. Se realizaron pruebas de Unidad y de Sistema, se usó la herramienta automatizada jUnit para las pruebas unitarias arrojando 11 no conformidades y de las pruebas de Sistema se obtuvieron 4, que fueron corregidas.

Conclusiones

Una vez cumplido con los objetivos propuestos, se arribó a las siguientes conclusiones:

- ✓ Se desarrolló una biblioteca que permite estandarizar la simulación de sistemas biológicos que son descritos mediante Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
- ✓ A partir de los estudios realizados sobre los métodos numéricos para la resolución de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales se decidió utilizar el Método de Elementos Finitos.
- ✓ A partir de la identificación de los requisitos y la realización de la vista lógica se logró el diseño de las funcionalidades de la biblioteca para la resolución de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales.
- ✓ Partiendo de las clases del diseño se realizó la implementación de las funcionalidades de la biblioteca para la resolución de Sistemas de Ecuaciones Diferenciales en Derivadas Parciales que puede ser utilizada en la simulación de sistemas biológicos.
- ✓ Con la realización de las pruebas de Unidad y Sistema en tres iteraciones, se validaron las funcionalidades implementadas.

Recomendaciones

- ✓ Por la complejidad que tiene la creación de mallas, se recomienda mejorar el método para crear las mismas, empleado en la biblioteca de clases, a fin de optimizar los resultados de las simulaciones.

Referencias Bibliográficas

1. **Muñoz, Emilio Profesor de Investigación del CSIC-Departamento CTS Instituto Filosofía.** Instituto Roche. [En línea] [Citado el: 24 de Octubre de 2012.] http://www.institutoroche.es/Biotecnologia_editorial/V29.html.

2. **León, Rolando Alfredo Hernández y González, Sayda Coello.** *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.* La Habana : Universitaria del Ministerio de Educación Superior, 2011. ISBN 978-959-16-1307-3. -- 110 pág..

3. *La Biología de Sistemas: ¿Un desarrollo normal de la biologia?* **Díaz, Manuel Cánovas.** 13, Murcia, España : s.n., 2 de 11 de Diciembre, 2012, Vol. 4.

4. **Torres, Dr. Néstor V.** *CAOS EN SISTEMAS BIOLÓGICOS.* Tenerife, Islas Canarias. España : s.n.

5. *La matemática y su relación con las ciencias como recurso pedagógico.* **Rodríguez, Milagros Elena.** Canaria, España : s.n., Julio, 2011, Vol. Vol. 77. 1887-1984.

6. Universidad de Alcalá (UAH). [En línea] [Citado el: 10 de Noviembre de 2012.] http://www2.uah.es/josemsalazar/material_docente_quimicas/calc/calcteor/t6/t6.pdf.

7. Universidad de Huelva. [En línea] 2005. [Citado el: 17 de Noviembre de 2012.] http://www.uhu.es/03006/ficheros/Temas/foralg4.pdf.

8. **Nakamura, Sholchlro.** *Métodos numéricos aplicados con software.*

9. **Toro, Jorge Díaz del y Alba, Aleida Perera.** *Trabajo de diploma: BIOSYS Adición de nuevos métodos de simulación y utilización e integración del asistente matemático Octave.* La Habana : s.n., 2009.

10. **Tapia, Farid Fleifel, Aranguren, Ricardo y Gascón, Manuel de la Herrán.** RED científica. [En línea] [Citado el: 10 de Diciembre de 2012.] http://www.redcientifica.com/gaia/ac/auto_c.htm.

11. **Vasquez, Ricardo Seminario.** eumed.net. [En línea] [Citado el: 7 de Enero de 2013.] http://www.eumed.net/libros-gratis/2009a/488/Que%20es%20un%20metodo%20numerico.htm.

12. **Castillero, Julián Bravo.** *Propiedades efectivas de materiales compuestos.* La Habana : s.n.

13. **GARCÍA, GUILLERMO HERNÁNDEZ.** Grupo de Modelación Matemática y Computacional. [En línea] [Citado el: 5 de Octubre de 2012.] http://mmc2.geofisica.unam.mx/cursos/hidrogeologia/NotasCurso/1-MDF1_1-10.pdf.

14. **Espinoza, C y Niño, Y.** *METODO DE ELEMENTOS FINITOS,CI71D Modelación Numérica en Ingeniería Hidráulica y Ambiental.* 2001.

15. *Una Aplicación del Método de Monte Carlo en el Análisis de Riesgo de los Proyectos: Su automatización a través de una planilla de cálculo.* **Perissé, Marcelo Claudio.** 2, Buenos Aires, Argentina : s.n., 9 de 10 de 2001, Vol. 1.

16. **Dan Pelleg y Moore, A.** *X-means: Extending K-means with Efficient estimation of the Number of Clusters.* Pittsburgh : s.n.

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

18. IBS Technologies. [En línea] 2009. [Citado el: 19 de enero de 2012.] http://www.ibssac.com/software-de-aplicacion/.

19. **Balduino, Ricardo**. [En línea] 24 de Septiembre de 2012. http://www.eclipse.org/epf/general/OpenUP.pdf.

20. **Perissé, Marcelo Claudio.** *PROYECTO INFORMÁTICO. Una Metodología Simplificada.* Buenos Aires, Argentina : s.n., febrero de 2001. ISBN: 987-43-2947-5.

21. Visual Paradigm IDE for Java. [En línea] [Citado el: 10 de Noviembre de 2012.] http://www.visual-paradigm.com/.

22. **Salinas Caro, Patricio y Histchfeld K, Nancy.** Facultad de Ciencias Fisicas y Matematicas Universidad de Chile. [En línea] [Citado el: 13 de Octubre de 2012.] http://www.dcc.uchile.cl/~psalinas/uml/..

23. Lenguajes de Programación. [En línea] 2009. [Citado el: 9 de Diciembre de 2012.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

24. **Hernández, Osvel Chávez.** *Servicio de Acoplamiento Molecular para la.* La Habana : s.n., 2012.

25. **Oracle.** Sitio Oficial de Netbeans. [En línea] [Citado el: 11 de Diciembre de 2012.] http://netbeans.org/index_es.html.

26. Introducción al entorno de desarrollo Eclipse. [En línea] [Citado el: 15 de Enero de 2012.] http://www-gris.det.uvigo.es/wiki/pub/Main/MiscResources/Manual_Eclipse.pdf.

27. Softonic. [En línea] 12 de Enero de 2012. http://eclipse-sdk.softonic.com/.

28. **Pérez, Pablo.** *Modelado matemático de los sistemas ecológicos.* Oviedo, España : s.n., 2007.

29. Ingeniería del Software I,Introducción al Modelo Conceptual,2do. Cuatrimestre. [En línea] 2005. [Citado el: 7 de Febrero de 2013.] http://www-2.dc.uba.ar/materias/isoft1/is1-2005_2/apuntes/MC.pdf.

30. **González Abreu, Luis Eduardo y Fonseca Guzmán, Mónica.** *Módulo de gestión para los reportes estadísticos de alas RIS.* Ciudad de la Habana : s.n., 2010.

31. **Garlan, David y Shaw, Mary.** *An Introduction to Software Architecture.* Pittsburgh : World Scientific, Enero, 1994.

32. **Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel.** Arquitecturas de Sofware,Guías de estudio. [En línea] [Citado el: 15 de Febrero de 2013.] http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

33. **Reynoso, Carlos y Nicolás, Kiccillof.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.Versión 1.0. Universidad de Buenos Aires. [En línea] Marzo de 2004. http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

34. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

35. **Larman, C.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : s.n.

36. **Visconti, Marcello y Astudillo, Hernán.** Fundamentos de Ingeniería de Software, Patrones de Diseño. [En línea] [Citado el: 15 de Marzo de 2013.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

37. —. *Fundamentos de Ingeniería de Software.* s.l. : Departamento de Informática. Universidad Técnica Federico Santa María.

38. **Ornelas, Raquel Ochoa.** Simulador gráfico de algoritmos de programación para computadora. Universidad de Colima, FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA. [En línea] [Citado el: 25 de Marzo de 2013.] http://digeset.ucol.mx/tesis_posgrado/Pdf/Raquel%20Ochoa%20Ornelas.pdf.

39. **Pressman, Roger S.** *Ingeniería de Software: un enfoque práctico, 6ta Edición.* Nueva York, E.U.A : Editorial McGraw-Hill, 2007. ISBN 978-0-07-337Z.

Bibliografía

1. **Muñoz, Emilio Profesor de Investigación del CSIC-Departamento CTS Instituto Filosofía.** Instituto Roche. [En línea] [Citado el: 24 de Octubre de 2012.] http://www.institutoroche.es/Biotecnologia_editorial/V29.html.

2. **León, Rolando Alfredo Hernández y González, Sayda Coello.** *EL PROCESO DE INVESTIGACIÓN CIENTÍFICA.* La Habana : Universitaria del Ministerio de Educación Superior, 2011. ISBN 978-959-16-1307-3. -- 110 pág..

3. *La Biología de Sistemas: ¿Un desarrollo normal de la biologia?* **Díaz, Manuel Cánovas.** 13, Murcia, España : s.n., 2 de 11 de Diciembre, 2012, Vol. 4.

4. **Torres, Dr. Néstor V.** *CAOS EN SISTEMAS BIOLÓGICOS.* Tenerife, Islas Canarias. España : s.n.

5. *La matemática y su relación con las ciencias como recurso pedagógico.* **Rodríguez, Milagros Elena.** Canaria, España : s.n., Julio, 2011, Vol. Vol. 77. 1887-1984.

6. Universidad de Alcalá (UAH). [En línea] [Citado el: 10 de Noviembre de 2012.] http://www2.uah.es/josemsalazar/material_docente_quimicas/calc/calcteor/t6/t6.pdf.

7. Universidad de Huelva. [En línea] 2005. [Citado el: 17 de Noviembre de 2012.] http://www.uhu.es/03006/ficheros/Temas/foralg4.pdf.

8. **Nakamura, Sholchlro.** *Métodos numéricos aplicados con software.*

9. **Toro, Jorge Díaz del y Alba, Aleida Perera.** *Trabajo de diploma: BIOSYS Adición de nuevos métodos de simulación y utilización e integración del asistente matemático Octave.* La Habana : s.n., 2009.

10. **Tapia, Farid Fleifel, Aranguren, Ricardo y Gascón, Manuel de la Herrán.** RED científica. [En línea] [Citado el: 10 de Diciembre de 2012.] http://www.redcientifica.com/gaia/ac/auto_c.htm.

11. **Vasquez, Ricardo Seminario.** eumed.net. [En línea] [Citado el: 7 de Enero de 2013.] http://www.eumed.net/libros-gratis/2009a/488/Que%20es%20un%20metodo%20numerico.htm.

12. **Castillero, Julián Bravo.** *Propiedades efectivas de materiales compuestos.* La Habana : s.n.

13. **GARCÍA, GUILLERMO HERNÁNDEZ.** Grupo de Modelación Matemática y Computacional. [En línea] [Citado el: 5 de Octubre de 2012.] http://mmc2.geofisica.unam.mx/cursos/hidrogeologia/NotasCurso/1-MDF1_1-10.pdf.

14. **Espinoza, C y Niño, Y.** *METODO DE ELEMENTOS FINITOS,CI71D Modelación Numérica en Ingeniería Hidráulica y Ambiental.* 2001.

15. *Una Aplicación del Método de Monte Carlo en el Análisis de Riesgo de los Proyectos: Su automatización a través de una planilla de cálculo.* **Perissé, Marcelo Claudio.** 2, Buenos Aires, Argentina : s.n., 9 de 10 de 2001, Vol. 1.

16. **Dan Pelleg y Moore, A.** *X-means: Extending K-means with Efficient estimation of the Number of Clusters.* Pittsburgh : s.n.

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

18. IBS Technologies. [En línea] 2009. [Citado el: 19 de enero de 2012.] http://www.ibssac.com/software-de-aplicacion/.

19. **Balduino, Ricardo**. [En línea] 24 de Septiembre de 2012. http://www.eclipse.org/epf/general/OpenUP.pdf.

20. **Perissé, Marcelo Claudio.** *PROYECTO INFORMÁTICO. Una Metodología Simplificada.* Buenos Aires, Argentina : s.n., febrero de 2001. ISBN: 987-43-2947-5.

21. Visual Paradigm IDE for Java. [En línea] [Citado el: 10 de Noviembre de 2012.] http://www.visual-paradigm.com/.

22. **Salinas Caro, Patricio y Histchfeld K, Nancy.** Facultad de Ciencias Fisicas y Matematicas Universidad de Chile. [En línea] [Citado el: 13 de Octubre de 2012.] http://www.dcc.uchile.cl/~psalinas/uml/..

23. Lenguajes de Programación. [En línea] 2009. [Citado el: 9 de Diciembre de 2012.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

24. **Hernández, Osvel Chávez.** *Servicio de Acoplamiento Molecular para la.* La Habana : s.n., 2012.

25. **Oracle.** Sitio Oficial de Netbeans. [En línea] [Citado el: 11 de Diciembre de 2012.] http://netbeans.org/index_es.html.

26. Introducción al entorno de desarrollo Eclipse. [En línea] [Citado el: 15 de Enero de 2012.] http://www-gris.det.uvigo.es/wiki/pub/Main/MiscResources/Manual_Eclipse.pdf.

27. Softonic. [En línea] 12 de Enero de 2012. http://eclipse-sdk.softonic.com/.

28. **Pérez, Pablo.** *Modelado matemático de los sistemas ecológicos.* Oviedo, España : s.n., 2007.

29. Ingeniería del Software I,Introducción al Modelo Conceptual,2do. Cuatrimestre. [En línea] 2005. [Citado el: 7 de Febrero de 2013.] http://www-2.dc.uba.ar/materias/isoft1/is1-2005_2/apuntes/MC.pdf.

30. **González Abreu, Luis Eduardo y Fonseca Guzmán, Mónica.** *Módulo de gestión para los reportes estadísticos de alas RIS.* Ciudad de la Habana : s.n., 2010.

31. **Garlan, David y Shaw, Mary.** *An Introduction to Software Architecture.* Pittsburgh : World Scientific, Enero, 1994.

32. **Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel.** Arquitecturas de Sofware,Guías de estudio. [En línea] [Citado el: 15 de Febrero de 2013.] http://prof.usb.ve/lmendoza/Documentos/PS-6116/Guia%20Arquitectura%20v.2.pdf.

33. **Reynoso, Carlos y Nicolás, Kiccillof.** Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.Versión 1.0. Universidad de Buenos Aires. [En línea] Marzo de 2004. http://www.willydev.net/InsiteCreation/v1.0/descargas/prev/estiloypatron.pdf.

34. **Gamma, Erich, y otros.** *Design Patterns: Elements of Reusable Object-Oriented Software.* 1995. ISBN 0-201-63361-2.

35. **Larman, C.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : s.n.

36. **Visconti, Marcello y Astudillo, Hernán.** Fundamentos de Ingeniería de Software, Patrones de Diseño. [En línea] [Citado el: 15 de Marzo de 2013.] http://www.inf.utfsm.cl/~visconti/ili236/Documentos/08-Patrones.pdf.

37. —. *Fundamentos de Ingeniería de Software.* s.l. : Departamento de Informática. Universidad Técnica Federico Santa María.

38. **Ornelas, Raquel Ochoa.** Simulador gráfico de algoritmos de programación para computadora. Universidad de Colima, FACULTAD DE INGENIERÍA MECÁNICA Y ELÉCTRICA. [En línea] [Citado el: 25 de Marzo de 2013.] http://digeset.ucol.mx/tesis_posgrado/Pdf/Raquel%20Ochoa%20Ornelas.pdf.

39. **Pressman, Roger S.** *Ingeniería de Software: un enfoque práctico, 6ta Edición.* Nueva York, E.U.A : Editorial McGraw-Hill, 2007. ISBN 978-0-07-337Z.

40. **Mestras, Juan Pavón.** *Patrones de diseño orientado a objetos*, Dep. Ingeniería del Software e Inteligencia Artificial. Universidad Complutense Madrid, 2004.

41. **Tello, Jesús Cáceres.** *Diagramas de Casos de Uso.* Dpto. Ciencias de la Computación.Universidad de Alcalá.

42. **Vega, Miguel.** *Casos de uso UML.* Granada, Octubre de 2010.

43. **Kwon,Young W. y Bang, Hyochoong.** *The Finite Element Method using MatLab*. University Minnesota, 1997. ISBN 0-8493-9653-0.

44. **Mecías, Ángela Mireya León.** *Introducción al Método de Elementos Finitos.* Universidad de La Habana, 2010.

45. **Peña, Jose S. Cánovas.** *Métodos numéricos para las ecuaciones diferenciales.* 18 de diciembre de 2009.

46. **Sieburg, H.B.** (1990). Physiological Studies *in-sílico*. Studies in the Sciences of Complexity 12, 321-342.

Anexos

Anexo 1: Descripción de las clases del diseño

MatrixNumber: Esta clase se usa para guardar resultados provenientes de la clase **Matlab**, en ella se pueden guardar números reales e imaginarios. Puede ser utilizada como matriz o como vector si sus columnas o filas son 1.

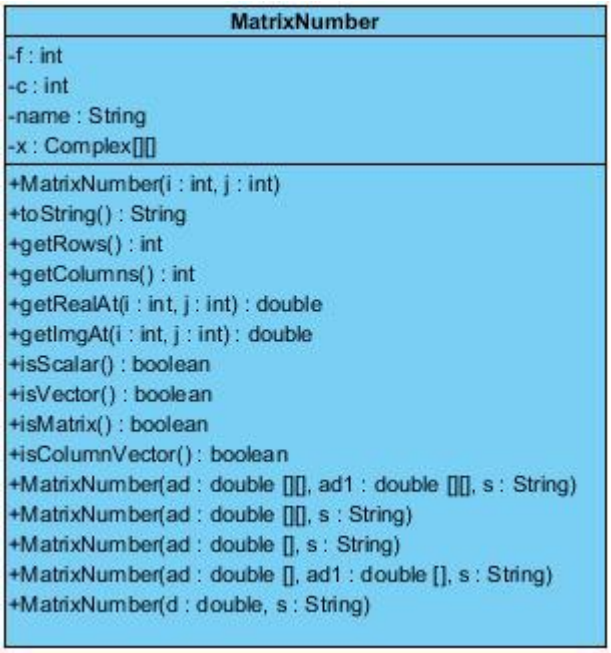

Figura 22: Clase MatrixNumber

MatrixObject: Clase que representa una matriz de cualquier tipo de dato, tiene los métodos necesarios para acceder a posiciones específicas dentro de ella.

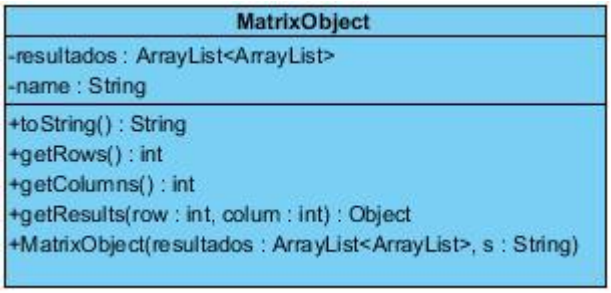

Figura 23: Clase MatrixObject

Complex: Representa los tipos de números complejos, es utilizada por la clase **MatrixNumber**.

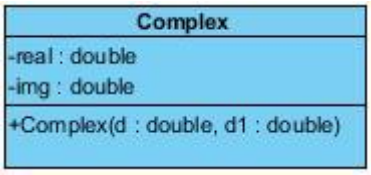

Figura 24: Clase Complex

FEM3Nodes: Es clase hija de **FEM**, su función es realizar la simulación de ecuaciones diferenciales en derivadas parciales de tipo elípticas utilizando elementos de tres nodos. El método principal encargado de solucionar las ecuaciones es **fem**. A esta clase se le pasan todos los parámetros necesarios para poder solucionar las ecuaciones diferenciales.

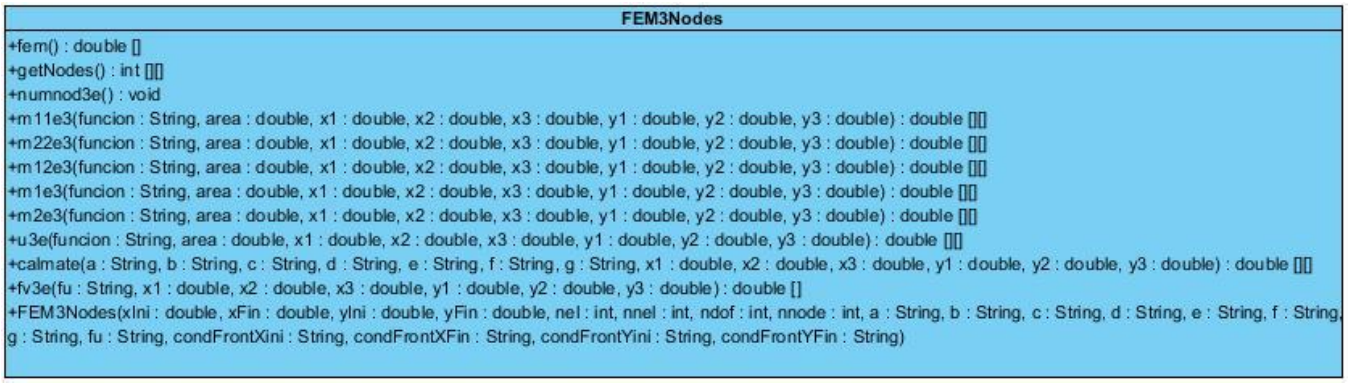

Figura 25: Clase FEM3Nodes

FEM3NodesTI: Es clase hija de **FEM** su función es realizar la simulación de ecuaciones diferenciales en derivadas parciales de tipo hiperbólicas y parabólicas utilizando elementos de tres nodos. El método principal encargado de solucionar las ecuaciones es **fem**. A esta clase se le pasan todos los parámetros necesarios para poder solucionar las ecuaciones diferenciales.

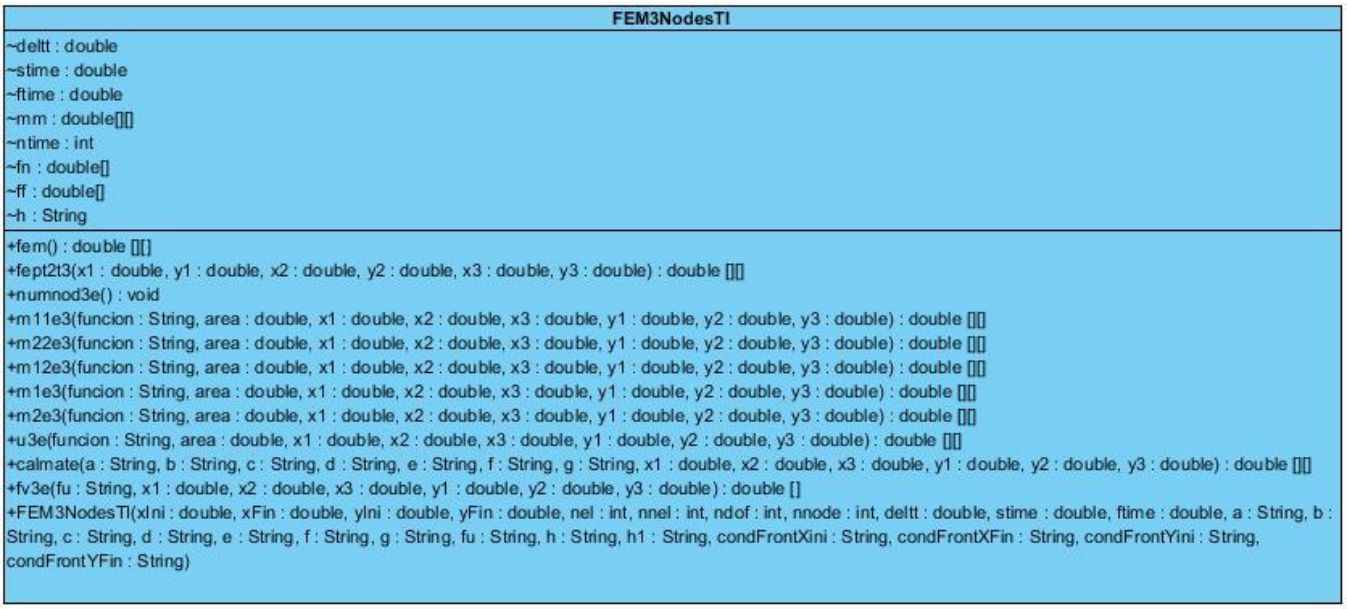

Figura 26: Clase FEM3NodesTI

Fem4Nodes: Es clase hija de **FEM** su función es realizar la simulación de ecuaciones diferenciales en derivadas parciales de tipo elípticas utilizando elementos de cuatro nodos. El método principal encargado de solucionar las ecuaciones es **fem**. A esta clase se le pasan todos los parámetros necesarios para poder solucionar las ecuaciones diferenciales.

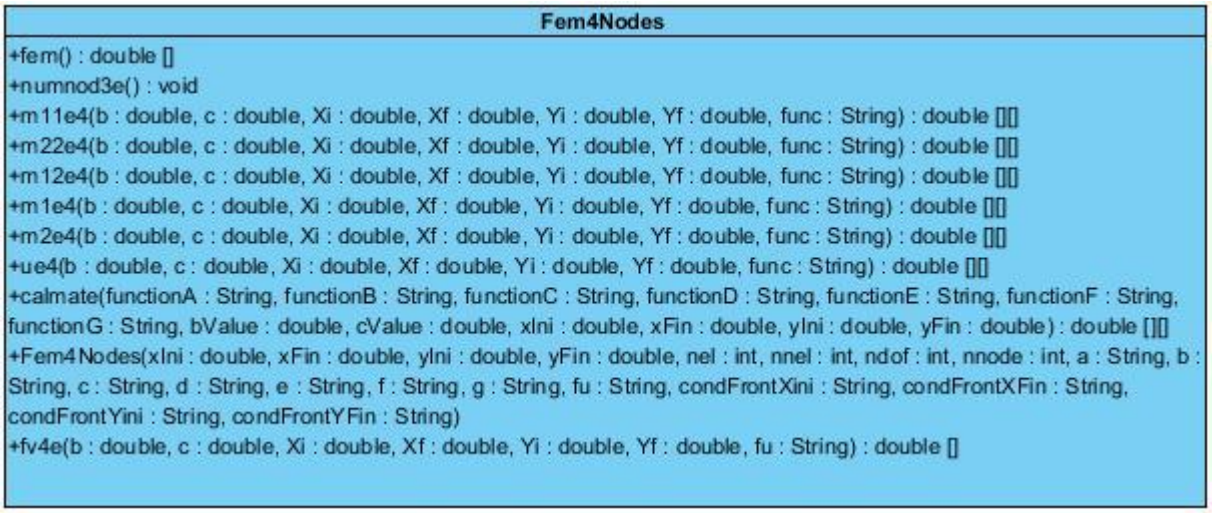

Figura 27: Clase FEM4Nodes

FEM4NodesTI: Es clase hija de **FEM** su función es realizar la simulación de ecuaciones diferenciales en derivadas parciales de tipo elípticas utilizando elementos de tres nodos. El método principal encargado de solucionar las ecuaciones es **fem**. A esta clase se le pasan todos los parámetros necesarios para poder solucionar las ecuaciones diferenciales.

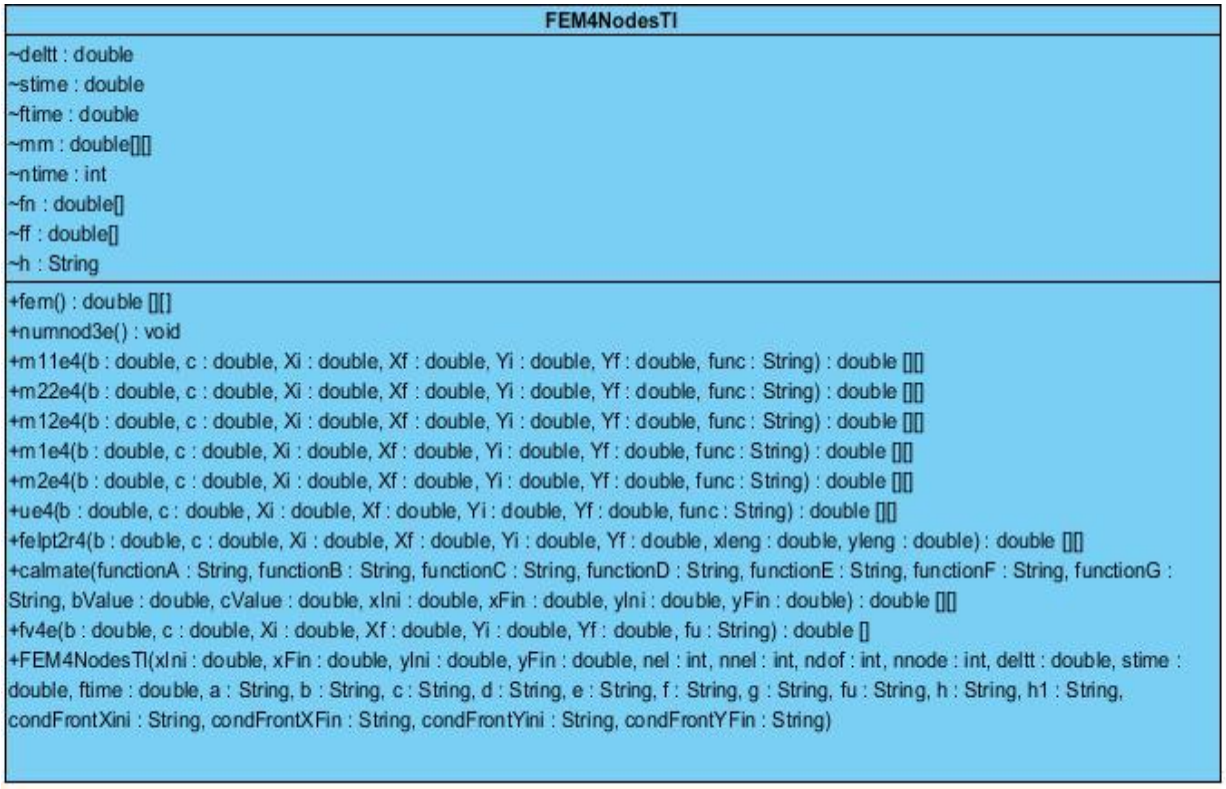

Figura 28: Clase FEMNodesTI

Anexo 2: Códigos fuente de los métodos empleados

Código fuente de la función fem3gen de la clase MatLab

```
function [ result ] = fem3gen(xini,xfin,yini,yfin,nel,nnel,ndof,nnode,deltt,stime,ftime,a,b,c,d,e,f,g,fu,h,h1,condFrontXini,condFrontXfin,
condFrontYini,condFrontYfin,condFrontXiniFlux,condFrontXfinFlux, condFrontYiniFlux,condFrontYfinFlux)
%Crear variables x y.
syms x y;
%Crear la malla.
 gcoord = posscoord( nnode,xini,xfin,yini,yfin,nnel,nel );
nodes = numnod3e( nel, gcoord,xini);
% Cargar las condiciones de fronteras y el flujo en la frontera.
[ bcdof,bcval ] = boundaryConditions( gcoord,yini,yfin,xini,xfin,condFrontXini,condFrontXfin, condFrontYini,condFrontYfin,nnode );
[ bcdofFlux,bcvalFlux ] = boundaryConditions( gcoord,yini,yfin,xini,xfin,condFrontXiniFlux,condFrontXfinFlux,
condFrontYiniFlux,condFrontYfinFlux,nnode );
%Inicializar matrices y vectores necesarios para ejecutar el método.
ff=zeros(sdof,1);
fn=zeros(sdof,1);
fsol=zeros(sdof,1);
kk=zeros(sdof,sdof);
mm=zeros(sdof,sdof);
index=zeros(nnel*ndof,1);
fv=zeros(nnel,1);
%Pasos para integración numérica.
nglx=3;
ngly=3;
for iel=1:nel % Ciclo para cada elemento
%Obtener coordenadas para cada nodo del elemento.
nd(1)=nodes(iel,1);
nd(2)=nodes(iel,2);
nd(3)=nodes(iel,3);
x1=gcoord(nd(1),1);y1=gcoord(nd(1),2);
x2=gcoord(nd(2),1);y2=gcoord(nd(2),2);
x3=gcoord(nd(3),1);y3=gcoord(nd(3),2);
% Extraer los grados de libertad del sistema para el elemento
index=feeldof(nd,nnel,ndof);
%Calcular la matriz del elemento
k= calmate3( a,b,c,d,e,f,g,x1,x2,x3,y1,y2,y3, nglx, ngly );% calcula la matriz del elemento
%Calcular del vector del elemento
fv= fv3e( x1,x2,x3,y1,y2,y3,fu );% calcula el vector del elemento
%Calcular el flujo del elemento
flux = fluxboundary(nd, bcdofFlux,bcvalFlux,gcoord );
%Unir vector del flujo con el vector del elemento
```

```
fv=fv+flux;
% Ensamblar los vectores de los elementos
ff=feasmblV( ff,fv,index );
% Ensamblar las matrices de los elementos
kk= feasmbl1(kk,k,index);
end
%Aplicar las condiciones de fronteras
[kk,ff]=feaplyc2(kk,ff,bcdof,bcval);
%Resolver la ecuación de la matriz
fsol=kk\ff;
%Devuelve la solución
result = fsol ;
end
```

Tabla 4: Función fem3gen de la clase Matlab

Código fuente del método ejecutar de la clase MatlabControler

```
public MatrixObject ejecutar(String userDir, String cmd, String dirFuncMatlab, double xIni, double xFin, double yIni, double yFin, int nel, int
nnel, int ndof, int nnode, double deltt, double stime, double ftime, String a, String b, String c, String d, String e, String f, String fu, String g,
String h, String h1, String condFrontXini, String condFrontXFin, String condFrontYini, String condFrontYFin) throws Exception {
 MatrixObject ress = null;
 String condFrontXFinflux, condFrontYFinflux, condFrontXiniflux, condFrontYiniflux;
//Verificar si son condiciones de fronteras o de flujo
 if (condFrontXFin.charAt(0) == '~') {
 condFrontXFinflux = condFrontXFin.substring(1);
 condFrontXFin = "NaN";
 } else {
 condFrontXFinflux = "NaN";
 } if (condFrontYFin.charAt(0) == '~') {
 condFrontYFinflux = condFrontYFin.substring(1);
 condFrontYFin = "NaN";
 } else {
 condFrontYFinflux = "NaN";
 } if (condFrontXini.charAt(0) == '~') {
 condFrontXiniflux = condFrontXini.substring(1);
 condFrontXini = "NaN";
 } else {
 condFrontXiniflux = "NaN";
 } if (condFrontYini.charAt(0) == '~') {
 condFrontYiniflux = condFrontYini.substring(1);
 condFrontYini = "NaN";
 } else {
 condFrontYiniflux = "NaN";
```

```
 } try {
//Ejecutar matlab
 m = new Matlab(userDir, cmd);
 m.start();
//Evaluar expresiones x y en matlab
 m.eval("syms x y ");
//Verificar los parámetros para mandar a ejecutar en dependencia de atributos
 if (nnel == 3 && deltt == 0 && h.equals("0") && h1.equals("0")) {
 m.eval("cd " + dirFuncMatlab + "\\MetodoFem3Nodos");
 m.eval("result = fem3gen(" + xIni + "," + xFin + "," + yIni + "," + yFin + "," + nel + "," + nnel + "," + ndof + "," + nnode + "," + deltt + "," + stime
+ "," + ftime + "," + a + "," + b + "," + c + "," + d + "," + e + "," + f + "," + g + "," + fu + "," + h + "," + h1 + "," + condFrontXini + "," +
condFrontXFin + "," + condFrontYini + "," + condFrontYFin + "," + condFrontXiniflux + "," + condFrontXFinflux + "," + condFrontYiniflux + ","
+ condFrontYFinflux + ");");
 } else if (nnel == 3 && deltt != 0 && (h.equals("1") || h1.equals("1"))) {
 m.eval("cd " + dirFuncMatlab + "\\MetodoFem3NodosConTI");
 m.eval("result = fem3genTI(" + xIni + "," + xFin + "," + yIni + "," + yFin + "," + nel + "," + nnel + "," + ndof + "," + nnode + "," + deltt + "," +
stime + "," + ftime + "," + a + "," + b + "," + c + "," + d + "," + e + "," + f + "," + g + "," + fu + "," + h + "," + h1 + "," + condFrontXini + "," +
condFrontXFin + "," + condFrontYini + "," + condFrontYFin + "," + condFrontXiniflux + "," + condFrontXFinflux + "," + condFrontYiniflux + ","
+ condFrontYFinflux + ");");
 } else if (nnel == 4 && deltt == 0 && h.equals("0") && h1.equals("0")) {
 m.eval("cd " + dirFuncMatlab + "\\MetodoFem4Nodos");
 m.eval("result = fem4gen(" + xIni + "," + xFin + "," + yIni + "," + yFin + "," + nel + "," + nnel + "," + ndof + "," + nnode + "," + deltt + "," + stime
+ "," + ftime + "," + a + "," + b + "," + c + "," + d + "," + e + "," + f + "," + g + "," + fu + "," + h + "," + h1 + "," + condFrontXini + "," +
condFrontXFin + "," + condFrontYini + "," + condFrontYFin + "," + condFrontXiniflux + "," + condFrontXFinflux + "," + condFrontYiniflux + ","
+ condFrontYFinflux + ");");
 } else if (nnel == 4 && deltt != 0 && (h.equals("1") || h1.equals("1"))) {
 m.eval("cd " + dirFuncMatlab + "\\MetodoFem4NodosConTI");
 m.eval("result = fem4genTI(" + xIni + "," + xFin + "," + yIni + "," + yFin + "," + nel + "," + nnel + "," + ndof + "," + nnode + "," + deltt + "," +
stime + "," + ftime + "," + a + "," + b + "," + c + "," + d + "," + e + "," + f + "," + g + "," + fu + "," + h + "," + h1 + "," + condFrontXini + "," +
condFrontXFin + "," + condFrontYini + "," + condFrontYFin + "," + condFrontXiniflux + "," + condFrontXFinflux + "," + condFrontYiniflux + ","
+ condFrontYFinflux + ");");
 } else {
 System.out.println("Parametros incorrectos");
 }
//Salvar los resultados que devuelva matlab
 ress = m.save("result");
 m.close();
 System.out.println("OK");
 } catch (Exception ex) {
 try {
 ex = new Exception("Parametros de simulacion incorrectos");
 System.out.println(ex.getMessage());
```

```
 m.close();
 } catch (IOException ex1) {
 Logger.getLogger(Main.class.getName()).log(Level.SEVERE, null, ex1);
 System.exit(0);
 }
 System.exit(0);
 }
//Retornar resultados.
 return ress;
 }
```

Tabla 5: Método ejecutar de la clase MatlabControler

Glosario de Términos

Discretización: Transformar un cuerpo de naturaleza continua en un modelo discreto aproximado.

Aplicaciones de Cliente Enriquecido: Término medio entre el cliente liviano y el cliente pesado. El objetivo del cliente enriquecido consiste en proporcionar una interfaz gráfica, escrita con una sintaxis basada en XML, que proporciona funcionalidades similares a las del cliente pesado (arrastrar y soltar, pestañas, ventanas múltiples, menús desplegables)