Universidad de las Ciencias Informáticas

Facultad 6: Bioinformática

TÍTULO: “Biblioteca de clases para la Estimación de Parámetros y Análisis de Sensibilidad en Sistemas de Ecuaciones Diferenciales Ordinarias.”

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

Autores:

Niuvys Rodríguez Hernández

Fernando Henríquez Amaya

Tutores:

Ing. Edel Moreno Lemus

Ing. Vilmavis La Rosa Sordo

Ing. Ernesto Contreras Torres

Consultantes:

Drc. Matemáticas Abel E. Fernández Infante.

MsC. Tonysé de la Rosa Martín

La Habana

“Año del 55 de la Revolución”

"Seamos realistas y hagamos lo imposible. "

Ernesto Guevara de la Serna.

DECLARACIÓN DE AUTORÍA

Declaramos ser los autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales sobre la misma.

Para que así conste firmamos la presente a los _____ días del mes de _____ del _____.

Niuvys Rodríguez Hernández

_____

Firma del autor

Fernando Henríquez Amaya

_____

Firma del autor

Ing. Vilmavis La Rosa Sordo

_____

Firma del Tutor

Ing. Edel Moreno Lemus

_____

Firma del Tutor

Ing. Ernesto Contreras Torres

_____

Firma del Tutor

DATOS DE CONTACTO

Tutores:

Ing. Vilmavis La Rosa Sordo.

Universidad de las Ciencias Informáticas, Carretera San Antonio km 21/2 La Lisa. Habana, Cuba.

Email: vlarosa@uci.cu

Ing. Edel Moreno Lemus

Universidad de las Ciencias Informáticas, Carretera San Antonio km 21/2 La Lisa. Habana, Cuba.

Email: edel.moreno@gmail.com

Ing. Ernesto Contreras Torres

Universidad de las Ciencias Informáticas, Carretera San Antonio km 21/2 La Lisa. Habana, Cuba.

Email: econtreras@uci.cu

AGRADECIMIENTOS

Hasta la más sencilla de las metas es imposible de alcanzar sin el apoyo brindado a quien lucha por ella, y hacerse profesional que sin dudas no es sencillo, no nos hubiese sido posible sin la ayuda de un grupo enorme de personas, por eso queremos agradecer a todos los que de alguna forma contribuyeran a que hoy lleguemos al final de la más importante de nuestras metas trazada como estudiantes y que nos ha ayudado a crecer en todas las esferas de la vida, pues como dice la frase:"El hombre no es más que lo que la educación hace de él".

Queremos agradecer de forma especial a:

A mi mamá por la fuerza y el apoyo incondicional en cada uno de mis pasos. Por no dejar pasar un día sin escuchar su voz, y aunque nunca se lo dije no hubo un el día en que no lo necesitara.

A mi papá por ser mi guía en todo momento, por el amor y la confianza depositada en mí.

A mi abuelita Cary por la ternura, los consejos y la compresión.

A toda mi familia y vecinos por la preocupación y el apoyo durante estos 5 años, en especial a mi tata Eddy, mi tía Barby y a mi Fela por cuidar de mi mamá en mi ausencia.

A mis amistades en la UCI, lo que están y los llevaré en mi corazón por siempre, en especial: a Ángel, Yunior, Arián, Erick, Omar, Randy, a mi hermanita Diana Malangón, mi mamá Diana y por último y no menos importante al insoportable de Bazán.

A Fer, mi dúo de tesis, por arrastrarme a esta aventura, por la paciencia, la confianza y el cariño.

A mis tutores Vilmavis, Edel y Ernesto por los consejos, la sabiduría y la dedicación que han tenido todo este tiempo que ha hecho posible el logro de este trabajo.

A esa persona tan especial que es Tonysé por la influencia de todos estos años, por la ayuda y los regaños.

A la familia de Fernando por el apoyo y la preocupación, en especial a su prima Sonia y a su tío Abel.

A todos las personas que compartieron conmigo durante los buenos y no tan buenos momentos de la universidad.

A todos, muchas gracias....
Niuvys Rodríguez Hernández

A mis padres, mis hermanos y demás familiares allegados quienes han sido para mí mis primeros educadores quienes han sabido guiarme por la senda correcta para llegar hoy hasta aquí.

A todas las personas que contribuyeron a que este trabajo se realizara, principalmente a mis tutores Edel, Vilmavis y Ernesto, mi prima Sonia y por último a mi tío y consultante Abel.

A Niuvys, por su dedicación y entrega, por su grado de responsabilidad, por ser una amiga y compañera de tesis ejemplar, muchas gracias.

A todas las personas que me tendieron la mano en momentos cruciales.

A todos mis amigos y compañeros de grupos con los que durante estos años he compartido alegrías y fracasos y que tendrán siempre un lugar reservado en mi corazón.

Muchas gracias a todos....
Fernando Henríquez Amaya

DEDICATORIA

Muchas personas caminan y caminarán dentro y fuera de tu vida pero solo las personas especiales dejarán una huella en tu corazón. A ellos va dedicado este Trabajo de Diploma.

A mi mamá, mi tesoro, mi estrella, mi ejemplo porque todo lo que soy se lo debo a ella y de volver a nacer desearía la dicha de ser su hija.

A mi papá, mi gran amor, mi más fiel caballero quien ha sido partícipe de cada uno de mis pasos, impulsándome en cada revés a ser mejor y dar más de mí.

A mi abuela Cary quien siempre ha estado por mí y para mí contra viento y marea.

A mis hermanas quienes me impulsan a ser cada día mejor, deseando ser un ejemplo para ellas.

A mi familia por tantos años de espera, de apoyo incondicional y desmedido.

Niuvys Rodríguez Hernández

Quiero dedicar mi Trabajo de Diploma primeramente a mi familia, que siempre me ha apoyado y respetado mis decisiones, es para mí mucho más que un honor poderlos llamar "mi familia" en especial a:

A mi padre: una de las personas más importante en mi vida, que aunque puede que tenga miles de defectos, como es natural -todos somos humanos-, para mi es el más grande ejemplo y mi mayor inspiración.

A mi Madre, quien me ha dado el regalo más grande que pudieran obsequiar: "La vida" y que por razones del destino, ya no se encuentra físicamente junto a mí, espero que donde quiera que esté se sienta orgullosa.

A mis hermanas, Mayeline y María Elena que más que hermanas han sido mis segundas madres, a quienes les debo mi juventud y mi vida toda.

A mi hermano, de quien he admirado siempre su genialidad e inteligencia, que por razones de la vida nunca pudo convertirse en un profesional, y que por tal motivo he dedicado tantas horas de estudio, espero llegar algún día parecerme al profesional que él podría haber sido.

A mi tía Delfina, por la confianza que siempre tuvo en mí, por todo su apoyo, por hacerme sentir como uno más de sus hijos durante estos 5 años.

A mis primos, en especial a Abelito, mi otro hermano.

A mi tío Abel y mi prima Sonia quienes han sido de gran ayuda en este trabajo, quienes me han enseñado mucho, sobre todo cuanto me falta por aprender.

A todos mis amigos, en especial a los que de una forma u otra aportaron su granito de arena para poder hacer un sueño realidad, mencionar a algunos sería cometer una injusticia. A ellos, quienes han sido capaces de soportar mi carácter y mis caprichos, quienes han sabido aconsejarme, alegrarme en el momento preciso, va dedicado mi esfuerzo. Gracias por su apoyo y comprensión.

Fernando Henríquez Amaya

RESUMEN

La determinación de parámetros o condiciones con los cuales se obtendrían mejores resultados para un problema específico en el campo de la Bioinformática, en ocasiones resulta prácticamente imposible si se cuenta con gran volumen de datos. Constantemente se crean algoritmos y aplicaciones que ayudan a la modelación, simulación y análisis de los sistemas biológicos. Producto del trabajo en conjunto entre el Departamento de Bioinformática, perteneciente al Centro de Tecnologías y Gestión de Datos de la Universidad de las Ciencias Informáticas y el Centro de Inmunología Molecular surge el Software para la Simulación y Análisis de Sistemas Biológicos. Conformado por un conjunto de módulos de desempeño independiente. Con la presente investigación se pretende desarrollar una biblioteca de clases para la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales (SED) ordinarias, ya que los módulos existentes no son capaces de estimar y analizar la sensibilidad de los parámetros de estos sistemas en un mismo módulo. Para el logro del objetivo trazado se hizo indispensable el estudio de diferentes sistemas existentes para la Estimación de Parámetros, definiéndose además la metodología, tecnologías y herramientas a emplear. Se obtiene como resultado una biblioteca de clases capaz de realizar la Estimación de Parámetros y el Análisis de Sensibilidad en SED ordinarias, empleando como técnicas de optimización: Algoritmos Genéticos, Optimización por Enjambre de Partículas y Búsqueda Directa.

PALABRAS CLAVE:

Algoritmos genéticos, análisis de sensibilidad, búsqueda directa, estimación de parámetros y optimización por enjambre de partículas.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

ÍNDICE DE ECUACIONES

ÍNDICE DE GRÁFICAS

INTRODUCCIÓN

La Biología de Sistemas (BS) es un área de investigación científica cuyo objetivo es el estudio de procesos biológicos usando un enfoque sistémico. Comenzó a desarrollarse en los años sesenta del siglo XX, aunque su institucionalización académica no se produjo hasta el año 2000. La BS constituye un área interdisciplinaria capaz de integrar a expertos procedentes de áreas como la Biología, la Informática, la Física y las Matemáticas.

A diferencia de los métodos tradicionales de estudio empleados por los biólogos: métodos científicos, centrados en la confirmación o refutación de hipótesis a través de resultados experimentales, la BS utiliza fundamentalmente la modelación. Esta técnica surge del uso de modelos matemáticos que describen el comportamiento de los sistemas biológicos.

Un modelo matemático (MM) es una descripción matemática de un fenómeno del mundo real, como el crecimiento de poblaciones, el funcionamiento de las neuronas y la dinámica intracelular, por citar algunos ejemplos de su aplicación en los procesos biológicos. La finalidad de estos modelos es representar de manera comprensible el comportamiento de dichos procesos. Algunos de los modelos más usados en la actualidad son las Redes Neuronales y los Sistemas de Ecuaciones Diferenciales (SED).

En la actualidad, para el estudio de sistemas biológicos se requieren variables o parámetros para comprenderlos y describirlos. En la construcción de estos modelos una de las principales dificultades que se enfrenta es la falta de valores de parámetros precisos. La ausencia de estos valores se asocia a menudo con un alto grado de incertidumbre o se calculan dentro de rangos grandes. Esto puede provocar que los parámetros medidos no reflejen la “verdadera” actividad de dicho sistema. Muchos de estos parámetros no se pueden obtener de forma experimental, por lo que se hace necesario estimarlos con la ayuda de herramientas informáticas.

En Cuba, hace ya algunos años, se llevan a cabo investigaciones en aras de contribuir al desarrollo del estudio de los sistemas biológicos, evidenciado en los recientes trabajos desarrollados entre el Departamento de Bioinformática, perteneciente al Centro de Tecnologías y Gestión de Datos (DATEC) de la Facultad 6 de la Universidad de las Ciencias Informáticas (UCI) y el Centro de Inmunología Molecular (CIM). Producto de este trabajo en conjunto surge el Software para la Simulación y Análisis de Sistemas Biológicos (BioSyS).

BioSyS, es un software en su versión 1.1 liberado, cuyo objetivo fundamental es permitir que los

investigadores realicen simulaciones a sistemas biológicos que estén descritos por ecuaciones diferenciales ordinarias. Las simulaciones se pueden hacer de forma local o distribuida. Además, el sistema almacena la información en una base de datos para el posterior meta-análisis del conjunto almacenado.

El software abarca un conjunto de funcionalidades distintas en temáticas específicas, agrupadas en módulos identificados e independientes. A continuación se enuncian algunos: el Módulo Editor de Ecuaciones realiza la modelación matemática del sistema. El Módulo Simulación se encarga de resolver el SED y el Módulo de Análisis estudia los resultados obtenidos. Por otra parte, el Módulo Estimación se encarga de calcular los valores de los parámetros desconocidos y el Módulo de Análisis de Sensibilidad analiza cuánto influye la variación de los parámetros en la solución del sistema.

Sin embargo, BioSyS 1.1 no permite realizar la Estimación de Parámetros y el Análisis de Sensibilidad a SED ordinarias en un mismo módulo, lo cual contribuye a un aumento en el tiempo de cómputo en la estimación y que el trabajo del investigador sea más engorroso. Por otra parte, los algoritmos y técnicas empleadas para la Estimación de Parámetros y el Análisis de Sensibilidad respectivamente, necesitan ser revisados y optimizados, así como también la incorporación de otros algoritmos favorecería en gran medida los resultados de las investigaciones que se efectúan.

Teniendo en cuenta lo antes expuesto, se plantea como **problema de la investigación**: ¿Cómo estandarizar la Estimación de Parámetros y el Análisis de Sensibilidad en un sistema de ecuaciones diferenciales ordinarias a partir de un conjunto de datos de laboratorio?

Por cuanto, el **objeto de estudio** de la presente investigación se enmarca en: La Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias. En aras de dar solución al problema planteado se propone como **objetivo general**: Desarrollar una biblioteca de clases que permita estimar los parámetros y realizar el análisis de sensibilidad en sistemas de ecuaciones diferenciales ordinarias.

Para el cumplimiento del objetivo general se delinearon los siguientes **objetivos específicos**:

- Fundamentar la selección de la metodología, herramientas y tecnologías a utilizar para la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias.
- Realizar el diseño de una biblioteca de clases para la Estimación de Parámetros y Análisis de

Sensibilidad en sistemas de ecuaciones diferenciales ordinarias.

- Implementar una biblioteca de clases que contenga las funcionalidades de los algoritmos seleccionados.
- Probar los resultados obtenidos.

Trazado el objetivo general de la investigación, se delimita como **campo de acción**: Herramientas informáticas para la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias. Para dar solución al problema planteado y cumplimiento al objetivo general trazado, se planearon las siguientes **tareas de la investigación**:

- Análisis de los algoritmos de optimización, de algoritmos para la Estimación de Parámetros y técnicas para el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias.
- Análisis de las tecnologías y tendencias actuales para la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias.
- Definición de las características de la biblioteca de clases a implementar.
- Selección de los algoritmos para la Estimación de Parámetros en sistemas de ecuaciones diferenciales ordinarias.
- Diseño de técnicas para el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias.
- Implementación de los algoritmos seleccionados para la Estimación de Parámetros.
- Implementación de las técnicas seleccionadas para el Análisis de Sensibilidad.
- Realización de pruebas de caja blanca a los algoritmos implementados.
- Realización de pruebas de caja negra a la biblioteca de clases desarrollada.

En la presente investigación se utilizaron métodos científicos de investigación como son: los métodos teóricos y los métodos empíricos, de los cuales se fundamenta a continuación su selección.

Los métodos teóricos permiten estudiar las características del objeto de la investigación que no son observables directamente (1). Los métodos teóricos se dividen en: históricos y lógicos. Para el desarrollo del presente trabajo de diploma se seleccionan los métodos lógicos, y dentro de esta clasificación el método sistémico partiendo de que el mismo propicia el estudio de las herramientas informáticas para realizar la Estimación de Parámetros, así como valorar las deficiencias de estas en relación al problema específico tratado.

Los métodos empíricos describen las características fenomenológicas del objeto. Aunque existen diversas opiniones, la mayoría de los autores concuerdan que los métodos empíricos generales son: la observación, la medición y la experimentación (1). Optándose, para el problema planteado, por el uso de la medición, ya que es aquel procedimiento que se realiza para obtener información numérica acerca de una propiedad o cualidad del objeto, donde se comparan magnitudes medibles y conocidas de este.

Estructura del Documento:

En aras de cumplir con los elementos antes expuestos el presente trabajo ha sido estructurado en tres capítulos, quedando de la siguiente manera:

Capítulo 1: Fundamentos Teóricos: En este capítulo se realiza un estudio de la Estimación de Parámetros y Análisis de Sensibilidad, además se profundiza en los algoritmos de optimización, haciendo particular énfasis en los Algoritmos Genéticos, Búsqueda Directa y PSO. Se analizan algunas soluciones informáticas existentes para la Estimación de Parámetros en sistemas de ecuaciones diferenciales ordinarias. Se exponen la metodología de desarrollo, lenguaje y herramienta de modelado, lenguaje de programación y entorno de desarrollo seleccionados para dar solución al problema planteado.

Capítulo 2: Descripción y Diseño: En este capítulo se describen las funcionalidades y características de la biblioteca de clases a desarrollar, las cuales responden a la especificación de los requisitos funcionales y no funcionales que se realizó. Se describe la arquitectura a través de una Vista Lógica evidenciada en el diagrama de clases. Finalmente, se presentan los patrones de diseño.

Capítulo 3: Implementación y Prueba: En este capítulo se realiza la implementación de los algoritmos y técnicas seleccionados para la Estimación de Parámetros y el Análisis de Sensibilidad respectivamente. Se expone parte del código fuente de los principales métodos y se realizan las pruebas necesarias para validar la biblioteca de clases implementada.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

Introducción

En el presente capítulo se realiza un análisis de la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias. Se exponen además, las tendencias, técnicas, tecnologías, herramientas, metodología de desarrollo de software y el lenguaje de programación seleccionados para dar solución al problema planteado.

1.1. Biología de Sistemas

Con el transcurso de los años y partiendo de los avances de la Biología Molecular, las nuevas técnicas han propiciado el acceso a miles de datos. Algoritmos novedosos y eficientes, y técnicas estadísticas para la gestión y análisis de datos biológicos han cambiado la actitud de muchos científicos sobre cómo solucionar algunos de sus problemas, permitiendo el nacimiento de una nueva disciplina: la Biología de Sistemas.

“La Biología de Sistemas consiste en el estudio de un organismo o sistema biológico, visto como un sistema integrado e interrelacionado de genes, proteínas y reacciones bioquímicas que dan lugar a procesos biológicos. En lugar de analizar los componentes individuales de un organismo, los biólogos de sistemas se centran en todos los componentes y sus interacciones como parte de un único sistema, que serán las responsables de la biología del organismo” (2).

La BS se caracteriza por el estudio de los sistemas biológicos de una forma global. Maneja una gran colección de datos procedentes de estudios experimentales. Basado en esto, propone modelos matemáticos que pueden explicar algunos fenómenos biológicos estudiados, propiciando soluciones matemáticas que permiten predecir los procesos biológicos. La comprobación de estos modelos matemáticos se realiza por medio de la comparación entre simulaciones numéricas y los datos experimentales.

La realización de comparaciones produce gran cantidad de datos de tipo biológico que han de ser transformados en conocimiento, lo que se consigue mediante el empleo de diferentes tecnologías computacionales. A continuación se realizará una revisión de estas tecnologías, cuyo desarrollo es clave para la evolución de la BS.

1.2. Tecnologías computacionales fundamentales en la Biología de Sistemas

Las principales tecnologías empleadas en la BS se dividen en dos categorías: Técnicas Experimentales y Técnicas Computacionales. Aunque es posible realizar una separación entre estas técnicas, en realidad no existe una disociación clara entre ellas. Los datos procedentes de experimentos a gran escala requieren de una interpretación y evaluación intensiva por medio de técnicas computacionales.

Las Técnicas Computacionales se basan en el desarrollo de algoritmos mediante los cuales la información obtenida por las técnicas experimentales es procesada y transformada en conocimiento. Se identifican esencialmente entre estas técnicas: la modelación matemática, la cual proporciona una vía para predecir el comportamiento de sistemas biológicos complejos, la simulación de procesos biológicos y por último, la Estimación de Parámetros**,** técnica determinante a su vez en la modelación (2).

1.2.1. Modelación matemática

Tradicionalmente, el modelado formal de sistemas ha sido a través de un modelo matemático. Un modelo matemático permite representar un problema médico o biológico de una manera objetiva en la que se definen una serie de relaciones matemáticas entre las mediciones cuantitativas (del problema) y sus propiedades. Es un tipo de modelo científico que utiliza algún formulismo matemático para expresar relaciones, proposiciones sustantivas de hechos, variables, parámetros, entidades y relaciones entre variables y/o entidades u operaciones (3).

Los sistemas biológicos son sistemas complejos porque cambian en el tiempo y están constituidos por diferentes partes interconectadas entre sí, por ese motivo son susceptibles al estudio mediante el uso de estas herramientas matemáticas. Ello justifica el planteamiento de un modelo matemático que permita describir y desarrollar predicciones sobre el comportamiento del sistema. Algunos de los modelos más usados en la actualidad son los Redes Neuronales y los Modelos de Markov (2). Ejemplo de otros modelos usados comúnmente son: los Sistemas de Ecuaciones Diferenciales (SED).

Según la definición matemática, una ecuación diferencial es toda ecuación que contiene las derivadas o las diferenciales de una o más variables dependientes con respecto a una o más variables independientes (4). Por tanto, un SED es un conjunto de funciones diferenciales que satisfacen todas y cada una de las ecuaciones del sistema.

En la modelación de problemas biológicos han sido los SED los de mayor utilización. Además de permitir representar, describir, analizar y estimar la evolución de un sistema en el tiempo, dadas las condiciones

límite para dicho sistema, brindan facilidades de modelación y de simulación (4). Partiendo de lo expuesto, se le concede gran importancia a la utilización de los SED, como modelo matemático en la biblioteca de clases a implementar.

La modelación matemática es un proceso clave para el logro de los objetivos de las investigaciones en el campo de la BS. Los científicos, previamente a la prueba de fármacos en animales o humanos, desarrollan modelos computacionales de fármacos candidatos y, a continuación, ejecutan simulaciones basadas en dichos modelos, ello permite rechazar aquellos que tienen pocas posibilidades de éxito y optimizar los más prometedores.

1.2.2. Simulación

La simulación por computadora es una técnica numérica para conducir experimentos a través de un computador. Los experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos períodos de tiempo. Las simulaciones permiten a los investigadores reproducir en el ordenador de forma fehaciente el comportamiento del sistema biológico en estudio.

Comprender el sistema biológico y evaluar nuevas estrategias dentro de los límites impuestos -por un cierto criterio o un conjunto de ellos- es fundamental para el funcionamiento correcto del sistema. La simulación permite determinar qué componentes del modelo tienen el mayor efecto en los resultados y por tanto cuáles son los más importantes bajo un conjunto específico de condiciones (3). Posibilita la optimización del proceso investigativo y del tiempo de ejecución, los científicos descartan infructuosas vías de investigación, concentrándose en experimentos más ajustados a los candidatos más prometedores.

Si bien el modelado y la simulación aún no se han adoptado universalmente, otras técnicas ingenieriles están siendo ampliamente utilizadas en la BS y están mejorando incluso el desempeño de estas. Entre ellas se destaca la Estimación de Parámetros, necesaria para el uso de modelos matemáticos en la predicción de fenómenos biológicos.

1.2.3. Estimación de Parámetros en la Biología de Sistemas

En la actualidad existen pocas investigaciones de laboratorio que han utilizado con pleno éxito técnicas computacionales como el modelado y el cálculo. Esto generalmente se debe a que los investigadores carecen de las herramientas necesarias que faciliten dichos procesos. Por otra parte, la mayoría de sistemas biológicos son muy complejos y se necesita una cantidad considerable de ingeniería inversa

(realizar experimentos y simulaciones) para reunir suficiente información y conocimiento que permita modelarlos.

La complejidad de los sistemas biológicos hace necesario concentrar los esfuerzos en modelos que describen partes definidas del sistema completo. Es por tanto esencial la utilización de metodologías que identifiquen los elementos clave de un sistema. Además, estimar los errores inevitables que surgen como consecuencia de las simplificaciones y asunciones derivadas del uso de modelos matemáticos es un aspecto imprescindible (2).

La estandarización de experimentos in vitro e in vivo y sus datos es a menudo inconsistente, inaccesible, incompleto, o desestructurada. Esto conlleva frecuentemente al diseño de modelos independientemente de los datos disponibles, con la esperanza de que aparezcan parámetros adecuados para el modelo, o que los parámetros puedan ajustarse razonablemente al modelo para conseguir que funcione.

Los investigadores utilizan la Estimación de Parámetros para calibrar la respuesta de un modelo a los resultados observados en un sistema físico (3). En lugar de recurrir a conjeturas para ajustar los parámetros del modelo y las condiciones iniciales, ellos automáticamente calculan estos valores utilizando datos recogidos a través de ensayos o experimentos. De igual forma, los biólogos de sistemas estiman los parámetros desconocidos necesarios para ajustar un modelo al comportamiento de un sistema biológico.

La Estimación de Parámetros consiste en determinar los parámetros desconocidos dentro del modelo matemático de un sistema x, del cual se tienen además, ciertas mediciones experimentales. Es decir, se tiene el modelo matemático del sistema x y una serie de mediciones experimentales, pero se desconoce el valor de ciertos parámetros, entonces lo que se pretende es encontrar qué valores de dichos parámetros hacen que el modelo matemático converja a las mediciones experimentales.

Si bien algunos de los valores de parámetros pueden estar disponibles para los investigadores, varios desconocen su valor exacto o aproximado debido a que los experimentos necesarios para determinarlos directamente son demasiado complejos o costosos. Es aquí donde el biólogo o investigador, partiendo de los resultados de las pruebas experimentales, puede estimar los valores de esos parámetros permitiendo que el modelo se ajuste con precisión al sistema biológico real. Por ello la Estimación de Parámetros es una capacidad fundamental en la BS. Conocer el impacto que tendría en el sistema que se modela las variaciones de uno o varios parámetros reporta una optimización de tiempo y una mejora en los resultados que se obtienen.

Análisis de Sensibilidad

El Análisis de Sensibilidad es un caso especial de la Estimación de Parámetros, es una técnica que se emplea para conocer cuánto influye la variabilidad de los valores de un parámetro desconocido en la salida del sistema. (5) Como resultado de este proceso, se sabrá si dicho parámetro es sensible o no. Esta técnica es ampliamente utilizada para evaluar el impacto que tienen los datos de entrada o las restricciones especificadas a un modelo definido en el resultado final o en las variables de salida de dicho modelo.

Generalmente, el Análisis de Sensibilidad se emplea para simplificar los modelos matemáticos partiendo de la exploración del impacto de la variación de entradas y escenarios. Además, permite identificar las variables más críticas e investigar las fortalezas de las predicciones de un modelo. Existen específicamente dos técnicas para realizar el Análisis de Sensibilidad, las cuales se enuncian a continuación.

Técnicas para realizar Análisis de Sensibilidad (5)**:**

Al considerar las variaciones de los parámetros en la evaluación de un modelo, se puede actuar de dos maneras:

- Variar un parámetro manteniendo constante los restantes.
- Variaciones combinadas de todos ellos.

El procedimiento más completo es indudablemente la variación combinada; aunque presenta el inconveniente que al poderse efectuar una multitud de posibles combinaciones no es posible determinar cuál de los parámetros fue el que produjo el cambio o si fue la variación de todos ellos. Sin embargo, para modelos más simples se puede considerar el efecto individual de las variables más importantes. Asimismo, desde un punto de vista operativo, el primer método puede servir para identificar las variables críticas o más influyentes, y una vez determinadas se puede pasar a realizar variaciones combinadas de estas técnicas. En la presente investigación se emplea la primera técnica enunciada.

Para realizar el Análisis de Sensibilidad primeramente se definen las condiciones iniciales y los parámetros fijos. Se establece el rango de variación de las condiciones iniciales o parámetros de los que se quiere estudiar la sensibilidad del sistema ante sus cambios. Luego, se realiza la simulación de estos parámetros, variando su valor dentro de rango establecido. Las simulaciones obtenidas para cada parámetro variado son comparadas, empleando una función objetivo (FO), con una simulación patrón realizada o con los datos experimentales.

La representación matemática de la función objetivo para este caso sería:

$$f_{obj}(k) = \sum_{i=1}^{q} \left(f_{obs}(i) - f_{per}(i,k)\right)^2,$$

Figura 1: Función Objetivo para Análisis de Sensibilidad (6)

La FO para el Análisis de Sensibilidad se define como una suma de errores cuadráticos entre los valores observados y los valores de la simulación realizada, donde:

- $Fobj\ (k)$: es la función objetivo que describe cómo la salida del sistema (modelo) se desvía de los datos observados ante la variación de los parámetros.
- q : es el número de puntos de muestreo de tiempo.
- $Fobs\ (i)$: denota un valor de salida observado en un tiempo i de muestreo.
- $Fper\ (i,k)$: denota el resultado de la simulación de acuerdo con la variación del parámetro k en un tiempo i (6).

Si el valor arrojado por la FO es menor que el error de corte definido, significa que el resultado no difiere del experimental y si está por encima significa que si difiere. Los resultados obtenidos al finalizar todas las iteraciones se procesan empleando una fórmula probabilística. Esta fórmula analiza si la diferencia entre la cantidad de resultados que difieren y que no difieren resulta significativo para el sistema que se analiza. Si lo es, se concluye que el sistema es inestable ante la variación de dicho parámetro, si no, es estable.

La Modelación, la Simulación y la Estimación de Parámetros de sistemas biológicos son técnicas que van de la mano en la BS. A partir de los resultados obtenidos en la Estimación de Parámetros se realizan simulaciones más cercanas a los objetivos de la investigación, lo cual posibilita ajustar con mayor precisión un modelo propuesto al comportamiento del sistema biológico que se estudia.

En la BS es necesaria la optimización de las funciones objetivo para saber los parámetros que permiten lograr los mejores resultados posibles para un experimento. La Estimación de Parámetros se realiza cuando se tiene un conjunto de datos experimentales y se quiere determinar los parámetros del SED ordinarias que mejor se ajustan a dichos datos, siendo resuelto entonces como un problema de optimización continuo.

1.3. Optimización

En términos informales, optimizar significa algo más que mejorar; sin embargo, en el contexto científico la optimización es el proceso de tratar de encontrar la mejor solución posible para un determinado problema (7). En un problema de optimización existen diferentes soluciones, el objetivo es encontrar la mejor. Un problema de optimización consiste entonces en tomar una decisión óptima para maximizar o minimizar un criterio determinado. El término de optimización se aplica entonces a aquellos casos en que se maximiza o minimiza una función objetivo.

1.4. Algoritmos de Optimización

A lo largo de la historia de la Informática se han desarrollado múltiples métodos para tratar de resolver los problemas de optimización. Las técnicas existentes se clasifican en exactas (enumerativas o exhaustivas) y en aproximadas. Las técnicas exactas garantizan encontrar la solución óptima para cualquier instancia de cualquier problema en un tiempo acotado (8). El inconveniente de este método es que el tiempo necesario para hallar la solución, aunque acotado, crece exponencialmente con el tamaño de la población, demorando inclusive años para dar una solución.

Los algoritmos aproximados o no exactos sacrifican la garantía de encontrar el óptimo a cambio de encontrar una “buena” solución en un tiempo “razonable”. Dentro de los algoritmos no exactos se pueden encontrar tres tipos: los heurísticos constructivos (también llamados voraces), los métodos de búsqueda local (o métodos de seguimiento del gradiente) y las metaheurísticas (8), siendo estas últimas motivo de estudio del presente trabajo de diploma.

Una metaheurística es una estrategia de alto nivel que usa diferentes métodos para explorar el espacio de búsqueda. Su objetivo es la exploración eficiente del espacio de búsqueda para encontrar soluciones (casi) óptimas. Utilizan funciones de *fitness* (en español, aptitud) para cuantificar el grado de adecuación de una determinada solución (9). Hay diferentes formas de clasificar y describir las técnicas metaheurísticas: basadas en trayectoria y basadas en población (10).

Las metaheurísticas basadas en trayectoria parten de un punto y mediante la exploración de los puntos cercanos van actualizando la solución actual, formando una trayectoria. Normalmente, la búsqueda termina cuando se alcanza un número máximo predefinido de iteraciones, se encuentra una solución con una calidad aceptable, o se detecta un estancamiento del proceso (8). Ejemplo de estas metaheurísticas son: el Enfriamiento Simulado o Simulated Annealing (SA), la Búsqueda Tabú o Tabu Search (TS), la Búsqueda Local Iterada o Iterated Local Search (ILS), entre otras.

Los métodos basados en población se caracterizan por trabajar con un conjunto de soluciones (población) en cada iteración, a diferencia de los métodos basados en trayectoria que únicamente utilizan un punto del espacio de búsqueda por iteración. El resultado final de estos algoritmos depende fuertemente de la forma en que es manipulada la población (8). Ejemplos de estos algoritmos son: los Algoritmos Evolutivos, dentro de los cuales se encuentran los Algoritmos Genéticos (AGs), los Algoritmos Basados en Cúmulos de Partículas o Particle Swarm Optimization (PSO), entre otros.

A continuación se realiza un estudio más profundo de algunos de estos algoritmos, destacando sus características y particularidades fundamentales.

1.4.1. Algoritmo de Búsqueda Directa

El algoritmo de Búsqueda Directa es un método para resolver problemas de optimización que no requieren ninguna información sobre el gradiente de la función objetivo. Básicamente, busca una serie de puntos alrededor del punto actual, buscando un punto donde el valor de la FO sea menor que el valor en el punto actual (11). Estos métodos son usados para resolver una variedad de problemas de optimización que no son muy adecuados para la optimización con algoritmos estándar.

A partir de 1971 comenzaron a aparecer los primeros resultados referentes a la convergencia de la Búsqueda Directa. Para una mejor organización, fueron divididos en tres categorías: Método Simplex, Búsqueda Multi-direccional, Búsqueda por Patrones o Patrón de Búsqueda. La presente investigación se centra en el Patrón de Búsqueda específicamente.

Patrón de Búsqueda.

El algoritmo Patrón de Búsqueda (PB) calcula una secuencia de puntos que se acerque al punto óptimo. En cada paso, el algoritmo busca un conjunto de puntos, llamado malla, en torno al punto actual, el punto calculado en el paso anterior del algoritmo. Construye una malla añadiendo al punto actual un escalar múltiplo de un conjunto fijo de vectores llamado patrón (11). Si el algoritmo encuentra un punto en la malla que devuelve un valor mejor de la función objetivo en el punto actual, el nuevo punto se convierte en el punto actual en el siguiente paso del algoritmo.

Patrones

Un patrón es una colección de vectores que utiliza el algoritmo para determinar cuáles puntos buscar en cada iteración (12). Por ejemplo, si hay dos variables independientes en el problema de optimización, el valor por defecto consiste en los siguientes vectores:

$v1 = [1\ 0], v2 = [0\ 1], v3 = [-1\ 0], v4 = [0\ -1].$

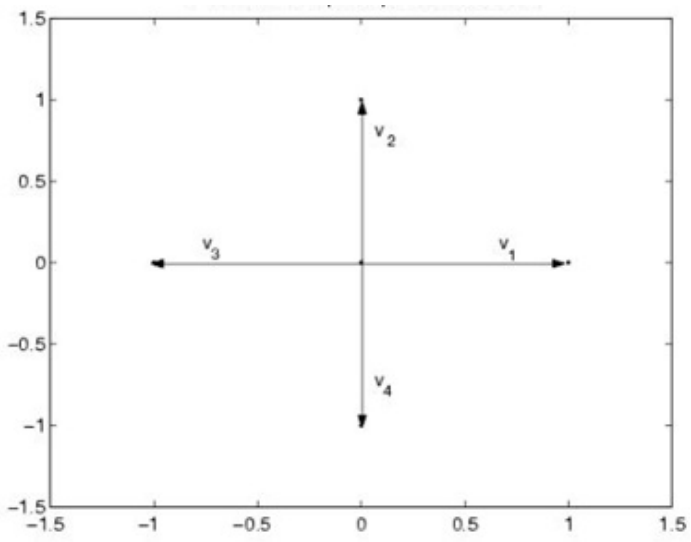


Figura 2: Patrón de Búsqueda

Mallas

En cada paso, el algoritmo PB busca un conjunto de puntos, llamado malla, para mejorar la FO. El algoritmo constituye la malla de la siguiente manera:

- **Primero:** Se multiplica el patrón de vectores por un escalar, que coincide con el tamaño de la malla.
- **Segundo:** Se agregan los vectores resultantes para el actual punto (el punto con el mejor valor de la función objetivo encontrada en el paso anterior).

Por ejemplo, supongamos que el punto actual es $[1{,}6, 3{,}4]$ y el tamaño de la malla 4. El algoritmo multiplica el patrón de vectores por 4 y los añade al actual punto para obtener la siguiente malla (12).

$[1{,}6\ ,3{,}4] + 4 * [1\ 0] = [5{,}6\ ,3{,}4]$

$[1{,}6\ ,3{,}4] + 4 * [0\ 1] = [1{,}6\ ,7{,}4]$

$[1{,}6\ ,3{,}4] + 4 * [-1\ 0] = [-2{,}4, 3{,}4]$

$[1{,}6\ ,3{,}4] + 4 * [0\ -1] = [1{,}6, -0{,}6]$

Plan del Patrón de Búsqueda

En cada paso, el algoritmo sondea los puntos en la actual malla para calcular los valores de la FO. Por defecto, el algoritmo detiene los sondeos de la malla tan pronto como encuentre un punto cuyo valor en la

FO es menor que el del punto actual. El sondeo es entonces llamado exitoso y el punto se convierte en el punto actual en la próxima iteración. Si el sondeo es cambiado a completo, el algoritmo calcula los valores de la función objetivo en todos los puntos de malla.

Después de un exitoso sondeo, el algoritmo multiplica la actual malla por 2, el valor por defecto del factor de expansión de la malla. Debido a que el tamaño de la malla inicial es 1, a la segunda iteración el tamaño de la malla es de 2. Si el algoritmo no puede encontrar un punto que mejore la función objetivo, el sondeo no tiene éxito y el punto actual sigue siendo el mismo en la próxima iteración. Después de un infructuoso sondeo, el algoritmo multiplica el actual tamaño de la malla por 0.5, el valor por defecto de factor de contracción de malla. El algoritmo realiza entonces el sondeo con un menor tamaño de la malla (11).

1.4.2. Optimización basado en Enjambre de Partículas

El Algoritmo Basado en Enjambres de Partículas (en inglés, Particle Swarm Optimization (PSO)) es una metaheurística evolutiva y de búsqueda, concebido como un algoritmo bio-inspirado, ya que se basa en el comportamiento social de sistemas naturales como las bandadas de pájaros o bancos de peces (13). Las soluciones, llamadas partículas, se “echan a volar" en el espacio de búsqueda, guiadas por la partícula que mejor solución ha encontrado hasta el momento y que hace de líder de la bandada (14).

El enjambre de partículas es un sistema multiagente, es decir, las partículas son agentes simples que se mueven por el espacio de búsqueda, guardan (y posiblemente comunican) la mejor solución que han encontrado condicionado por dos factores básicos, la memoria autobiográfica de la partícula o nostalgia y la influencia social de todo el enjambre (8). La simulación de este comportamiento da lugar a un método para resolver problemas de optimización.

En el modelo básico de PSO el enjambre está compuesto por un conjunto de partículas que sobrevuelan el espacio d-dimensional. En general, una partícula está compuesta por:

- Vector $p_i = (p_{i,1}, p_{i,2}, \dots, p_{i,d})$ que representa la posición (solución) actual de la partícula.
- Vector $v_i = (v_{i,1}, v_{i,2}, \dots v_{i,d})$ que representa la velocidad actual de la partícula.
- Vector $pbest_i = (pbest_{i,1}, pbest_{i,2}, \dots pbest_{i,d})$ que almacena la mejor posición (solución) encontrada por la partícula hasta ese momento.
- Valor de aptitud $bestFitness_i$ que almacena el valor asociado a la mejor solución encontrada por la partícula *i* hasta ese instante.

- Valor de aptitud $fitness_i$ que almacena el valor asociado a la solución actual encontrada por la partícula.

Donde:

- i: i-ésima partícula del enjambre.
- d: dimensiones del espacio de búsqueda.

El enjambre se inicializa generando las posiciones p_i de las partículas de forma aleatoria, regular o heurística, posteriormente se actualizan los valores $fitness_i$ y $bestFitness_i$ respectivamente. Las velocidades v_i se generan en el intervalo $[-v_{máx}, v_{máx}]$, donde $v_{máx}$ representa la velocidad máxima que podrá alcanzar una partícula (8).

Operador velocidad y parámetros del algoritmo PSO

En la formulación de PSO se define la velocidad de partícula como el único operador disponible para controlar la evolución de la optimización (15). El vector velocidad se actualiza en cada iteración teniendo en cuenta 3 componentes: la velocidad anterior (inercia), un componente cognitivo[1] y un componente social. El modelo matemático resultante y que constituye el núcleo conceptual del algoritmo PSO está representado por las siguientes ecuaciones (8):

$$v_i(k+1) = wv_i(k-1) + c_1 rand1\big(pBest_i - p_i\,(k-1)\big) + c_2 rand2\big(pgBest_i - p_i\,(k-1)\big) \quad (1)$$

$$p_i(k+1) = p_i(k) + v_i(k+1) \quad (2)$$

Figura 3: Ecuaciones del Modelo Matemático del algoritmo PSO.

Donde:

- $v_i\,(k)$: velocidad de la partícula *i* en la iteración *k.*
- *w:* factor inercia*.*
- c_1, c_2: coeficientes de aprendizaje que controlan la influencia de los factores cognitivo y social.
- $p_i\,(k)$: posición de la partícula *i* en la iteración *k.*
- $rand1, rand2$: números aleatorios uniformemente distribuidos en el intervalo [0,1] para emular el comportamiento estocástico y un tanto impredecible que exhibe la población del enjambre.
- $pBest_i$: mejor posición (solución) encontrada por la partícula *i* hasta el momento.
- $pgBest_i$: mejor posición (solución) del entorno de la partícula *i*$(lBest)$ o de todo el enjambre $(gBest)$ encontrada hasta el momento.

[1]También conocido como experiencia personal, memoria autobiográfica, etc.

La ecuación (1) de la Figura 3 representa la actualización del vector velocidad de la partícula *i* en la iteración *k*, el componente cognitivo lo representa el factor: $c_1\, rand1\big(pBest_i - p_i(k-1)\big)$ cuyo significado es la distancia entre la mejor posición encontrada por la partícula y su posición actual, es decir, que la decisión que tomará la partícula según la influencia de su propia experiencia.

El componente social lo representa: $c_2 rand2\big(pgBest_i - p_i(k-1)\big)$ que análogamente representa la distancia entre la posición de la mejor partícula de su vecindario y la posición actual de la partícula. Significa la decisión que tomará la partícula de acuerdo con la influencia que el entorno social ejerce sobre ella.

El factor inercia *w* controla el impacto de la velocidad anterior con la respecto a la actual, representa un equilibrio entre la capacidades de exploración y explotación del enjambre; valores elevados de *w* favorecerán la exploración del espacio de soluciones, mientras que valores más pequeños de *w* incrementarán la explotación de una región determinada del espacio. Con el objetivo de lograr este balance Shi y Eberhart proponen iniciar con w= 0.9 e ir disminuyendo linealmente hasta w=0.4 (16). El factor de inercia puede ser calculado como:

$$w = w_{max} - \frac{w_{max} - w_{min}}{iter_{max}}\ iter$$

Ecuación 1: Ecuación para el Factor de Inercia

Donde:

- w_{max}: peso inercial inicial.
- w_{min}: peso inercial final.
- $iter_{max}$: cantidad máxima de iteraciones del algoritmo.
- $iter$: iteración actual del algoritmo.

Existen en la literatura varias formas de calcular la velocidad de la partícula, una alternativa al modelo anteriormente propuesto es el denominado factor de constricción introducido por Clerc y Kennedy, el cual está regido por las ecuaciones (4) y (5) de la Figura 4:

$$v_i(k+1) = K[v_i(k-1) + c_1 rand1\big(pBest_i - p_i\,(k-1)\big) + c_2 rand2\big(pgBest_i - p_i\,(k-1)\big)] \quad (4)$$

$$K = \frac{2\delta}{\left|2-\varphi-\sqrt{\varphi^2-4\varphi}\right|}\text{, donde: } \delta \in [0,1],\ \varphi = c_1 + c_2, \varphi > 4 \quad (5)$$

Figura 4: Ecuaciones para el Factor de Constricción.

En este modelo puede obviarse el parámetro $v_{máx}$, aunque en la práctica se suele combinar esta estrategia con dicho parámetro (17).

La utilización de ambos modelos PSO: con factor de inercia y con factor de constricción se encuentra generalizada en la literatura, aunque el último ofrece mejores resultados (18). La elección de los parámetros del algoritmo depende del problema que se desea resolver. La configuración de estos parámetros se realiza en el fichero *PSOConfiguration.xml.*

1.4.3. Algoritmos Genéticos

Los Algoritmos Genéticos (AGs) son métodos adaptativos que pueden ser usados para resolver problemas de búsqueda y optimización, corresponden a la clase de métodos estocásticos de búsqueda (19). Estos algoritmos operan en una población de soluciones. La idea básica, inspirada en los procesos evolutivos en biología, es que el contenido genético de una población contiene potencialmente la solución, o una solución mejor, a un problema dado de adaptación.

En un Algoritmo Genético (AG), después de parametrizar el problema en una serie de variables, $(x_i, \ldots, x_n)$ se codifican en un cromosoma. Todos los operadores utilizados por un AG se aplicarán sobre estos cromosomas, o sobre poblaciones de ellos (20). Este algoritmo es independiente del contexto, lo cual lo hace un algoritmo robusto, por ser útil para cualquier problema, pero a la vez débil, pues no está especializado en ninguno.

En un AG, se denomina población al conjunto de soluciones factibles del problema a resolver; cromosoma o individuo es una estructura de datos que representa una de las posibles soluciones en la población y gen a cada elemento de la sucesión lineal que constituye el cromosoma (21). Los cromosomas son sometidos a un proceso de evolución que envuelve: evaluación, selección, recombinación sexual o cruzamiento y mutación. Después de varios ciclos de evolución, la población deberá contener individuos más aptos.

Para distinguir entre los mejores y los peores individuos de la población se emplea una función objetivo o de evaluación, que desempeña el rol de medio ambiente, incorporando los requerimientos del problema a resolver. Se denomina *fitness* del cromosoma al valor que retorna la FO al ser evaluada en la solución factible representada por tal cromosoma.

El algoritmo mantiene una población $P(t) = \{x_{t,1}, x_{t,2}, \ldots, x_{t,m}\}$ de *m* cromosomas en cada iteración *t* y $x_{t,i}$, *i* = 1, 2,. . ., *m*, es cada individuo en ella. Una nueva población (iteración $t + 1$) se forma por la selección

de los cromosomas con mejor *fitness* (21). Los individuos seleccionados experimentan transformaciones por la aplicación de los operadores genéticos, denominados: mutación y cruzamiento, formando así individuos (soluciones factibles) nuevos.

La estructura de todo AG está conformada por los siguientes elementos (7):

- Problema a ser optimizado
- Representación de Soluciones del Problema
- Decodificación del Cromosoma
- Evaluación
- Selección
- Operadores Genéticos
- Inicialización de la Población
- Parámetros y Criterios de Parada

A continuación se abordan y especifican cada uno de los elementos mencionados anteriormente:

Problema a ser optimizado:

Los AGs son particularmente aplicados en problemas complejos de optimización: problemas con diversos parámetros o características que precisan ser combinadas en busca de la mejor solución; problemas con muchas restricciones o condiciones que no pueden ser representadas matemáticamente y problemas con grandes espacios de búsqueda (7). La Estimación de Parámetros en SED ordinarias se resuelve como un problema de optimización, por tanto, se define como tal para la presente investigación.

Representación de Soluciones del Problema:

La representación de las posibles soluciones dentro del espacio de búsqueda de un problema define la estructura del cromosoma que va ser manipulado por el algoritmo (7). Normalmente, la representación binaria es la más empleada por ser más simple y fácil de manipular a través de los operadores genéticos. Sin embargo, pueden ser usados otros tipos representaciones como un arreglo de objetos, de números, en dependencia del problema a tratar.

Para la representación cromosómica de la solución factible μ del problema a ser optimizado se utilizará un arreglo de números reales. Luego, un cromosoma será un vector de R_p, donde p es la dimensión del vector de parámetros μ y estará condicionada por la cantidad de parámetros a estimar. Se denota por $[a_i, b_i]$ el intervalo donde μ_i toma valores (21). Sin pérdida de generalidad se asume que los parámetros

están acotados ya que en la mayoría de las aplicaciones se conoce un rango donde dichos parámetros toman valores. Además, se considera una población de tamaño *m*.

Decodificación del Cromosoma:

La decodificación del cromosoma consiste básicamente en la construcción de la solución real del problema. El proceso de decodificación construye la solución para que esta sea evaluada por el problema. Sin importar el caso, esta fase se encarga de transformar cada gen en la realidad del problema.

Evaluación:

La evaluación es la unión entre el AG y el mundo externo. Se realiza a través de una función objetivo que representa de forma adecuada el problema y tiene como fin suministrar una medida de aptitud de cada individuo en la población actual.

Para cada miembro de la población se le calcula un valor de aptitud a través de su evaluación sobre el problema. El *fitness* es una cualidad determinante en el AG pues, el más apto es el que debe sobrevivir, ya que representa una mejor solución. Para medir el *fitness* de un cromosoma se precisa: resolver el SED del problema y evaluar la FO en la solución numérica obtenida (21), el valor alcanzado será el *fitness* del cromosoma.

La solución óptima debe tomar el mejor valor posible de *fitness* cuando sea evaluada por dicha función. Tomando en consideración que la optimización de la FO del problema específico a tratar viene dada por la tendencia a su minimización, las mejores soluciones serán las que posean menor valor de *fitness*.

Para el problema de optimización tratado los resultados de laboratorios y los experimentales son matrices de valores y para cada individuo (cromosoma) o posible solución se puede tener más de un resultado simulado. La FO debe comprender para cada experimento simulado todos sus valores, es decir, recorrer el total de filas y columnas de cada matriz que se obtuvo en las simulaciones para cada individuo.

La representación matemática de dicha función sería de la siguiente forma:

$$\mathcal{F}_{obj} = \sum_{k=0}^{n} \sum_{i=0}^{l} \sum_{j=0}^{m} \left(Y_{exp}(k,i,j) - Y_{sim}(k,i,j) \right)^2$$

Ecuación 2: Fórmula matemática de la Función Objetivo.

Donde:

- *n:* sería el total de experimentos simulados igual para cada individuo.

- *l:* la cantidad de filas de cada matriz (cantidad de tiempos para la cual fue medida experimentalmente o simulada en dependencia del caso).
- *m:* la cantidad de entidades o variables medidas.
- Y_{sim}: es la simulación de datos del modelo matemático.
- Y_{exp}: es el resultado de la transformación de los datos medidos en el experimento.

Literalmente es la sumatoria de la diferencia cuadrática que existe entre un valor experimental de una entidad y el simulado que se le hace corresponder, obtenidos para iguales tiempos. Esta función ha mostrado buen comportamiento como FO a optimizar para el problema de Estimación de Parámetros en SED ordinarias y es empleada en aplicaciones como LibSRES, la cual es una librería implementada en C para resolver problemas de optimización aplicando la estrategia estocástica de evolución. En próximos acápites serán abordadas con mayor profundidad sus características.

Selección:

El proceso de selección en los AGs se encarga de escoger individuos para la reproducción. La selección está basada en la aptitud de los individuos (cromosomas): individuos más aptos tienen mayor probabilidad de ser escogidos para la reproducción (7). En el problema específico tratado, se lleva a cabo la selección elitista como se especifica a continuación.

La selección elitista garantiza la selección de los miembros más aptos de cada población (22). Este operador selecciona el 90% de los individuos de la población de mejor *fitness*, para el caso estudiado serán aquellos de menor valor. El 10% restante lo conforman la clonación de los elementos seleccionados, uno por uno, con mejor aptitud, hasta que el tamaño de la población alcance el límite definido.

Operadores Genéticos:

Los individuos seleccionados (y reproducidos en la siguiente población) son recombinados sexualmente a través del operador de cruzamiento (22). Para la solución del problema planteado, se emplea el operador de cruzamiento: cruce por un punto, teniendo como tasa de cruzamiento el 0.35 del tamaño de la población, lo que significa que se realizará una cantidad de operaciones de cruce equivalentes al 35% de la población. Los pares de progenitores son escogidos aleatoriamente y nuevos individuos son generados a partir del intercambio del material genético.

El punto de cruce se obtiene de forma aleatoria, y el mismo debe ser menor que el tamaño de la cadena (cromosoma). Una vez obtenido este número, se copia a los dos individuos nuevos la parte de la cadena de sus padres que le corresponde. Los descendientes serán diferentes de sus padres pero con la

combinación de las características genéticas de ambos progenitores. Los cromosomas que no fueron cruzados son sometidos a la operación de mutación.

La mutación es un operador exploratorio que tiene como objetivo aumentar la diversidad en la población. Este operador altera el contenido de una posición del cromosoma, según una determinada probabilidad, en general baja (<1%) (7). La mutación se produce si un número entero aleatorio entre 0 y la tasa definida (de valor 12), es 0, lo que equivale una probabilidad de 1/12. Esta operación se aplica a cada gen en cada elemento de la población (excluyendo los producidos por el cruce) (22). Es notable destacar que la configuración de los operadores genéticos adoptada para el caso específico tratado viene dada por la configuración por defecto de la librería JGAP, la cual será descrita en próximos acápites.

Luego de aplicar los operadores genéticos, la nueva descendencia producida por cruce y completada con los cromosomas afectados por la mutación, sustituye la antigua población y el proceso vuelve a repetirse a partir de la evaluación.

Inicialización de la Población:

La inicialización de la población determina el proceso de creación de los individuos para el primer ciclo del algoritmo (7). Normalmente, la población inicial se forma a partir de individuos creados aleatoriamente ocurriendo de igual modo en el problema planteado, tomando valores dentro del intervalo definido para cada parámetro a estimar.

Parámetros y Criterios de Parada:

En un AG varios parámetros controlan el proceso de evolución (7):

- Tamaño de la Población: Constituye el número de puntos del espacio de búsqueda siendo considerados en paralelo.
- Tasa de Cruzamiento: Constituye la probabilidad de un individuo de ser recombinado con otro.
- Tasa de Mutación: Establece la probabilidad de que el contenido de cada posición/gen del cromosoma sea alterado.
- Número de Generaciones: Consiste en el número total de ciclos de evolución de un AG.
- Total de Individuos: Es número total de tentativas (tamaño de la población por número de generaciones).

Los dos últimos parámetros son empleados generalmente como criterio de parada de un AG. Se definen como criterios de parada para el problema de optimización tratado el margen de error y el número de iteraciones a realizar.

El margen de error definido es comparado con el valor del error global, el cual se calcula empleando la fórmula que se expone a continuación:

$$E_{global} = \sum_{k=0}^{n} \sum_{i=0}^{l} \sum_{j=0}^{m} | Y_{exp}(k,i,j) - Y_{sim}(k,i,j) |$$

Ecuación 3: Fórmula matemática del Error Global.

Básicamente esta fórmula representa la sumatoria del valor absoluto de la diferencia que existe entre un valor experimental de una entidad y el simulado que se le hace corresponder obtenidos para iguales tiempos. Este error se toma en cuenta para dar una medida real de la relación que tiene la solución encontrada con respecto a los datos experimentales.

En general, un AG puede ser descrito como un proceso continuo que repite ciclos de evolución controlados por un criterio de parada, como se representa en la Figura 5:

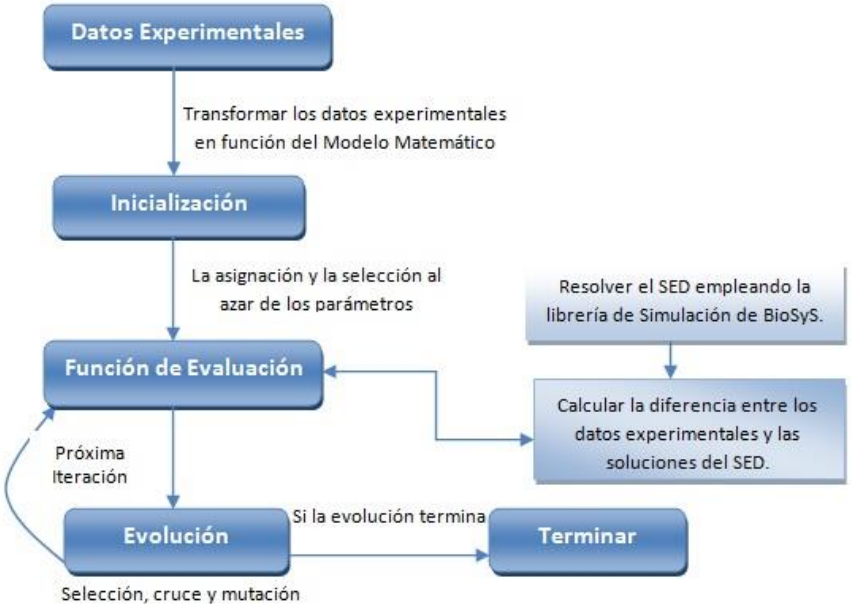


Figura 5: Representación de la ejecución del AG para el problema de optimización tratado.

El estudio de los AGs ha demostrado que constituyen una estrategia enormemente poderosa y exitosa para resolver problemas de optimización, tales como la Estimación de Parámetros en SED ordinarias. El análisis de una población grande de soluciones candidatas en cada iteración introduce una mejora: el paralelismo. El hecho de explorar varias posibilidades en una sola corrida, permite examinar rápidamente espacios grandes llegando a conclusiones generales sobre las mejores zonas, acelerando la exploración de vastos espacios. Con esta cualidad es posible en una sola iteración seleccionar entre muchas variantes y encaminar la búsqueda de forma paralela a los lugares más prometedores.

La propiedad fundamental que sustenta el uso de los AGs radica en la necesidad de combinar soluciones para obtener otras nuevas, originales y mejores. Esto parte de la base de que tendrán más posibilidad de ser escogidas, y por lo tanto combinadas, aquellas soluciones que han demostrado ser las más prometedoras. Esta necesidad se apoya en que si dos soluciones han mostrado un comportamiento bueno, entonces la solución resultante de la combinación de ellas debe utilizar los aspectos mejores de ambas de forma eficiente.

En la presente investigación se determina utilizar AGs, PSO y Búsqueda Directa como algoritmos para la resolución del problema de optimización tratado.

1.5. Soluciones informáticas: aplicaciones informáticas y bibliotecas de clases.

Dentro del proceso de desarrollo de software se emplean para estandarizar las soluciones informáticas aplicaciones informáticas y bibliotecas de clases.

Una aplicación informática es un tipo programa informático diseñado para permitir la ejecución de ciertas tareas o solucionar un problema específico. Las aplicaciones surgen de alguna necesidad concreta de los usuarios. Según su finalidad, suelen ser en general, la solución para ciertos problemas que requieren ser automatizados e inclusive pueden relacionarse con algunas ramas de las ciencias como la ingeniería, matemática y las telecomunicaciones.

Sin embargo, encontrar una aplicación informática que pueda ser reutilizada o adaptada a un nuevo problema resulta un poco difícil. Como solución a este impedimento, en ocasiones, se crean dentro de las aplicaciones informáticas, componentes reutilizables. Estos componentes facilitan el uso de las aplicaciones, aunque no en su totalidad sino aquella parte que resulte más significativa. Pero generar o crear estos componentes resulta costoso y en ocasiones engorroso, ya que las aplicaciones a los que pertenecen son enfocadas a un problema específico y descomponerlos implicaría perder algunas de sus

funcionalidades. Como alternativa a este inconveniente, se opta, en algunos casos, por el desarrollo de bibliotecas de clases.

Una biblioteca de clases es un conjunto de clases o plantillas de código escrito previamente, que pueden ser modificadas de forma modular y utilizadas por un programador en el desarrollo de una aplicación. Las bibliotecas de clases mejoran la reutilización de código, ya que incluyen todas las clases esenciales en un formato codificado, simplificando la programación y propiciando un aumento en la calidad del código. La personalización de la plantilla de clases se realiza acorde a las necesidades del problema al que se dará solución.

En el presente trabajo de diploma la biblioteca de clases a desarrollar es de propósito general, permitiendo estandarizar la Estimación de Parámetros y el Análisis de Sensibilidad en sistemas de ecuaciones diferenciales ordinarias. El desarrollo de esta biblioteca, así como la utilización del paradigma orientado a objetos para su concepción permitirá las siguientes ventajas:

- **Reusabilidad:** una misma implementación de los algoritmos permite resolver una amplia gama de problemas muy diferentes entre sí.
- **Facilidad de ampliación y de modificación:** se dispone de varios mecanismos de ampliación como son la herencia, la redefinición de métodos, polimorfismo, que permiten ampliar la biblioteca de clases con relativa facilidad. El encapsulamiento en clases de los elementos que conforman la biblioteca de clases permite modificar uno de ellos sin afectar al resto.
- **Facilidad de aprendizaje y transparencia de uso:** la biblioteca de clases oculta la implementación de los algoritmos, con lo que el usuario final no necesita conocerlo, únicamente rellenar las clases específicas al problema para utilizarla.

Soluciones existentes para la Estimación de Parámetros

En el mundo han sido desarrolladas algunas bibliotecas y aplicaciones que pretenden facilitar la Estimación de Parámetros, logrando tener éxito en las investigaciones utilizadas. Las más destacadas son: Biojava, Cellware, LibSRES, SBML-PET y JGAP. Aunque presentan grandes ventajas, evidencian también algunas deficiencias. A continuación se ha realizado un breve análisis de cada una.

- **Biojava:** Es un proyecto de código abierto que proporciona un framework en Java para el procesamiento de datos biológicos. Incluye objetos para manipular las secuencias biológicas, analizadores de archivos, soporta e implementa un cliente y servidor DAS, acceso a las bases de datos Ensemble y BioSQL. Además, proporciona herramientas para el análisis de secuencias

gráficas, rutinas potentes de análisis estadístico y una herramienta de programación dinámica. Está distribuida bajo licencia GPL.

Al introducir el paquete *org.biojavax.ga* es posible utilizar AGs, incorporando así la estrategia evolutiva como método de solución. (23) El marco de trabajo está diseñado para ser muy flexible al propiciar funcionalidades deseables por los investigadores, pero al ser un sistema de propósito general para procesamiento de datos biológicos se torna engorroso reajustarlo a problemas más específicos, constituyendo este su mayor impedimento.

- **Cellware:** Es una herramienta de modelado y simulación que se concentra en las reacciones químicas de los sistemas biológicos (24). Ofrece una interfaz gráfica de usuario para configurar el modelo de sistema que consta de reactivos, productos y las reacciones correspondientes que pueden tener propiedades específicas de velocidad de reacción. El modelo creado puede ser simulado, ya sea con algoritmos de simulación determinista, estocástica o híbridos y los resultados de la simulación pueden ser visualizados gráficamente.

  No solo ha sido diseñado para realizar el modelado y simulación de genes reguladores y de vías metabólicas, sino también ofrecer un entorno integrado de las diversas representaciones matemáticas, entre estas la Estimación de Parámetros. A pesar de las características expuestas no puede ser adaptado a un nuevo objetivo ya que su código fuente no se encuentra disponible para uso académico.

- **LibSRES:** Es una librería implementada en C como una estrategia evolutiva de algoritmos estocásticos para la Estimación de Parámetros, facilitando una rápida implementación de los programas informáticos para el estudio de la vías bioquímicas no-lineales. Distribuida bajo la licencia GPL (25).

  Aunque se encuentra disponible para uso académico presenta una desventaja notable: no es multiplataforma. Además, su uso requiere traducir todo su código a lenguaje Java, ya que este último es el que se pretende utilizar en la implementación de la biblioteca de clases.

- **SBML-PET** (Systems Biology Markup Language based Parameter Estimation Tool, en español Lenguaje de Marcado de Biología de Sistemas basado en Herramientas de Estimación de Parámetros). Esta herramienta ha sido diseñada para llevar a cabo la Estimación de Parámetros

para modelos biológicos, incluyendo vías de señalización, redes de regulación de genes y rutas metabólicas. SBML-PET soporta la importación y exportación de modelos en el formato SBML (26).

Está implementada en C y actualmente su código fuente no se encuentra disponible para uso académico. Además, la forma en que define los datos experimentales y su importación al sistema no se corresponden con lo que se desea lograr en la fase experimental de la presente investigación. Por tanto, no es posible su reutilización.

- **JGAP** (Java Genetic Algorithms Package, en español Paquete de AG en Java): Es un componente de Algoritmos Genéticos y Programación Genética implementado en Java. Proporciona mecanismos de base genética que pueden ser fácilmente utilizados para aplicar los principios evolutivos en la búsqueda de soluciones a problemas de optimización. Es una librería distribuida bajo la licencia GPL y Mozilla (27).

  JGAP fue diseñado para ser muy fácil de usar, es altamente modular, de modo que los desarrolladores pueden fácilmente perfeccionar y personalizar los operadores genéticos y otros sub-componentes. Su código fuente está disponible para el uso académico y existe documentación suficiente para su total estudio.

- **AlasBioSyS** (Software para la Simulación y el Análisis de Sistemas Biológicos): Integra algoritmos y herramientas necesarias para servir de apoyo en las investigaciones de aquellos científicos dedicados al estudio de la Biología de Sistemas. Este software incluye la modelación, simulación (local y utilizando computación Grid) y análisis utilizando técnicas de Inteligencia Artificial, así como una base de datos para almacenar la información tanto del sistema en estudio, como de los resultados obtenidos.

  Entre sus múltiples funcionalidades resaltan además: resolver sistemas de ecuaciones diferenciales haciendo uso de diferentes asistentes matemáticos y librerías (Matlab, Octave, ODEtoJava), realizar el análisis de series temporales y estimar parámetros en sistemas de ecuaciones. Además importa y exporta usando los estándares SBML y MathML. Es una aplicación desarrollada en Java, es multiplataforma siempre y cuando se instale la máquina virtual en su versión 1.6, como gestor de base de datos puede utilizar MySQL o PostgreSQL (28).

El análisis de estas herramientas evidencia que presentan potencialidades y facilidades para el diseño gráfico, el análisis y el procesamiento de datos biológicos, la modelación y simulación de procesos

bioquímicos, y en algunos casos hacen uso de la Estimación de Parámetros en sistemas biológicos. De igual modo, poseen limitantes relacionadas generalmente con la disponibilidad del código fuente para uso académico, incompatibilidad de lenguajes de programación o que no ofrecen todas las funcionalidades deseables.

No obstante, las particularidades de la librería JGAP demuestran que es ideal para la reutilización y adaptación a los intereses que se persiguen con el presente trabajo de diploma. La utilización de AG como técnica para la búsqueda de soluciones, en este caso los valores estimados, reporta grandes beneficios en los resultados de la investigación.

Se determina reutilizar además, la biblioteca para la Optimización basada en Enjambre de Partículas sobre la Plataforma T-arenal, desarrollada en el Departamento de Bioinformática, incorporando este algoritmo en la búsqueda de soluciones para el problema de optimización tratado. Esta inclusión permite obtener una biblioteca de clases más completa y mejor equipada funcionalmente para una futura reutilización, así como dar solución al problema tratado.

Las potencialidades evidentes del software AlasBioSyS resultan un base para la estandarización de la Estimación de Parámetros y el Análisis de Sensibilidad en SED ordinarias, puesto que hasta el momento se realizan ambos en distintos módulos. La creación de una solución que posibilite su integración, así como incorporar nuevos algoritmos y técnicas a los existentes, empleando las librerías JGAP y la Optimización basada en Enjambre de Partículas, permitirá obtener una solución que mejore los resultados que se obtienen en las investigaciones.

1.6. Tecnología y herramientas a utilizar

En el proceso desarrollo de software es fundamental la correcta elección de las tecnologías y herramientas que mejor se adapten y mayores facilidades brinden para el logro de las metas trazadas. Para llevar a cabo la documentación e implementación del presente trabajo fueron objeto de análisis teniéndose en cuenta las tendencias actuales, novedades y desempeño de cada una. Además, se enfatizan como candidatas aquellas definidas en la arquitectura de la Plataforma de Servicios Bioinformáticos. Basado en ello, se seleccionaron las que a continuación se enuncian.

1.6.1. Metodología de Desarrollo

Aunque la biblioteca de clases a desarrollar no es una aplicación informática resulta necesario documentar su proceso de desarrollo. Para ello se realiza el análisis que se expone a continuación.

Una metodología de desarrollo de software es un conjunto de procedimientos, técnicas, herramientas, y un soporte documental que ayuda a los desarrolladores a producir nuevo software (29). Actualmente, existen diferentes metodologías y técnicas para el desarrollo de software, las cuales fueron analizadas para posteriormente definir la arquitectura del proyecto, resultando como propuesta final la metodología Proceso Unificado Abierto (OpenUP).

OpenUP es una familia de plugging de procesos de fuente abierta. Conlleva un acercamiento ágil para el desarrollo del software. Provee, con solo un contenido fundamental, un conjunto simplificado de artefactos, roles, tareas y guías de trabajo. Es un proceso iterativo que es mínimo, completo, y extensible que apoya el desarrollo ágil, e incremental. Es aplicable a un amplio sistema de plataformas y de usos de desarrollo (30).

La elección de OpenUP como metodología de desarrollo reporta beneficios al ser apropiada para proyectos pequeños y de bajos recursos, permitiendo disminuir las probabilidades de fracaso e incrementar las posibilidades de éxito. Posibilita detectar errores tempranos a través de un ciclo iterativo. Como metodología ágil tiene un enfoque centrado en cliente y con iteraciones cortas.

La metodología OpenUP establece cuatro fases de desarrollo, las cuales se describen brevemente a continuación.

Fases de OpenUP (31)**:**

- **Concepción:** El objetivo de esta fase es capturar las necesidades del proyecto a desarrollar. Se identifican las funciones claves del sistema y se determina una posible solución.
- **Elaboración:** Segunda fase del ciclo de vida de OpenUP donde se tratan los riesgos significativos para la arquitectura. Además, se obtiene una comprensión más detallada de los requisitos. El propósito de esta fase es establecer la línea base de la arquitectura del sistema.
- **Construcción:** Esta fase está enfocada en el diseño, implementación y prueba de las funcionalidades para desarrollar un sistema completo. El propósito de la misma es completar el desarrollo del sistema basado en la arquitectura definida.
- **Transición:** Es la última fase, cuyo propósito es asegurar la entrega del sistema a los usuarios. Además, evalúa la funcionalidad y performance de la última versión entregada en la fase de construcción.

Partiendo de las características mencionadas anteriormente, se adopta OpenUP como metodología de desarrollo. Por tanto, se generarán solo algunos artefactos para su documentación y descripción, los cuales serán de utilidad para futuras reutilizaciones. Definiéndose entonces los Requisitos Funcionales y No Funcionales, las clases de diseños, representadas e interrelacionadas en el Diagrama de clases de diseño correspondiente. Se describe además, la estructura de la arquitectura a través de una Vista Lógica.

1.6.2. Herramienta CASE para el modelado

UML (acrónimo de Unified Modeling Language, en español Lenguaje Unificado de Modelado) es ante todo, un lenguaje que proporciona un vocabulario y reglas para permitir una comunicación (32). UML tiene una notación gráfica muy expresiva que permite representar en mayor o menor medida todas las fases de un proyecto informático. Parte desde el análisis con los casos de uso, el diseño con los diagramas de clases, objetos, entre otros, hasta la implementación y configuración con los diagramas de despliegue. Es independiente del lenguaje de implementación.

Por ser un método formal de modelado aporta las ventajas de mayor rigor en la especificación, verificación y validación del modelo realizado. Permite automatizar algunos procesos y posibilita la generación de código a partir de modelos y viceversa (32). Por lo antes expuesto se define adoptar como lenguaje de modelado UML en su versión 2.1.

Herramientas CASE: Visual Paradigm para UML

Las herramientas CASE (acrónimo de Computer Aided Software Engineering, en español Ingeniería de Software Asistida por Computadora) son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el costo de las mismas. Es un sistema de software que intenta proporcionar ayuda automatizada a las actividades del proceso de desarrollo de software (33). Actualmente las de mayor auge son: el ArgoUML, Rational Rose, Visual Paradigm, Easy CASE, Xcase, CASE Studio 2 y CASE Wise. Destacando dentro de estas: Rational Rose, muy conocida mundialmente para los clientes de Windows, y Visual Paradigm, herramienta que está tomando auge en la Comunidad de Software Libre.

Visual Paradigm para el Lenguaje Unificado de Modelado (VP-UML) es una herramienta completamente equipada para UML. Está diseñada para la construcción de sistemas de software de gran escala de forma fiable a través de un enfoque orientado a objetos, además, las transiciones de diseño a la aplicación se encuentran perfectamente integradas en la herramienta de modelado visual (34). Admite la integración con

los entornos de desarrollo: Eclipse/IBM, NetBeans IDE, Oracle JDeveloper BEA Weblogic, Visual Studio, entre otros.

VP-UML permite la modelación de procesos del negocio y la administración de requerimientos. Posibilita la generación de la capa Objeto-Relacional, de código e ingeniería inversa, incluyendo 10 lenguajes de programación, entre ellos: Java, C++, .NET, PHP y XML. Además, facilita la generación de reportes en formatos: PDF, MS Word, HTML. Por cuanto, se decide utilizar como herramienta de modelado para el desarrollo de la biblioteca de clases, Visual Paradigm para UML en su versión 8.0.

1.6.3. Lenguaje de Programación

Un lenguaje de programación es un idioma artificial diseñado para expresar procesos que pueden ser llevados a cabo por máquinas, particularmente las computadoras (35). Se clasifican según:

- Nivel de abstracción (bajo, medio y alto).
- Forma de ejecución (compilado e interpretado).
- Paradigma de programación (imperativo, funcional, lógico, orientado a objeto.

Java

Java es un lenguaje de programación de alto nivel orientado a objetos, fue diseñado para ser independiente de la plataforma y de entorno de ejecución (Máquina Virtual de Java, en inglés Java Virtual Machine (JVM)). Ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de estos (36). Este lenguaje tiene ventajas significativas con respecto a otros lenguajes y ambientes de programación que lo hacen adecuado para realizar cualquier tarea de programación.

Las ventajas más significativas del uso de este lenguaje se enumeran a continuación (37):

- **Orientado a objetos:** Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo.
- **Sencillez:** Característica que posibilita toda la funcionalidad de un lenguaje potente, pero sin las particularidades sofisticadas que provocan confusiones.
- **Robusto:** Java realiza verificaciones en busca de problemas tanto en tiempo de compilación como en tiempo de ejecución. La comprobación de tipos ayuda a detectar errores, lo antes posible, en el ciclo de desarrollo.

- **Seguridad:** El código en Java pasa por una serie de pruebas (un verificador de bytecodes) que comprueba el formato de los fragmentos de código y aplica un probador de teoremas para detectar fragmentos de código ilegal, el cual viola derechos de acceso sobre objetos o intenta cambiar el tipo o clase de un objeto) antes de ejecutarse en una computadora.

Partiendo de las características antes mencionadas, se selecciona Java en su versión 1.6 como lenguaje de programación para el desarrollo de la biblioteca de clases.

1.6.4. Entorno de Desarrollo Integrado

Las herramientas de desarrollo son aquellos programas o aplicaciones que tengan cierta importancia en el desarrollo de un programa (programación). Entre los más usados actualmente para la programación en Java pueden mencionarse: Eclipse y NetBeans. Para el desarrollo de la biblioteca de clases se consideró trabajar con NetBeans por una serie de aspectos que se detallan a continuación.

NetBeans

NetBeans es una herramienta para programadores, pensada para escribir, compilar, depurar y ejecutar programas. Aunque es escrito en Java puede servir para cualquier otro lenguaje de programación. Diseñado para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas. Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, entre otras (38).

NetBeans provee soporte completo para las últimas tecnologías Java y las mejoras más recientes. Proporciona herramientas de análisis estático, especialmente la integración con la ampliamente utilizada herramienta: FindBugs, para identificar y solucionar problemas comunes en código Java. Partiendo de las potencialidades de ser multiplataforma, integrarse con UML, sin restricciones de uso, y soportar varios sistemas de control de versiones, se concluye trabajar NetBeans 7.1 como entorno de desarrollo integrado.

1.7. Conclusiones

En el desarrollo de este capítulo se expuso todo el fundamento teórico que avala la implementación de la biblioteca de clases, motivo del actual trabajo de diploma. Se describió el concepto de BS, se abordaron las técnicas computacionales empleadas por los biólogos de sistemas, profundizando en la Estimación de Parámetros y el Análisis de Sensibilidad, como un caso específico de la misma. Se analizaron los algoritmos de optimización, haciendo fundamental énfasis en los AGs, PSO y la Búsqueda Directa,

abordando sus características y concluyendo que serán usados para dar solución al problema de optimización planteado.

Teniendo como base el análisis realizado de las metodologías y tecnologías actuales para el proceso de desarrollo de software, se define como metodología de desarrollo: OpenUP, lenguaje de modelado: UML en su versión 2.1, optando como herramienta CASE: Visual Paradigm para UML en su versión 8.0. Se selecciona como lenguaje de programación para la implementación: Java en su versión 1.6, y como entorno integrado de desarrollo: NetBeans en su versión 7.1.

CAPÍTULO 2: DESCRIPCIÓN Y DISEÑO

Introducción

En el siguiente capítulo se realiza una descripción de la biblioteca de clases, definiendo las principales funcionalidades a implementar, las cuales responden a la especificación de los requisitos funcionales. Se precisan además, los requisitos no funcionales que procurarán el rendimiento y la reutilización de la biblioteca de clases. Se define la arquitectura empleada en la elaboración de la biblioteca de clases y sus particularidades. Se fundamenta además, la utilización de patrones de diseño.

2.1. Modelo Conceptual

La metodología OpenUP no define el flujo de trabajo Modelado de Negocio. Se ha definido un Modelo Conceptual porque se hace necesario como parte de la documentación del proyecto BioSyS, además de brindar el mismo un mejor entendimiento del problema a resolver. Un modelo conceptual es una representación de conceptos en el dominio del problema en el mundo real, representa cosas del mundo real, no componentes del software (39).

En el Modelo Conceptual propuesto un **Estudio Experimental** está compuesto por un conjunto de **Experimento**, por un conjunto de **Condición** y de otro de **VariableExp**. Este estudio se asocia con un **Modelo Matemático**, conformado por varias **VariableMM** y **Coeficientes** que acompañan estas variables y por un grupo de **Condiciones Iniciales** para su resolución. Esta asociación es la transformación del estudio al modelo obteniéndose un **Estudio_Transformado** compuesto por un conjunto de **Experimento_Transformado**.

Para realizar la estimación se utiliza el **Estudio_Transformado** y el **Modelo Matemático** los que son asociados a **Estimador** quien se encarga de calcular el valor aproximado de los parámetros del modelo que se desean estimar y que son recogidos en **Valores Estimados**. Para realizar el Análisis de Sensibilidad se varía uno o varios **Coeficientes** del **Modelo Matemático** asociado al **Análisis Sensibilidad**, y una vez obtenidos los resultados se comparan con el **Estudio Transformado** asociado, como consecuencia de este proceso, se sabrá si dicho **Coeficiente** es sensible o no.

2.2. Definición de las clases del Modelo Conceptual.

Condicion_Variable: Concepto que representa una condición de entorno bajo la cual se realiza el Estudio, toma un valor de un conjunto de valores para cada Experimento de dicho Estudio.

Condicion_Fija: Concepto que representa una condición de entorno bajo la cual se realiza el Estudio, toma un único valor para todo Experimento del Estudio.

Condicion: Es el concepto general que abarca a Condicion_Variable y Condicion_Fija.

VariableExp: Concepto que representa a las Variables que se miden en un Estudio.

Experimento: Concepto que representa un experimento de laboratorio, el cual incluye los valores de las Condiciones bajo las cuales fue realizado y una tabla de resultados en la que cada celda contiene el valor de la VariableExp *N* para el tiempo *M*.

Estudio Experimental: Concepto que representa un conjunto de Experimentos realizados bajo un conjunto de Condiciones determinadas en el cual se miden variables experimentales específicas.

Modelo_Matematico: Constituye la modelación matemática del sistema biológico a estudiar, expresado en este caso por un SED ordinarias.

Condiciones Iniciales: Concepto asociado con los valores iniciales asignados a las variables para resolver el SED ordinarias.

Coeficientes: Concepto concerniente a los coeficientes usados en cada ecuación del SED ordinarias.

VariableMM: Son las variables que forman parte del modelo matemático, son determinadas a partir de las ecuaciones diferenciales existentes en el modelo.

Estimador: Concepto que representa la entidad encargada de estimar los valores de los parámetros. Comprende un conjunto de técnicas de búsqueda y optimización.

Valores Estimados: Concepto asociado a los valores de los parámetros que se pretenden estimar. Representa una lista de valores de los parámetros.

Experimento_Transformado: Concepto que representa a un experimento obtenido de la transformación de un Experimento a un Modelo_Matematico.

Estudio_Transformado: Representa el conjunto de todos los Experimentos de un Estudio que fueron transformados a un mismo Modelo_Matematico.

Análisis Sensibilidad: Concepto que representa la entidad encargada de realizar el análisis de sensibilidad a uno o varios coeficientes del modelo matemático.

A continuación se representa el Modelo Conceptual:

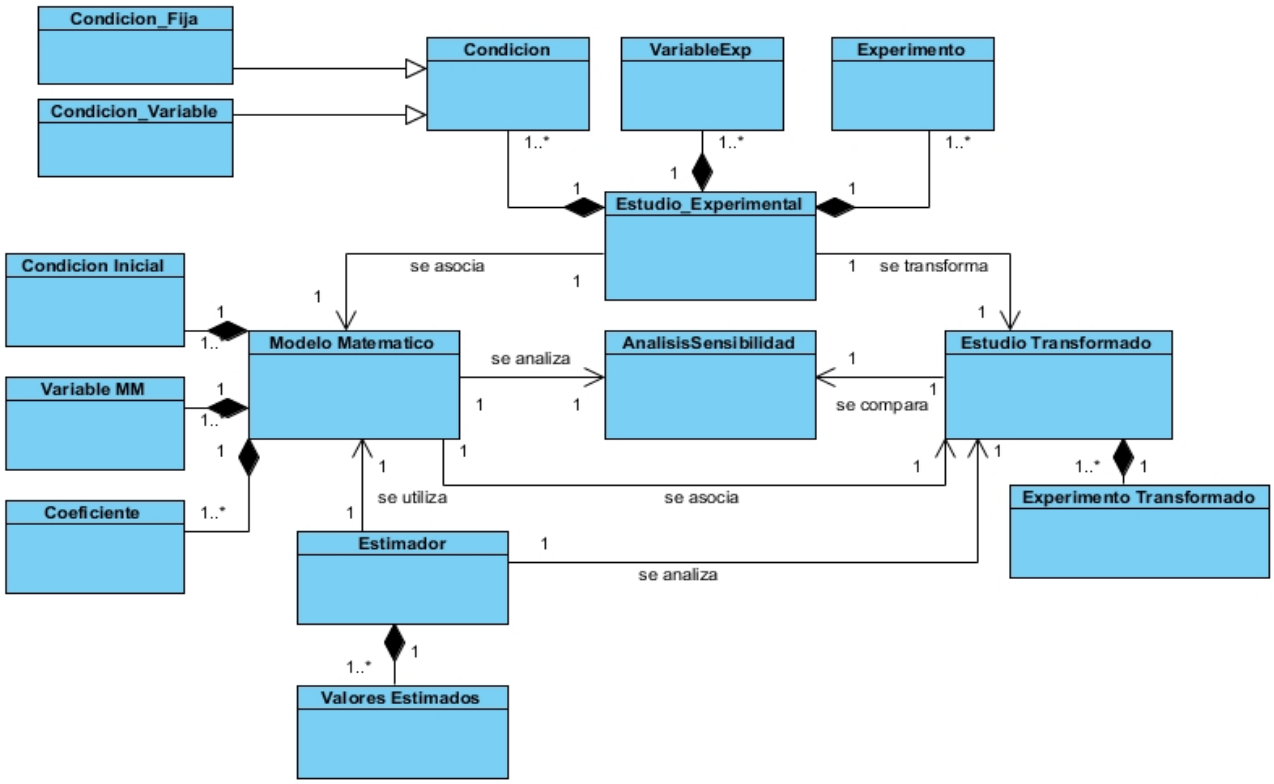


Figura 6: Modelo Conceptual

2.3. Especificación de requisitos de la Biblioteca de Clases

Un requisito es un atributo necesario en un sistema, una declaración que identifica una capacidad, característica, o factor de calidad de un sistema a fin de que tenga valor y utilidad a un cliente o usuario (40). En este sentido se definen dos tipos de requisitos: Requisitos Funcionales y Requisitos no Funcionales.

2.3.1. Requisitos Funcionales

Los Requisitos Funcionales son capacidades o condiciones que el sistema debe cumplir. Son el punto de partida para identificar qué debe hacer el sistema (40). Partiendo de los requisitos definidos para el Módulo de Estimación de Parámetros del proyecto BioSyS, se decide redefinir y establecer los siguientes para la biblioteca de clases a implementar:

RF1: Crear estudio experimental.

RF2: Cargar estudio experimental.

RF3: Definir Condiciones Iniciales del Modelo Matemático.

RF4: Transformar Condiciones Experimentales a Coeficientes.
RF5: Transformar Variables Experimentales a Variables del Modelo Matemático.
RF6: Estimar Parámetros con Algoritmos Genéticos.
RF7: Estimar Parámetros con Búsqueda Directa.
RF8: Estimar Parámetros con PSO.
RF9: Realizar Análisis de Sensibilidad a Parámetros.
A continuación se detallan cada uno de los requisitos funcionales expuestos:

El **RF1: Crear estudio experimental** permite, una vez especificadas las variables experimentales: el experimento y las condiciones que conformarán el estudio, crear el mismo. El **RF2: Cargar estudio experimental** permite cargar, proveniente de un fichero .xls, el estudio experimental ya creado. El **RF3: Definir Condiciones Iniciales del Modelo Matemático** permite establecer las condiciones iniciales a las cuales estará sujeto el modelo matemático. El **RF4: Transformar Condiciones Experimentales a Coeficientes** permite convertir las condiciones del entorno, entiéndase que son aquellas bajo las que se encuentra el experimento, en coeficientes del modelo matemático.

El **RF5: Transformar Variables Experimentales a Variables del Modelo Matemático** permite transformar las variables experimentales de dicho estudio experimental en variables del modelo matemático. El **RF6: Estimar Parámetros con Algoritmos Genéticos** permite realizar la estimación de los parámetros especificados empleando Algoritmos Genéticos. El **RF7: Estimar Parámetros con Búsqueda Directa** permite realizar la estimación de los parámetros aplicando el algoritmo de Búsqueda Directa. El **RF8: Estimar Parámetros con PSO** realiza la estimación de parámetros empleando PSO. El **RF9: Realizar Análisis de Sensibilidad a Parámetros** permite realizar el análisis de sensibilidad a uno o varios parámetros del modelo matemático, arrojando como resultado cuánto influye la variación de esos parámetros en la salida del sistema.

2.3.2. Requisitos no Funcionales

Los Requisitos no Funcionales son propiedades o cualidades que el producto debe tener (41). Representan las características del producto, procurando que este resulte atractivo, confiable, usable y seguro. De acuerdo a lo anteriormente planteado, se identificaron los siguientes Requisitos no Funcionales:

Software:

- Se debe disponer en las computadoras de sistema operativo Linux o Windows para el uso de la biblioteca de clases, garantizando una interoperabilidad de forma similar en diferentes sistemas operativos o plataformas.
- Se requiere tener instalada la Máquina Virtual de Java nombrada Java Runtime Enviroment (JRE) versión 1.6.

Hardware:

- Para el desarrollo y utilización de la biblioteca de clases se debe contar con 256 MB de espacio de memoria RAM o más.
- Disponer de 50 MB de capacidad de disco duro como mínimo.

Soporte:

- Mantenimiento: La biblioteca de clases debe contar con un *javadoc* y *un demo* para facilitar la reutilización y el mantenimiento de las funcionalidades implementadas.

Portabilidad:

- La biblioteca de clases podrá ser ejecutada en sistemas operativos Windows o cualquier distribución de Linux siempre que se haya instalado la Máquina Virtual de Java en su versión 1.6.

Requerimientos en el diseño y la implementación:

- Máquina Virtual de Java (JDK) en su versión 1.6.
- NetBeans 7.1 como entorno de desarrollo.
- Visual Paradigm para UML 8.0 como herramienta CASE.

2.4. Diseño de la Biblioteca de Clases

El flujo de trabajo de diseño se traza como objetivo transformar los requerimientos funcionales y no funcionales en un diseño de cómo va a ser implementada la biblioteca de clases, evolucionando hacia una arquitectura del software más sólida. Se persigue adaptar el diseño para que coincida con el ambiente de implementación, diseñando la biblioteca de clases con un enfoque hacia el rendimiento.

2.4.1. Arquitectura propuesta

La arquitectura de software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución (42). Es una vista estructural de alto nivel, define los estilos o grupos de estilos adecuados para cumplir

con los requerimientos no funcionales (43). En la presente investigación, la arquitectura o estructura a definir se basa en tratar de diseñar una biblioteca de clases que sea capaz de adaptarse a nuevas inclusiones o variaciones con un mínimo impacto en el resto de las clases, definiendo en cada caso la forma específica de comunicación con las funcionalidades que se brindan.

Estilos Arquitectónicos

Un estilo es un concepto descriptivo que define una forma de articulación u organización arquitectónica. El conjunto de los estilos cataloga las formas básicas posibles de estructuras de software, mientras que las formas complejas se articulan mediante composición de los estilos fundamentales (42). En una forma más específica, un estilo determina el vocabulario de componentes y conectores que pueden ser utilizados en instancias de este estilo con un conjunto de restricciones en las descripciones arquitectónicas.

Un ejemplo de los estilos arquitectónicos es la arquitectura de llamada y retorno.

Llamada y Retorno

Este estilo refleja la estructura del lenguaje de programación. Permite al diseñador del software construir una estructura de programa relativamente fácil de modificar y ajustar a escala. Se fundamenta en un razonamiento jerárquico, por lo que cambios en una subrutina implican cambios en las subrutinas que hacen la invocación (44). Miembros de la familia son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas (42).

Emplear este estilo en el diseño de la biblioteca de clases permitirá obtener una estructura organizada de manera tal que variaciones realizadas a funcionalidades o componentes específicos no afecten el funcionamiento de la biblioteca en general. Se definiría por tanto una estructura jerárquica en capas.

Arquitectura en capas.

La arquitectura en capas es un sub-estilo dentro del estilo Llamada y Retorno. Constituye una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas (43). Las restricciones topológicas del estilo pueden incluir una limitación, más o menos rigurosa, que exige a cada capa operar solo con capas adyacentes, y a los elementos de una capa entenderse solo con otros elementos de la misma.

Partiendo de las especificidades de la biblioteca de clases a implementar se definen solamente dos capas: la **Capa Gestión de Control** y la **Capa Entidad**.

Capa Gestión de Control: Es diseñada partiendo de la necesidad de un mediador entre el beneficiario que utilice la biblioteca de clases y las funcionalidades que esta brinda. Es en esta capa donde se establecen todas las reglas que deben cumplirse. Es la encargada de recibir las solicitudes que se le realizan a la biblioteca de clases y dar respuestas a estas, auxiliándose de la Capa Entidad para acceder a los datos almacenados y las funcionalidades específicas de cada clase de esta capa.

Esta capa está representada en la biblioteca de clases por la clase controladora "*CentralControlManager.java*".

Capa Entidad: Esta capa está conformada por las restantes clases que contendrá la biblioteca de clases, las cuales poseen particularidades específicas y realizan las funcionalidades acorde a estas. Es la capa donde residen los datos y es la encargada de acceder a estos.

En la biblioteca de clases esta capa está representada por clases como: "*TransformacionManager.java*", "*EstudioExperimental.java*", "*EstimadorClass.java*", "*AnalisisSensibilidad.java*", "*ExperimentoTransformado.java*", entre otras.

2.5. Vista Lógica de la arquitectura de software

La vista lógica de la arquitectura de software muestra los componentes principales de diseño y sus relaciones de forma independiente de los detalles técnicos y de cómo la funcionalidad será implementada en la plataforma de ejecución en términos de la estructura estática y comportamiento dinámico del sistema. Describe el diseño más importante de las clases y su organización en paquetes y subsistemas, y la organización de éstos en capas.

Para el caso específico tratado, la vista lógica de la arquitectura estará determinada por dos capas: la Capa Gestión de Control y la Capa Entidad, las cuales serán expuestas a continuación, detallando las clases que las componen y relaciones que denotan la comunicación entre estas.

Diagrama de Clases del Diseño

El diagrama de clases del diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación (39). En el Modelo Conceptual no se representa una definición de software; más bien una abstracción de un concepto del mundo real acerca del cual se quiere afirmar algo, en cambio, los diagramas de clases del diseño expresan, para el sistema computarizado, la definición de clases como componentes del software.

El Diagrama de clase es el diagrama principal de análisis y diseño para un sistema. En él, se especifica la estructura de clases del sistema, con relaciones entre clases y estructuras de herencia. Durante el análisis del sistema, el diagrama se desarrolla buscando una solución ideal. Durante el diseño, se usa el mismo diagrama, y se modifica para satisfacer los detalles de las implementaciones.

A continuación, se presenta el diagrama de clases del diseño de la biblioteca de clases a implementar.

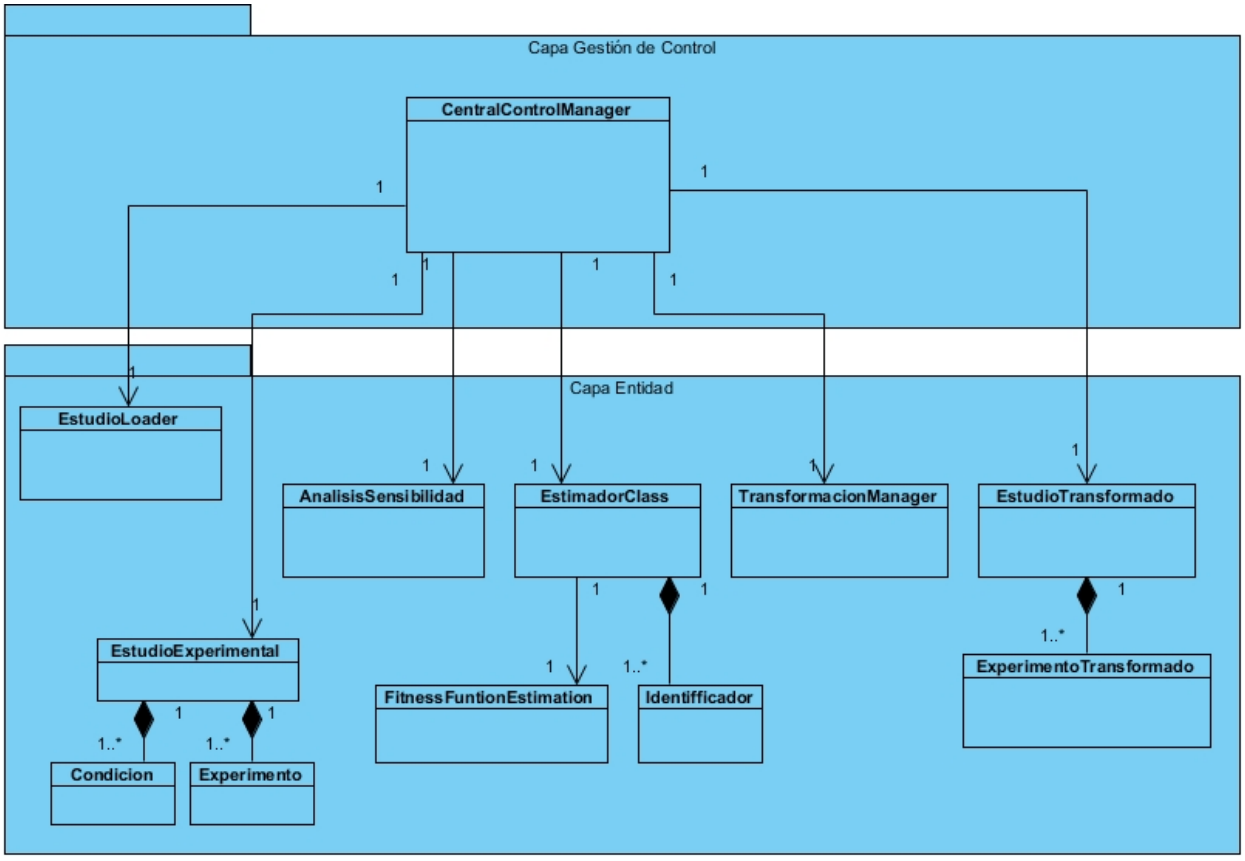


Figura 7: Diagrama de la Vista Lógica de las Clases de Diseño

2.6. Descripción de las Clases del Diseño

A continuación, se describen algunas de las clases del diagrama presentado, enfatizando en aquellas que resultan significativas en el desarrollo de la biblioteca de clases. A su vez, se agrupan estas clases en las capas correspondientes, definidas en la arquitectura propuesta.

Capa de Gestión de Control

Clase CentralControlManager: El propósito de esta clase es controlar todo el flujo de eventos de la biblioteca de clases. Desde una vista general, se encarga de gestionar la respuesta necesaria a todas las

peticiones realizadas. Para garantizar la Alta Cohesión y el Bajo Acoplamiento, ella delega en otras clases, aplicando además el patrón Experto, responsabilidades específicas y únicas dentro de la biblioteca de clases. Ver Anexo 1.

Capa Entidad

Clase TransformacionManager: El propósito de esta clase es gestionar la transformación del *EstudioExperimental* al modelo matemático, es decir, transforma cada Experimento del *EstudioExperimental* al modelo y como resultado genera un *ExperimentoTransformado*. Es esta clase la que define los elementos de un *ExperimentoTransformado*, labor que le fue delegada por *CentralControlManager*. Ver Anexo 2.

Clase EstimadorClass: El propósito de esta clase es gestionar la estimación, es decir, realizar la estimación de los parámetros definidos de la clase *EstudioExperimental* y utilizando la clase *FitnessFuntionEstimation*. Empleando para la búsqueda de soluciones AG, Búsqueda Directa o PSO. Ver Anexo 3.

Clase EstudioTransformado: El propósito de esta clase es gestionar todas las acciones que se realizan sobre cada *ExperimentoTransformado* tales como la creación y destrucción de cada uno. Ver Anexo 4.

Clase AnalisisSensibilidad: Esta clase tiene como propósito analizar si el sistema es sensible o no a la variación de uno o varios de los parámetros especificados. Ver Anexo 5.

2.7. Patrones de Diseño

Un patrón es una descripción de un problema y su solución que recibe un nombre y que puede emplearse en otros contextos. En teoría, indica la manera de utilizarlo en circunstancias diversas (39). Específicamente, un patrón de diseño es una solución estándar para un problema común de programación. Consiste en una técnica para flexibilizar el código, es una manera más práctica de describir ciertos aspectos de la organización de un programa.

La calidad de diseño de la interacción de los objetos y la asignación de responsabilidades presentan gran variación. Las decisiones poco acertadas dan origen a sistemas y componentes frágiles, difíciles de mantener, entender, reutilizar o extender. Una implementación hábil se funda en los principios cardinales que rigen un buen diseño orientado a objetos.

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos expresados en forma de patrones. GRASP es un acrónimo que significa: General Responsibility

Asignment Software Patterns (en español, Patrones Generales de Software para Asignar Responsabilidades) (45). En el desarrollo de la biblioteca de clases se emplean algunos de estos patrones como garantía de un diseño eficaz y orientado a objetos. A continuación se exponen.

Experto: Este patrón plantea asignar una responsabilidad al experto en esa información (39). Este experto lo constituye la clase que cuenta con la información necesaria para cumplir la responsabilidad asignada.

En la biblioteca de clases este patrón se evidencia en la clase *EstudioExperimental*. Esta clase cuenta con la información necesaria para crear y modificar una *Condición*, esta última a su vez, conserva los datos requeridos que debe tener una *Condición Experimental*. Ver Figura 8.

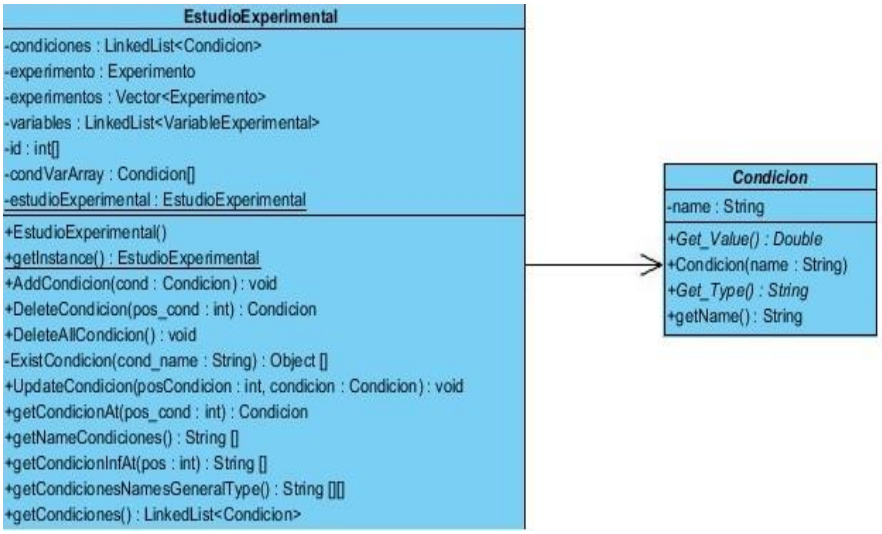


Figura 8: Aplicación del patrón Experto.

Creador: Este patrón guía la asignación de responsabilidades relacionadas con la creación de objetos. Propone asignar a una clase la responsabilidad de crear los objetos de la otra en los casos de contener, agregar, registrar o utilizar a esta (45).

En el caso específico, la clase *EstudioExperimental* tiene la responsabilidad de crear y contener instancias de las clases *Condición* y *Experimento* ya que posee la información necesaria o datos de inicialización de ambas. Ver Figura 9.

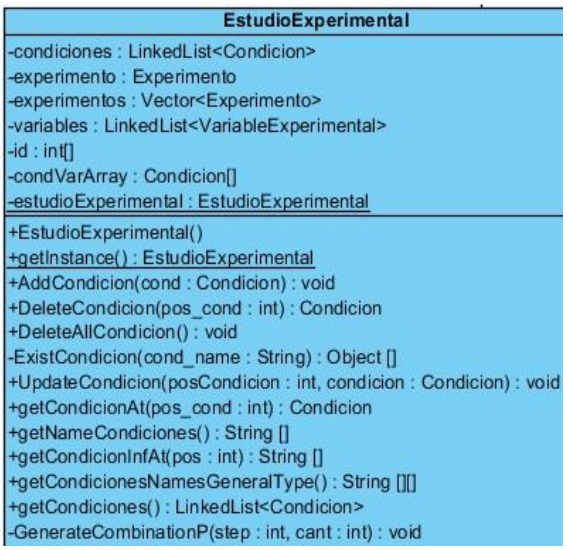


Figura 9: Aplicación del patrón Creador

Bajo Acoplamiento: Este patrón se basa en asignar una responsabilidad para mantener Bajo Acoplamiento. El acoplamiento es una medida de la fuerza con la que una clase está conectada a otras clases, con qué las conoce y con qué recurre a ellas (39). El grado de acoplamiento no puede considerarse aisladamente de otros principios como Experto y Alta Cohesión. Sin embargo, es un factor a considerar cuando se intenta mejorar el diseño.

El Bajo Acoplamiento soporta el diseño de clases más independientes que reducen el impacto de los cambios, y también más reutilizables, acrecentando la oportunidad de una mayor productividad. Este patrón es un principio que asigna la responsabilidad de controlar el flujo de eventos del sistema a clases específicas. Esto facilita la centralización de actividades. El controlador no realiza estas actividades, las delega en otras clases con las que mantiene un modelo de Alta Cohesión.

Un ejemplo, para el caso actual, es la clase *CentralControlManager* que delega a la clase *TransformacionManager* el proceso de transformar un *EstudioExperimental*. Por su parte *TransformacionManager* contiene un conjunto de funcionalidades relacionadas entre sí que se encargan de realizar dicha transformación. Igualmente, *CentralControlManager* delega otras actividades específicas en las clases *EstudioExperimental*, *EstimadorClass* y *AnalisisSensibilidad*.

Alta Cohesión: Este patrón propone asignar la responsabilidad de manera que la complejidad se mantenga dentro de límites manejables, asumiendo solamente las responsabilidades que deben manejar, evadiendo un trabajo excesivo (39). Su utilización mejora la claridad y facilidad con que se entiende el

diseño. Permite simplificar el mantenimiento y las mejoras de funcionalidad, genera un Bajo Acoplamiento y soporta mayor capacidad de reutilización.

La cohesión es una medida de la fuerza con la que se relacionan las clases y el grado de focalización de las responsabilidades de un elemento. Cada elemento de diseño debe realizar una labor única dentro del sistema, no desempeñada por el resto de los elementos y auto-identificable (45). Una clase con baja cohesión hace muchas cosas no relacionadas o hace demasiado trabajo, radicando en estos inconvenientes la importancia de la correcta utilización de este patrón.

Este patrón se ha aplicado en clases como *EstudioExperimental* cuya labor es única dentro de la biblioteca de clases y es la de controlar los procesos que ocurren en la fase experimental, específicamente, manejar las acciones sobre las *Condiciones* y *Experimentos*. También se evidencia la aplicación de este patrón en las clases *TransformacionManager*, ya descrita, en la clase *EstimadorClass* que brinda la funcionalidad exclusiva de estimar los parámetros aplicando los algoritmos implementados y en la clase *AnalisisSensibilidad* cuya función es el análisis de sensibilidad de parámetros.

2.8. Conclusiones

En este capítulo se realizó la descripción de la biblioteca de clases a implementar, definiendo las particularidades y funcionalidades de la misma. Se estableció el Modelo Conceptual que enmarca el entorno donde se situará la biblioteca de clases. Fueron identificados los Requisitos Funcionales, denotando estos las funcionalidades requeridas en la biblioteca de clases; y los Requisitos no Funcionales.

Se presentó el diseño de la biblioteca de clases, definiéndose como arquitectura a implementar: la arquitectura en capas. Se fundamentó el uso de patrones de diseño con la finalidad de obtener una biblioteca de clases robusta, de fácil comprensión, mantenimiento y reutilización. Se presentó el diagrama de clases del diseño, detallando aquellas clases que se consideran de mayor impacto en la implementación de la biblioteca de clases.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

Introducción

En este capítulo, y teniendo como base el diseño desarrollado anteriormente, se realiza el análisis de la implementación que se llevará a cabo. Se expone la biblioteca de clases en términos de componentes, es decir, ficheros de código fuente y ejecutables. Una vez definida la mayor parte de la arquitectura durante el diseño, se pasa a desarrollar la arquitectura y la biblioteca de clases como un todo. Luego de la implementación se procede a la realización de pruebas para garantizar que la biblioteca de clases funcione correctamente y que los resultados obtenidos sean aceptables.

3.1. Estándares de codificación

Un estándar de codificación comprende los aspectos de la generación de código. Si bien los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. Además, si se aplica de forma continua un estándar de codificación bien definido, se utilizan técnicas de programación apropiadas y, posteriormente, se efectúan revisiones del código de rutinas, existen posibilidades de que un proyecto de software se convierta en un sistema de software fácil de comprender y de mantener.

Estilos de codificación empleados:

Durante la implementación de la biblioteca de clases, fue necesario utilizar algunos estilos de codificación en busca de un estándar que aportara una mayor organización al código.

Identación: Las normas de identación indican la posición en la que se deben colocar los diferentes elementos que se incluyen en el código fuente. El identado debe ser de cuatro espacios para:

- Declaraciones dentro de las clases.
- Enunciado dentro de los métodos y funciones.
- Enunciados dentro de bloques y comandos.
- Enunciados dentro de cuerpos switch/case.

Declaración de variables, parámetros, objetos y métodos: El nombre de las variables y parámetros debe comenzar con la primera letra en minúscula, la cual identificará el tipo de datos al que se refiere, en

caso que sea un nombre compuesto, se empleará notación lowerCamelCase[2]. Además, los métodos están compuestos por múltiples palabras juntas iniciando cada palabra con letra mayúscula.

Clases: El objetivo fundamental es nombrar las clases e instancias de forma estándar para la biblioteca de clases. Los nombres de las clases deben comenzar con la primera letra en mayúscula y el resto en minúscula, en caso de que sea un nombre compuesto se empleará notación CamelCase[3].

Comentarios, separadores, líneas, espacios en blanco y márgenes: Establecer un modo común para comentar el código, utilizar separadores, línea, espacios en blanco y márgenes, de forma tal que sea comprensible el código, siguiendo las pautas que a continuación se muestran:

- Comentar al inicio de la clase o función especificando el objetivo de la misma.
- Dejar una línea en blanco antes y después de la declaración de una clase o de una estructura y de la implementación de una función.
- Evitar comentar cada línea de código. Cuando el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco. En caso de que se necesite comentar una sola instrucción se suprime la línea en blanco o se escribe a continuación de la instrucción. No se debe usar espacio en blanco:
- Después del corchete abierto y antes del cerrado de un arreglo.
- Después del paréntesis abierto y antes del cerrado.
- Antes de un punto y coma.

3.2. Principales métodos implementados

Para obtener los resultados esperados de la biblioteca de clases, se implementaron dos métodos fundamentales para realizar la Estimación de Parámetros:

- Método **EstimarConAG**, realiza la estimación de los parámetros definidos empleando AG.
- Método **EstimarConPSO**, estima los parámetros definidos empleando PSO.

Además de estos métodos se utilizó:

- Método **evaluate**, devuelve el valor del fitness del cromosoma.

Para realizar el Análisis de Sensibilidad fue implementado el método:

[2] lowerCamelCase: es un estilo de escritura que se aplica a frases o palabras compuestas. Se caracteriza por colocar en minúscula la primera letra de un nombre y la primera letra de cada palabra interna en mayúscula.
[3] CamelCase: es un estilo de codificación que consiste en usar mayúscula al principio de un nombre y al inicio de cada palabra que contenga el identificador.

- Método **sensitivityAnalysis**, determina cuán sensible es la salida del sistema ante la variación de sus parámetros.

Además de este método se empleó:

- Método **comparar**, compara la diferencia entre los resultados de la simulación realizada y la simulación tomada como patrón, elemento a elemento.

A continuación se expone parte del código fuente de la implementación de uno de los métodos citados anteriormente, la especificación de algunos de los restantes podrá ser consultada en los Anexos 6 y 7 respectivamente.

3.2.1. Método EstimarConAG

El siguiente fragmento de código pertenece a la clase *EstimadorClass.* El propósito de esta clase es gestionar la estimación de parámetros empleando para ello AG, Búsqueda Directa o PSO. En este caso se expone el método de estimación de parámetros con AG.

El método recibe un conjunto de estudios transformados a través de "*transformados*" que es el resultado de haber transformado un Estudio Experimental a un Modelo Matemático definido en *"odeModelo*", los parámetros a estimar son establecidos en *"identificadores"*. El rango de valores en el que debe estar comprendido el valor de cada parámetro viene dado por *"parametersInterval*".

Otros elementos necesarios para gestionar la corrida del algoritmo son: *"iterCount", "organimsCount"* y *"errorBound"*, quienes indican, respectivamente, la cantidad de iteraciones o evoluciones que va a efectuar el algoritmo, la cantidad de individuos o soluciones candidatas a tener en cada población y el margen de error que se desea que tenga la solución a encontrar. La cantidad de iteraciones y el margen de error especificado son los criterios de parada en la ejecución del algoritmo.

Algoritmo para estimar los valores de los parámetros especificados usando AG

```
public void EstimarConAG(EstudioTransformado transformados,
            Identificador[] identificadors, double[][] parametersInterval,
ODE odeModelo, int iterCount, int organimsCount, double errorBound) throws
Exception {

 /*Configuración de la clase que gestiona el Algoritmo Genético. Se crea una
configuración con valores predeterminados.*/

        Configuration.reset();
        Configuration conf = new DefaultConfiguration();
        Configuration.reset();
```

```
 /*especificando que la optimización de la función objetivo viene dada por la
tendencia a su minimización, es decir, las mejores soluciones tienen menor
fitness para este caso */

        conf.setFitnessEvaluator(new DeltaFitnessEvaluator());
        conf.setPreservFittestIndividual(true);

/*definiendo la función de aptitud a optimizar y se modifica en la
configuración*/

        FitnessFunctionEstimation ffe = new
FitnessFunctionEstimation(transformados, identificadors, odeModelo);
        conf.setFitnessFunction(ffe);

/*definiendo el rango de valores para cada gen, la cantidad de genes es igual
a  la cantidad de parámetros que se estimarán*/

        Gene[] sampleGenes = getSampleGene(parametersInterval, conf);

/*Indicando a la configuración como serán los cromosomas *Se crea un
cromosoma de ejemplo y se carga en la configuración.*/

        IChromosome sampleChromosome = new Chromosome(conf, sampleGenes);
        conf.setSampleChromosome(sampleChromosome);

/*definiendo la población inicial de forma aleatoria */

        conf.setPopulationSize(organimsCount);//Indicando el tamaño de la
población
        Genotype population = Genotype.randomInitialGenotype(conf);
        currentIterAG = 0;
        lastUsedMethod = 1;
        configBEstocastica = new double[]{iterCount, organimsCount,
errorBound};

/* proceso de evolución de la población que se realiza tantas veces como el
valor de iterCount (cantidad de iteraciones)*/

        for (; currentIterAG < iterCount; currentIterAG++) {
            population.evolve();// método que evoluciona la población actual
para obtener resultados más aptos

/* Obteniendo el cromosoma más apto una vez evolucionada la población*/

            bestSolution = population.getFittestChromosome();
            arrayBestSolution = getBestSolutionGenesValue();

/*calculando el fitness y los errores que genera la mejor solución obtenida
en esta evolución*/
```

```
            dataSimpleSearch =
Calculator.CalculateCuadraticDiffAndGlobalErrorsChromosome(arrayBestSolution,
transformados, identificadors, odeModelo);
            leastOneSol = true;

/*condición de parada regida por el margen de error*/

            if (dataSimpleSearch[1] <= errorBound) {
                break;
            }
        }
    }
```

Tabla 1: Método EstimarConAG de la clase *EstimadorClass*

Los resultados de este algoritmo, dígase *fitness* de la solución, así como los tipos de errores que acarrea, son almacenados en el atributo *"dataSimpleSearch"*, al cual se accede desde la clase *CentralContorlManager* una vez finalizada la búsqueda.

3.2.2. Método evaluate

El segmento de código que se muestra a continuación pertenece a la clase *FitnessFunctionEstimation* que interviene en la estimación de parámetros con AG. Este algoritmo, a través de la función objetivo a optimizar, en este caso la diferencia cuadrática, se encarga de calcular y devolver el valor del *fitness* (nivel de aptitud) que posee un cromosoma (posible solución) respecto a los experimentos transformados definidos.

Algoritmo para devolver el *fitness* que posee un cromosoma

```
protected double evaluate(IChromosome cromosoma) {
        Identificador identificador;
        ExperimentoTransformado transformado;
        double[]times;
        double fitness=0;

/* ciclo que se realiza para cada experimento transformado*/

        for (int i = 0; i < transformados.getTransformados().length;
i++) {
            transformado = transformados.getTransformados()[i];
            double[]cond = transformado.getCond().clone();
            double[]coef= transformado.getCoef().clone();
            times = transformado.getTiempos();

/*combinando los genes(valores estimados) del cromosoma con los
conocidos del experimento transformado para construir la solución real
del problema*/
```

```
            for (int j = 0; j < identificadors.length; j++) {
                identificador=identificadors[j];
                if(identificador.getTipo() =='i' )
cond[identificador.getPos()]=(Double)cromosoma.getGene(j).getAllele();
                else
coef[identificador.getPos()]=(Double)cromosoma.getGene(j).getAllele();
            }

/*simulando dado el conjunto de condiciones iniciales, coeficientes y
tiempos necesarios(solución construida) para el modelo dado por
odeModelo */

            Set test = SimulacionAux.SimularWithJava(cond, coef,
times, odeModelo);
            Object [] resultados=test.toArray();

/*ordenando el set de resultados obtenido */

            UtilArrayManager.QuickSort(0, resultados.length-1,
resultados);
            UtilArrayManager.SortAfterQuickSort(cond.length,
resultados);

/*calculando el fitness que posee el cromosoma para el experimento
transformado analizado dado por transformado y acumulándolo en
fitness*/
            fitness +=
Calculator.CuadraticDifference(transformado.getMatriz(), resultados,
cond.length);

        }       //fin del ciclo

/*devolviendo el fitness que posee el cromosoma para todos los
experimentos transformados*/

        return fitness;
    }
```

Tabla 2: Método evaluate de la clase *FitnessFunctionEstimation*

3.3. Pruebas

Las pruebas son actividades en las cuales un sistema o componente es ejecutado bajo condiciones o requerimientos determinados, los resultados son observados, registrados y analizados (41). La prueba de software es un elemento crítico para la garantía de la calidad del mismo y representa una revisión final de las especificaciones del diseño y de la codificación.

Existen muchas estrategias que pueden ser empleadas para probar un software. Se necesita una visión progresiva de las pruebas a partir de las pruebas de las unidades individuales del programa, pasando a

pruebas diseñadas para facilitar la integración de las unidades, y culminando con pruebas que ejecutan el sistema construido. La prueba es aplicada para diferentes tipos de objetivos, en diferentes escenarios o niveles de trabajo. Se distinguen los siguientes niveles de pruebas (46):

- Prueba de desarrollador
- Prueba independiente
- Prueba de unidad
- Prueba de integración
- Prueba de sistema
- Prueba de aceptación

Para validar las funcionalidades implementadas en la biblioteca de clase se realizaron pruebas de unidad.

3.3.1. Pruebas Unidad

Las pruebas de unidad están enfocadas al código fuente de los componentes, verificando todos los flujos de control. De los métodos de prueba existentes para este nivel se aplica la prueba de caja blanca, ya que la biblioteca de clases no cuenta con interfaz visual y es un requerimiento imprescindible validar el correcto funcionamiento de los métodos implementados. La prueba de caja blanca comprueba los caminos lógicos del software proponiendo casos de prueba que ejerciten conjuntos específicos de condiciones y/o bucles (41).

La técnica empleada para el método de caja blanca es la prueba de camino básico, para ello es necesario conocer el número de caminos independientes de un determinado algoritmo mediante el cálculo de la complejidad ciclomática. Esta prueba le permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución (46). Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Aplicación de la Técnica Camino Básico

La técnica camino básico es aplicada a todos los métodos implementados en la biblioteca de clases. Para este caso, se expone su aplicación sobre la funcionalidad encargada de gestionar la estimación de parámetros empleando AG, el método *EstimarConAG.*

Se identificaron 7 bloques de ejecución para este método atendiendo a las dependencias procedimentales, los cuales se representan en la Figura 10. Se construye el grafo de flujo asociado como

se muestra en la Figura 11. Se identifican, además, dos nodos predicados: el número 2 y 4, siendo de estos de donde emergen dos o más aristas.

```
public void EstimarConAG (EstudioTransformado transformados,
            Identificador [] identificadors, double [][] parametersInterval,
ODE odeModelo, int iterCount, int organismsCount, double errorBound) throws
Exception {
        Configuration.reset ();
        Configuration conf = new DefaultConfiguration ();
        Configuration.reset ();
        conf.setFitnessEvaluator (new DeltaFitnessEvaluator ());
        conf.setPreservFittestIndividual (true);
        FitnessFunctionEstimation ffe = new FitnessFunctionEstimation
(transformados, identificadors, odeModelo);
        conf.setFitnessFunction (ffe);
        sampleGenes = getSampleGene (parametersInterval, conf);
        IChromosome sampleChromosome = new Chromosome (conf, sampleGenes)
        conf.setSampleChromosome (sampleChromosome);
        conf.setPopulationSize (organismsCount);
        Genotype population = Genotype.randomInitialGenotype (conf);
        currentIterAG = 0;
        lastUsedMethod = 1;
        configBEstocastica = new double [] {iterCount, organismsCount,
errorBound};

for (; currentIterAG < iterCount; currentIterAG++) {
            population.evolve ();
            BestSolution = population.getFittestChromosome ();
            arrayBestSolution = getBestSolutionGenesValue ();
            dataSimpleSearch =
Calculator.CalculateCuadraticDiffAndGlobalErrorsChromosome (arrayBestSolutior
transformados, identificadors, odeModelo);
            leastOneSol = true;

    if (dataSimpleSearch [1] <= errorBound) {
        break;
    }
}
}
```


Figura 10: Código fuente del método *EstimarConAG*

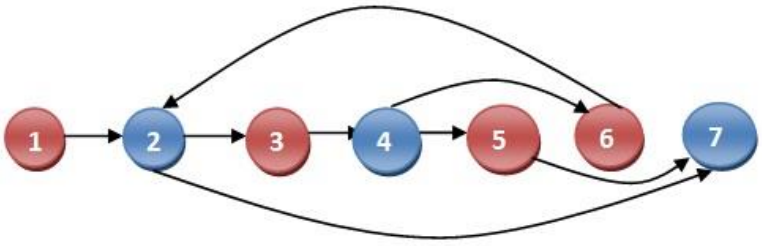


Figura 11: Grafo de flujo del método *EstimarConAG*

Se procede entonces al cálculo de la complejidad ciclomática del grafo de flujo ($V(G)$). Para esta operación existen tres vías de solución, las cuales se enuncian a continuación:

- $V\,(G)\;=\;(A-N)\;+\;2$
- $V\,(G)\;=\;P\;+\;1.$
- $V\,(G)\;=\;R$

Siendo *A* la cantidad total de aristas, *N* la cantidad total de nodos, *P* la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas) y *R* la cantidad total de regiones (41).

Se aplican las tres formas para afirmar un resultado seguro y confiable. Luego, se obtiene una complejidad ciclomática $V(G) = 3$. La cifra de la complejidad representa la cantidad de caminos independientes para el grafo de flujo construido para el método *EstimarConAG.*

Luego de tener elaborado el grafo de flujo e identificados los caminos a recorrer, se preparan los casos de prueba que forzarán la ejecución de cada uno de esos caminos. Se escogen los datos de manera que las condiciones de los nodos predicados estén adecuadamente establecidas, con el fin de comprobar cada camino. A continuación se especifican estos casos de prueba.

Casos de prueba para cada camino básico:

Descripción: Se inicializan las variables necesarias para realizar la estimación de los parámetros especificados, empleado como técnica AG. Para éste caso se adoptará solo variar los valores de las variables *"iterCount"* y *"errorBound",* ya que estas son las condiciones de parada del algoritmo.

Camino 1: 1, 2, 3, 4, 6, 2, 7

- **Entrada:** Se define para las variables los siguientes valores: iterCount = 100, errorBound =0.

Camino 2: 1, 2, 3, 4, 5,7

- **Entrada:** Se define para las variables los siguientes valores: iterCount = 10, errorBound =5.

Camino 3: 1, 2, 7

- **Entrada:** Se define para las variables los siguientes valores: iterCount = 10, errorBound =0.25.

Partiendo de las 9 funcionalidades implementadas en la biblioteca de clases se aplican 30 casos de prueba para validar su correcto funcionamiento, realizando para ello 3 iteraciones. En la primera iteración fueron arrojados un total de 12 NC, las cuales fueron corregidas en su totalidad. Tras la segunda iteración se detectaron 8 errores, también corregidos, finalizando en una tercera iteración sin errores Los resultados obtenidos son graficados a continuación.

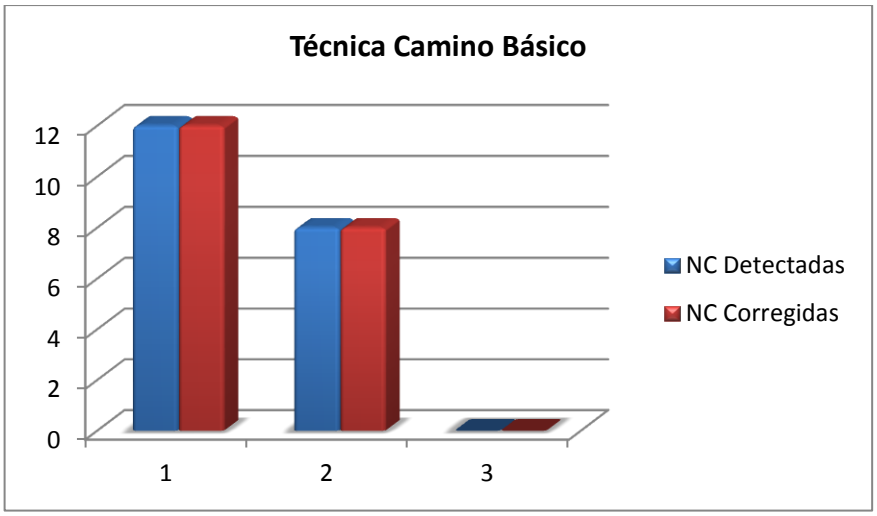


Gráfico 1: Relación de No Conformidades (NC) Detectadas y NC Corregidas para la técnica: Camino Básico.

Luego de aplicar los distintos casos de pruebas, se pudo comprobar que el flujo de trabajo de la biblioteca de clases es correcto ya que cumple con las condiciones necesarias que se habían planteado. Los resultados obtenidos en cada iteración realizada permitieron rectificar los errores detectados para completar el correcto funcionamiento de la biblioteca de clases.

3.3.2. Pruebas de Sistema

Para finalizar la validación de las funcionalidades implementadas y corroborar los resultados obtenidos en la prueba de unidad se realizan además, las pruebas de sistema. Las pruebas de sistema son pruebas que se hacen cuando el software está funcionando como un todo, dirigidas a verificar el programa en su totalidad (41).

Como método para este tipo de prueba se aplica la prueba de caja negra. Aunque esta prueba se lleva a cabo sobre la interfaz del software, y en el caso particular de la biblioteca de clases no existe tal, en las ejecuciones realizadas para comprobar el correcto funcionamiento de la implementación se emplea una consola para mostrar los resultados generados tras cada ejecución.

Dentro del método de caja negra la técnica de la partición de equivalencia es una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el software. El diseño de casos de prueba para la partición equivalente se basa en una evaluación de las clases de equivalencia para una condición de entrada. Una clase de equivalencia representa un conjunto de estados válidos o

inválidos para condiciones de entrada (41). Se determina aplicar esta técnica a la biblioteca de clases implementada.

Para validar las 9 funcionalidades implementadas se aplican 18 casos de pruebas, realizando 3 iteraciones. Se obtienen como resultados para la primera iteración realizada 10 NC, siendo solucionadas en su totalidad. Para la segunda iteración se detectan 4 NC, solucionadas también; concluyendo la tercera iteración con resultados satisfactorios.

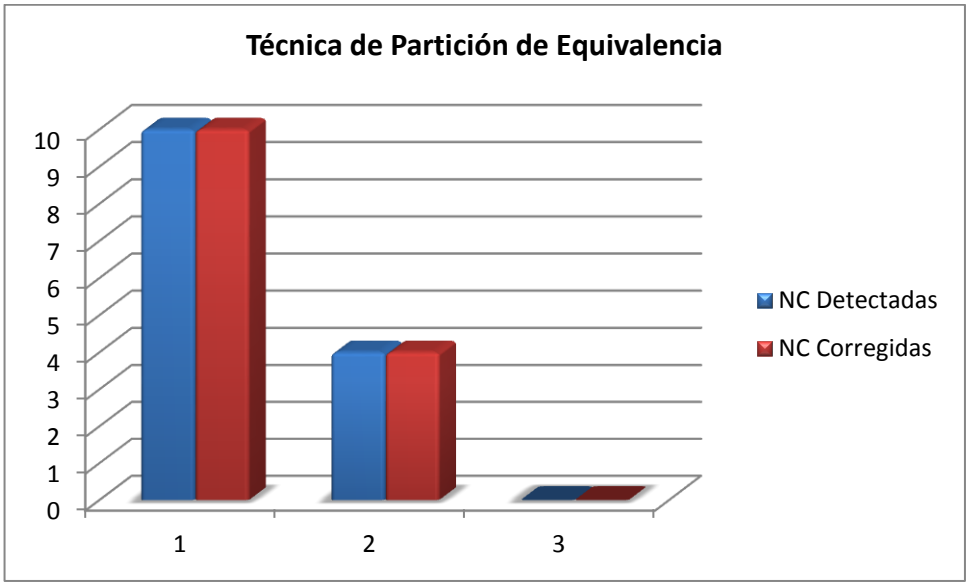


Gráfico 2: Relación de No Conformidades (NC) Detectadas y NC Corregidas para la técnica: Partición de Equivalencia.

3.4. Conclusiones

En este capítulo se mostraron y explicaron detalladamente secciones del código fuente de la biblioteca de clases, escogiendo aquellos segmentos de código que resultan significativos en el cumplimiento de los objetivos del trabajo de diploma. Luego se realizaron pruebas de unidad a la biblioteca de clases, aplicando para ello el método de caja blanca, empleando la técnica camino básico. Se efectuaron además, pruebas de sistema, enfocadas a comprobar el correcto funcionamiento de la biblioteca de clases, valiéndose del método de caja negra utilizando la técnica de partición de equivalencia. La realización de estas pruebas permitió detectar errores que fueron corregidos, validando la calidad de la implementación realizada.

CONCLUSIONES GENERALES

Con la elaboración del presente trabajo de diploma se implementó una biblioteca de clases que estandariza la Estimación de Parámetros y el Análisis de Sensibilidad en SED ordinarias, dándole solución al problema de investigación planteado:

- Con el estudio y desarrollo de la fundamentación teórica se realizó una propuesta de biblioteca de clases para la estandarización de la Estimación de Parámetros y el Análisis de Sensibilidad en SED ordinarias.
- Se realizó el diseño de la biblioteca de clases para la estandarización de la Estimación de Parámetros y Análisis de Sensibilidad en SED ordinarias.
- Se desarrolló una biblioteca de clases que integra los algoritmos seleccionados atendiendo a los requisitos funcionales definidos.
- Se validó el correcto funcionamiento de la biblioteca de clases a través de las pruebas realizadas.

RECOMENDACIONES

- Integrar la biblioteca de clases a BioSyS 1.1.
- Incorporar a la biblioteca de clases la funcionalidad de definir nuevas funciones objetivos.
- Valorar la inclusión en la biblioteca de clases de nuevos algoritmos de optimización.
- Incorporar a la biblioteca de clases funciones estadísticas para interpretar el paquete de resultados obtenidos luego de realizar el Análisis de Sensibilidad a los parámetros especificados.

TRABAJOS CITADOS

1. **Rolando Alfredo Hernández León, Sayda Coello González.** *Proceso de Investigaciión Científica.* Ciudad de La Habana, Cuba : Editorial Universitaria del Ministerio de Educación Superior, 2011. Vol. 2. ISBN 978-959-16-1307-3.

2. **López, Marta, y otros.** *Biología de Sistemas, Informe de Vigilancia Tecnológica.* España, Madrid : Cintia Refojo (Genoma España), Septiembre 2007. ISBN: 84-609-9762-6.

3. **Ricardo Paxson, Kristen Zanella.** *Studying the World's Most Complex Dynamic Systems.* s.l. : The Math Works News & Notes, 2007. ISBN: 91483V00..

4. **Acosta., Jaime Escobar.** *Ecuaciones Diferenciales con aplicaciones en MAPLE.* Antioquia : s.n., 2004. Vol. Capítulo 1.

5. **Cavalcanti, Pacheco Marco Aurélio.** *Algoritmos Genéticos. Principios y Aplicaciones.* Río de Janeiro : s.n., 2002.

6. *Experimental Design in Systems Biology based on Parameter Sensitivity Analysis using Monte Carlos Method.* **Cho, Kwang-Hyun, y otros.** 11-12, s.l. : The Society for Modeling and Simulation International, 2003, Vol. 79.

7. **Goldberg, David.** *Genetic Algorithms in Search, Optimization and Machine Learning.* 1989.

8. *Algoritmos Basados en Cúmulos de Partículas para la Resolución de Problemas Complejos.* **García, José Manuel.** 2006, pág. 14.

9. *Handbook of Metaheuristics.* **Glover, F y Kochenberger, G.** Norwell : Kluwer Academic Publishers, 2002.

10. —. **Crainic, T y Toulouse, M.** s.l. : Kluwer Academic Publishers, 2003, Vol. Parallel Strategies for Metaheuristics, págs. 475-513.

11. **Rasúa, Rafael Arturo Trujillo.** *Tesis Doctoral: Algoritmos paralelos para la solución de problemas de optimización discretos aplicados a la decodificación de señales.* España : Departamento de Sistemas Informáticos y Computación., 2009. págs. 60-64.

12. **Kolda, T. G., Lewis, R. M. y Torczon, V.** *Optimization by Direct Search: New perspective on Some Clasical and Modern Methods.* s.l. : SIAM Review, 2003.

13. **Engelbrecht, A.P.** *Fundamentals of Computational Swarm Intelligence.* s.l. : John Wiley & Sons, 2006.

14. **Álvarez, Nelson Guerra y Labrín, Broderick Crawford.** *Optimización de funciones a través de Optimización por Enjambre de Partículas y Algoritmos Genéticos.* Valparaíso, Chile : Escuela de Ingeniería Informática.

15. **Kennedy, Jammes y Eberhart, R.C.** *Particle Swarm Optimization.* 1995.

16. **Shi, Y. y Eberhart.** *Empirical Study of Particle Swarm Optimization.In Proceedings of the 1999 IEEE Congress on Evolutionary Computation.* 1999.

17. **Clerc, M. y Kennedy, J.** *The particle swarm: Explosion, stability, and convergence in a multidimensional complex space.* 2002.

18. **López, Jesús Ramón Pérez.** *Contribución a los métodos de optimización basados en procesos naturales y su aplicación a la medida de antenas en campo próximo.* 2005.

19. **Matías, Ison, Jacobo, Sitt y Marcos, Trevisan.** *Algoritmos Genéticos: aplicación en MATLAB.* 2005.

20. **Guervós y Merelo, Juan Julián.** *Informática evolutiva: Algoritmos Genéticos.* 2007.

21. *Algoritmos evolutivos en la solución de problemas de estimación de parámetros.* **Severo, Aymée de los Ángeles Marrero.** La Habana, Cuba : s.n., 2006, págs. 142,144. ISSN: 1409-2433.

22. Wordpress: JGAP Default Initialisation Configuration. [En línea] Mathew Hall, 18 de febrero de 2013. [Citado el: 25 de abril de 2013.] http://mathewjhall.wordpress.com/2013/02/18/jgap-default-initialisation/.

23. *Oxford Journals"BioJava: an open-source framework for bioinformatics in 2012".* [En línea] Oxford Journals, 24 de enero de 2013. [Citado el: 5 de febrero de 2013.] http://bioinformatics.oxfordjournals.org/cgi/pmidlookup?view=long&pmid=22877863.

24. Oxford Journals. [En línea] [Citado el: 12 de enero de 2013.] http://bioinformatics.oxfordjournals.org/content/20/8/1319.short.

25. Oxford Journals. [En línea] 27 de octubre de 2007. [Citado el: 24 de enero de 2013.] http://services.oxfordjournals.org/cgi/tslogin?url=http%3A%2F%2Fwww.oxfordjournals.org.

26. *Oxford Journals SBML-PET: a Systems Biology Markup Language-based parameter estimation tool.* [En línea] 27 de octubre de 2006. [Citado el: 24 de enero de 2013.] http://www.oxfordjournals.org/subject/mathematics/.

27. Java Genetic Algorithms Package. [En línea] 2012. [Citado el: 24 de enero de 2013.] http://jgap.sourceforge.net/.

28. **UCI.** Albet S Ingeniería y Sistema. [En línea] 2005. [Citado el: 27 de mayo de 2013.] http://www.albet.cu/lineas/salud-y-ciencias/alas-biosys.

29. **Quesada, Juan Antonio López.** Fundamentos de Ingeniería del Software. Mérida : s.n., 2007, Tema 7. Métodos de desarrollo de SW.

30. **José Esteva, Lianet.** *Introducción a OpenUP/Basic.* [En línea] Teleinformatics Technology Group, 2009. [Citado el: 2 de febrero de 2013.] http://epf.eclipse.org/wikis/openupsp/openup_basic/customcategories/introduction_to_openup_basic,_BTJ_YMXwEduywMSzPTUUwA.html.

31. Introduction to OpenUP (Open Unified Process). [En línea] [Citado el: 14 de enero de 2013.] http://www.eclipse.org/epf/general/OpenUP.pdf.

32. **Booch, Rumbaugh y Jacobson.** *El Lenguaje Unificado de Modelado.* 2007. Vol. Cap 1 y 2.

33. Herramientas CASE. [En línea] [Citado el: 25 de enero de 2013.] www.itescam.edu.mx/principal/sylabus/fpdb/recursos/r88866.DOC.

34. Visual Paradigm for UML. [En línea] [Citado el: 25 de enero de 2013.] http://www.visual-paradigm.com/product/vpuml/.

35. Lenguajes de Programación. [En línea] [Citado el: 8 de Diciembre de 2012.] http://www.lenguajes-de-programacion.com/lenguajes-de-programacion.shtml.

36. Java. [En línea] [Citado el: 6 de febrero de 2013.] http://www.java.com/.

37. ETSI. Escuela Técnica Superiorde Ingeniería. [En línea] 2012. [Citado el: 6 de febrero de 2013.] http://www.etse.urv.es/ri/manuals/java/JavaTut/Intro/carac.html.

38. NetBeans IDE The Smarter and Faster Way to Code. [En línea] 2012. [Citado el: 6 de febrero de 2013.] http://netbeans.org/.

39. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, Inc, 1999. págs. 11. 40,193 - 221. ISBN: 970-17-0261-1.

40. **Young, Ralph R.** *The Requirements Engineering Handbook.* s.l. : ARTECH HOUSE, INC., 2004. págs. 1, 51.

41. **Presman, Roger S.** *Ingeniería de Software Un Enfoque Práctico.* s.l. : McGraw-Hill, 2010. ISBN 978-0-07-337Z.

42. **Reynoso, Carlos Billy.** *Introducción a la Arquitectura de Software.* Buenos Aires : UNIVERSIDAD DE BUENOS AIRES, Versión 1.0 marzo 2004.

43. **Larman, Craig.** *UML y Patrones. Introducción al análisis y diseño orientado a objetos.* México : Prentice Hall, Inc, 1999. ISBN: 970-17-0261-1.

44. **Nava, Mayli.** *Arquitectura de Software Sistemas de Llamada y Retorno.* Caracas, Venezuela. : s.n., 2002.

45. **Hern, Marcello Visconti.** *Fundamentos de Ingeniería de Software.* 2002.

46. **Bruegge, Bernd y Dutoit, Allen.** *Ingeniería de software orientado a objetos.* s.l. : Prentice Hall - Pearson Educación, 2002. Capítulos 9 Páginas 326-369.

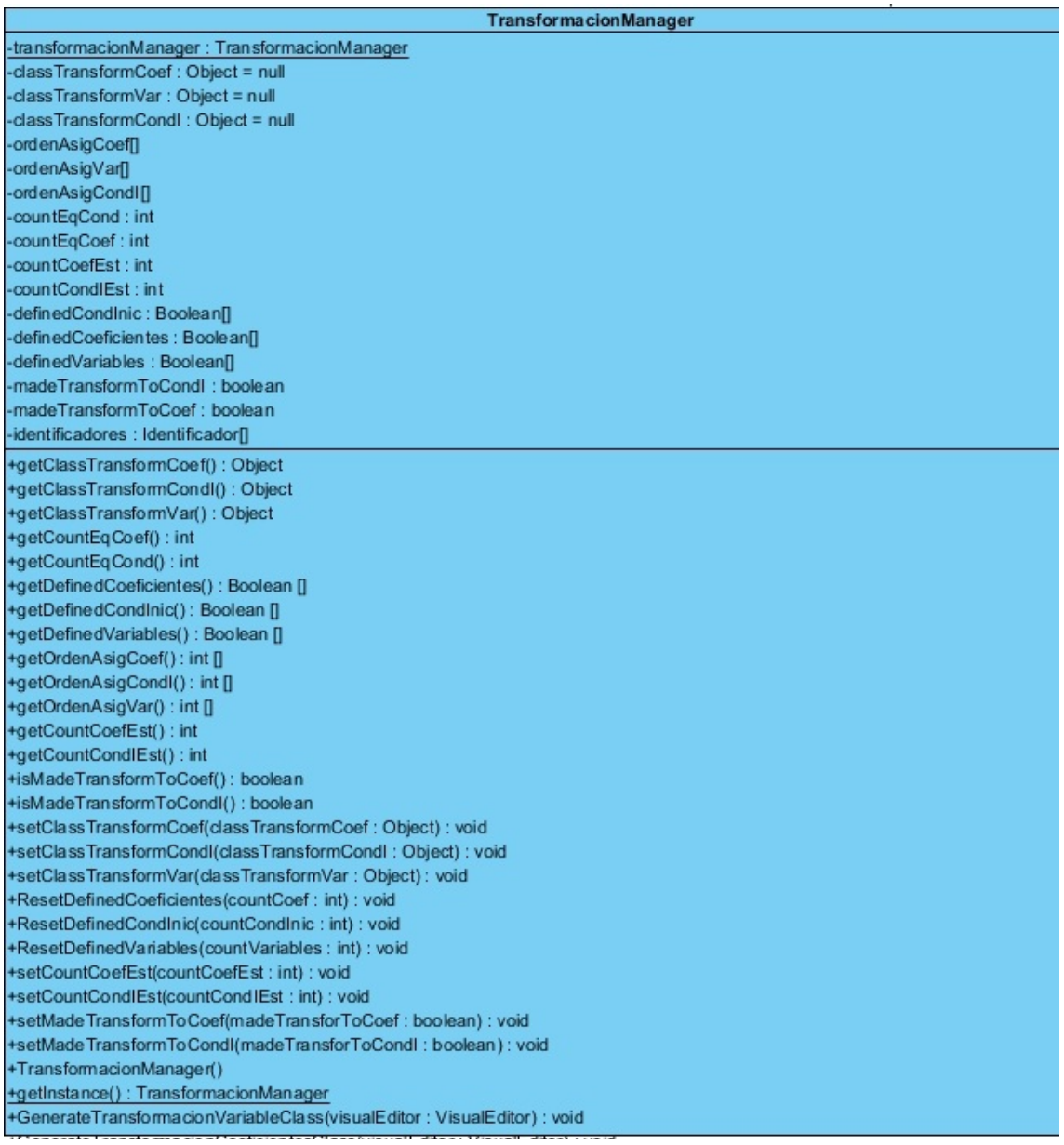
47. **Pérez, P Mariñán.** *Patrones de Diseño (Design Patterns).*

48. **Granma, E.V.A. UCI Facultad Regional.** *Conferencia #6. Fase de Construcción. Flujo de Trabajo de Implementación. Modelo de implementación, ISW II.* s.l. : Granma, E.V.A. UCI Facultad Regional, 2012.

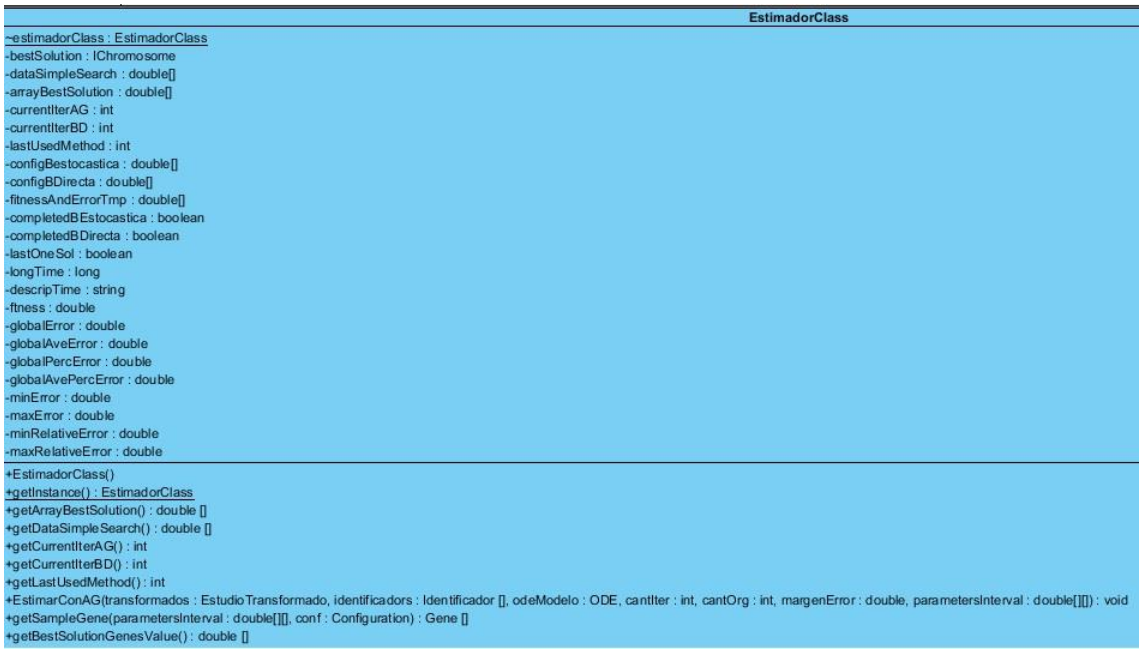
49. **Slezak, Diego Fernández.** *Estimación de parámetros en modelos biológicos complejos..Aplicación a modelos de crecimiento tumoral.* Buenos Aires, Argentina : s.n., 2010.

ANEXOS

ANEXOS 1: Diagrama UML: Clase CentralControlManager.

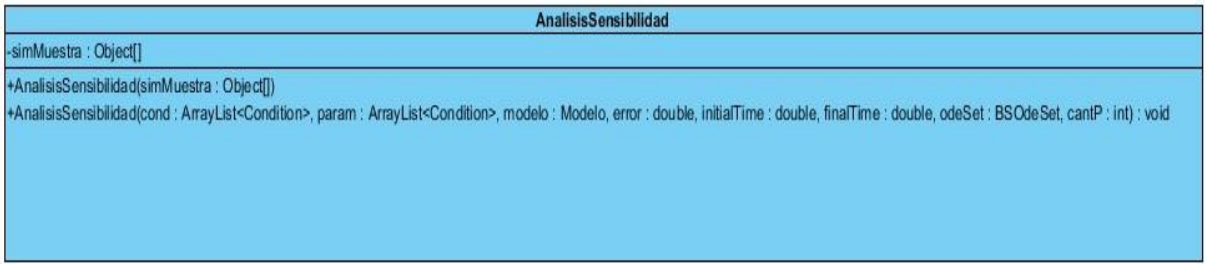


ANEXOS 2: Diagrama UML: Clase TransformacionManager.



ANEXOS 3: Diagrama UML: Clase EstimadorClass.

ANEXOS 4: Diagrama UML: Clase EstudioTransformado.



ANEXOS 5: Diagrama UML: Clase AnalisisSensibilidad.

Algoritmo para estimar los valores de los parámetros especificados usando Optimización por Enjambre de Partículas(PSO)

```
public void EstimarConPSO(EstudioTransformado transformados,
            Identificador[] identificadors, double[][] parametersInterval,
ODE odeModelo, int iterCount, int particleCount, double errorBound) throws
Exception, Throwable {

        //cargando la configuración desde un archivo xml

        PSOConfiguration config =
PSOConfigurationFactory.getConfiguration("/pso-xml-config.xml",
"configuration");

        //Modificando las condiciones de parada

        config.getStoppingConditions().clear();

        /*condición de parada regida por las cantidad máxima de iteraciones*/

        config.getStoppingConditions().add(new
MaximumIterations<PSOAlgorithm>(iterCount));

        /*condición de parada regida por el margen de error*/

        config.getStoppingConditions().add(new
StoppingConditionErrorBound(transformados, identificadors, odeModelo,
errorBound));

        /*definiendo el tamaño del enjambre de partículas*/

        config.setSwarmSize(particleCount);

        /*definiendo la dimensión de las partículas del enjambre, esta
dimensión es igual a la cantidad de parámetros que se estimarán*/

        config.setDimention(identificadors.length);

        /*definiendo el rango de valores para la posición de cada partícula*/
```

```
        config.setPositionMin(new PSOParameter(parametersInterval[0][0]));
        config.setPositionMax(new PSOParameter(parametersInterval[0][1]));

        configPSO = new double[]{iterCount, particleCount, errorBound};

        /* definiendo la función objetivo a optimizar*/

        TestFitnessFunction ffe = new TestFitnessFunction(transformados,
identificadors, odeModelo);

        /*creando el problema de optimización, especificando que la
optimización de la función objetivo viene dada por la tendencia a su
minimización */

        PSOFunctionBenchmark problem = new PSOFunctionBenchmark(ffe, new
MinimisationFitnessEvaluator<StandardParticle>(), config);
        PSOAlgorithm psoAlgorithm = new PSOAlgorithm(problem);

        // ejecutando el algoritmo

        psoAlgorithm.run();

        /* Obteniendo la partícula más apta*/

        PSOParticle mejorSol = psoAlgorithm.getBestSolution().getParticle();
        arrayBestSolution = getBestSolutionParticleValue(mejorSol);

        /*calculando el fitness y errores que genera la mejor solución
obtenida*/

        dataSimpleSearch =
Calculator.CalculateCuadraticDiffAndGlobalErrorsChromosome(arrayBestSolution,
transformados, identificadors, odeModelo);

    }
```

ANEXOS 6: Método EstimarConPSO de la clase EstimadorClass.

Algoritmo para calcular la diferencia entre los valores de las simulaciones realizadas y un patrón de simulaciones en el Análisis de Sensibilidad.

```
        private double comparar(Object[] resSimulacion, Object[]
                        resSimulacion1, int cantVar) {
                    double dif = 0, diferenciaE = 0;
  int total = Math.min(resSimulacion.length, resSimulacion.length /
                                cantVar), i = 0;
                            while (i < total) {
   dif = ((Resultado) resSimulacion1[i]).getValor() - ((Resultado)
```

```
resSimulacion[i]).getValor();
  diferenciaE += dif * dif;
            i++;
              }
     return diferenciaE;
              }
```

ANEXOS 7: Método Comparar de la Clase AnalisisSensibilidad.