Universidad de las Ciencias Informáticas

Facultad 6

TÍTULO: "PREDICCIÓN DE ACTIVIDAD ANTICANCERÍGENA DE COMPUESTOS ORGÁNICOS PARTIENDO DE DESCRIPTORES, UTILIZANDO PROGRAMACIÓN GENÉTICA."

Trabajo de diploma para optar por el título de
Ingeniero en Ciencias Informáticas.

Autores:
Lien Costales Leiva
Asnay Guirola González

Tutores:
Dr. Ramón Carrasco Velar
MSc. Aurelio Antelo Collado

Julio, 2007
"Año 49 de la Revolución"

“La ciencia es la verdadera escuela moral; ella enseña al hombre el amor y el respeto a la verdad, sin el cual toda esperanza es quimérica”

Berthelot

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de Ciencias Informáticas y al Centro de Química Farmacéutica los derechos patrimoniales de la misma con carácter exclusivo.

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

Lien Costales Leiva

_____

Firma del autor

Asnay Guirola Gonzáles

_____

Firma del autor

MSc. Aurelio Antelo Collado

_____

Firma del tutor

Dr. Ramón Carrasco Velar

_____

Firma del tutor

AGRADECIMIENTOS

A nuestros tutores Aurelio y Carrasco, por su asesoramiento científico, por su disposición permanente e incondicional en aclarar nuestras dudas, por su paciencia, esmero y ayuda.

A todos aquellos que de una forma u otra han hecho posible el desarrollo de la Tesis en especial a Lesly, Liesner y Daniel Gálvez por la infinita ayuda, no tenemos palabras para agradecerles su colaboración sin la cual no fuera posible el desarrollo del trabajo.

A nuestros amigos, por permitirnos disfrutar al máximo estos maravillosos cinco años a su lado y por ser partícipes de nuestros logros, desconciertos, malos y buenos momentos.

A nuestros padres, por su constante apoyo, comprensión y soporte durante todos estos años; por su ejemplo de superación incansable; por lo que hemos sido y seremos gracias a ustedes, son los mejores padres del mundo.

A Yanet, mi querida novia, por su paciencia y su amor infinito.

A la Revolución Cubana, por darnos la posibilidad de superarnos.

Un agradecimiento especial a nuestro Comandante por habernos dado la oportunidad de formarnos como ingenieros en una Universidad de nuevo tipo, que ha marcado nuestro pensamiento revolucionario y profesional.

DEDICATORIA

“A nuestros padres”

RESUMEN

Esta investigación surge en el marco de trabajo del Proyecto: “Plataforma inteligente para la predicción de de actividad biológica de compuestos orgánicos”, desarrollado conjuntamente por el Centro de Química Farmacéutica (CQF) y la Facultad de Bioinformática de la UCI. La plataforma cuenta con varios módulos que se han desarrollado de forma independiente, entre ellos el módulo de Inteligencia Artificial (IA). Este módulo es el encargado de predecir la actividad biológica de compuestos orgánicos a través de varias técnicas de Inteligencia Artificial e ir enriqueciendo la base de conocimientos del proyecto, para contribuir a la disminución de los costos en la investigación y el desarrollo de nuevos fármacos.

Se analizó, diseñó e implementó una aplicación para la predicción de actividad anticancerígena de compuestos orgánicos a partir de descriptores topológicos e híbridos como forma de describir una molécula, utilizando la Programación Genética como técnica de Inteligencia Artificial. La aplicación se implementó como plug-in para su incorporación al visualizador agregándole la funcionalidad de obtener modelos matemáticos mediante la programación genética. Se obtuvieron varios modelos de una muestra de entrenamiento de 2000 y una de prueba de 500 compuestos clasificados en activos e inactivos, obtenidos todos del ensayo NCI Yeast Anticancer Drug Screen; los modelos desarrollados tienen entre un 67 y un 70 por ciento de acierto en su clasificación.

PALABRAS CLAVE

Programación genética, predicción, descriptores, anticancerígeno, actividad biológica.

TABLA DE CONTENIDOS

INTRODUCCIÓN

La historia de los medicamentos es parte del devenir del hombre y de la historia de la medicina. Desde siempre, el ser humano buscó una explicación a los fenómenos y una solución a sus males. La aplicación de remedios para sanar o, al menos, para aliviar el sufrimiento, es tan antiguo como la humanidad.

En la actualidad, para el desarrollo de un nuevo medicamento desde su obtención en el laboratorio hasta su uso en terapéutica puede llegar a alcanzar un costo de 360-500 millones de USD y en un periodo promedio de 12-15 años, donde un bajo por ciento de las moléculas que llegan a fase de ensayo clínico terminan siendo comercializadas. Para optimizar recursos y tiempo se han apoyado en todos los adelantos tecnológicos y computacionales; esta situación evidencia la necesidad de encontrar y realizar métodos rápidos, eficientes y de bajo costo computacional para optimizar el proceso de obtención de nuevos fármacos. [1]

Una de las principales causa de muerte a nivel mundial hoy en día es el cáncer, del cual se dispone de abundante información con respecto a entidades químicas evaluadas en diferentes ensayos. Estas razones la convierten en un blanco interesante y muy útil, desde el punto de vista informático para el establecimiento de modelos que permitan predecir relaciones entre la estructura química y la actividad biológica.

El diseño cuantitativo de fármacos nació formalmente en el año 1964 con el trabajo de Hansch y Fujita. Sin embargo, en el mundo de los químicos se venía trabajando desde mucho antes, tratando de establecer relaciones cuantitativas entre la estructura química y determinadas propiedades de las moléculas. Estas propiedades podían ser químico-físicas, biológicas o ambas. Una necesidad estratégica de los investigadores que antecedieron a las actuales generaciones de químicos fue la búsqueda de descriptores y métodos que permitieran establecer las relaciones anteriormente mencionadas.

Los métodos que relacionan la estructura química con la actividad biológica pueden dividirse en dos grandes categorías (Sheridan, 1987): métodos topológicos/estadísticos y métodos de modelado molecular. En la aproximación topológica sólo se tiene en cuenta la estructura química "plana" de la molécula y se utilizan técnicas estadísticas o de reconocimiento de patrones para encontrar las correlaciones entre la estructura química de una serie de compuestos y su actividad biológica. De ahí surgieron las famosas siglas QSAR, acrónimo de Quantitative Structure-Activity Relationships. En los métodos de modelado se consideran las propiedades de las moléculas en tres dimensiones y son

importantes, entre otros, la mecánica cuántica, los campos de fuerzas, la termodinámica estadística, y los gráficos moleculares interactivos. Estos últimos permiten la representación y manipulación de las moléculas en tres dimensiones, lo que proporciona una información espacial. [2]

El presente trabajo tiene su origen dentro del proyecto de investigación conjunta del Centro de Química Farmacéutica y la Facultad de Bioinformática de la Universidad de las Ciencias Informáticas denominado *Plataforma inteligente para la predicción de actividad biológica de compuestos orgánicos* con el cual se persigue disponer de una herramienta para el tamizaje virtual de moléculas pequeñas en la actividad anticancerígena.

Por todos los elementos antes expuestos surge el siguiente ***problema*** ¿Cómo predecir la actividad anticancerígena de compuestos orgánicos empleando una plataforma inteligente?

El principal ***aporte práctico*** esperado con la realización de este trabajo es brindar una herramienta de trabajo capaz de predecir actividad anticancerígena de compuestos orgánicos a partir del cálculo de descriptores topológicos e híbridos. Donde el ***objeto de estudio*** es la Inteligencia Artificial aplicada a la predicción de actividades biológicas.

La programación genética aplicada al estudio de la relación estructura-actividad anticancerígena de compuestos orgánicos utilizando el cálculo de descriptores topológicos e híbridos será el ***campo de acción***.

Para dar solución al problema planteado se trazó como ***objetivo general*** desarrollar un módulo para la plataforma capaz de predecir actividad anticancerígena de compuestos orgánicos a partir de descriptores topológicos e híbridos, utilizando programación genética como técnica de inteligencia artificial.

Para cumplir el objetivo general de la investigación se trazaron los ***objetivos específicos*** que exponemos a continuación:

- Analizar un módulo basado en la programación genética que permita predecir relaciones entre la estructura y la actividad biológica.
- Diseñar el módulo analizado.
- Implementar el módulo diseñado.
- Validar el modelo implementado.

Para llegar a los objetivos anteriores se determinó desarrollar las siguientes tareas:

- Revisión del estado del arte acerca de sistemas de predicción existentes en el mundo.

- Selección de la técnica de inteligencia artificial óptima para el desarrollo de un módulo que permita obtener el modelo matemático.
- Elaboración del modelo de dominio correspondiente a un Sistema de Predicción.
- Definición de los requisitos funcionales y no funcionales de la herramienta.
- Desarrollo del diagrama de casos de uso del sistema.
- Descripción de los casos de uso correspondientes al sistema.
- Desarrollo del diagrama de clases del análisis.
- Desarrollo del diagrama de clases del diseño.
- Implementación del algoritmo para crear los modelos matemáticos.
- Implementación del algoritmo para realizar la predicción basada en el modelo creado.
- Validación del modelo desarrollado, comparando los valores obtenidos con valores experimentales.

La tesis esta estructurada en Resumen, Introducción, 4 capítulos como cuerpo fundamental de la tesis, Conclusiones, Recomendaciones, Referencias bibliográficas y Bibliografía. Los capítulos corresponden a diferentes aspectos temáticos:

- **Capítulo 1: Fundamentación Teórica.**

En este capítulo se realiza una breve explicación del estudio realizado sobre los métodos existentes para el diseño racional de fármacos asistidos por computadora y las técnicas de predicción actuales. Se muestran una serie de aplicaciones que están vinculadas a la predicción de la relación estructura-actividad y se describen las tendencias y tecnologías actuales a tener en cuenta para implementar la herramienta.

- **Capítulo 2: Características del sistema.**

Aborda lo referente al funcionamiento del negocio. Se desarrolla un modelo de dominio donde analizamos cada uno de los conceptos y entidades presentes en el negocio. Se plantean cada uno de los requisitos que debe tener la herramienta, se presenta el diagrama de casos de usos del sistema así como la descripción textual de cada uno de ellos.

- **Capítulo 3: Descripción de la Solución Propuesta.**

En el mismo se exponen los diagramas de clases por cada caso de uso, correspondientes a las etapas de análisis y diseño del sistema y el modelo de despliegue. Se muestra el diagrama de clases persistentes y se enumeran los principios de diseño para la implementación del sistema.

- **Capítulo 4: Implementación y Validación del Modelo.**

En el presente capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, para esto se muestra el diagrama de componentes. Además se validan los modelos matemáticos generados, mediante técnicas estadísticas, comparando los valores obtenidos con valores experimentales, con el objetivo de determinar el grado de precisión con que predicen los mismos.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

Los estudios de relación estructura-actividad constituyen una especialidad multidisciplinaria pues en ella convergen aspectos de descripción de la estructura química, generación, almacenamiento y recuperación de datos estructurales, de propiedades químico-físicas y biológicas. Otro aspecto importante que es necesario conocer son las diferentes técnicas matemáticas para el procesamiento de dichos datos lo cual constituye la propuesta de solución al problema que se plantea en este documento.

1.1 Búsqueda de nuevos fármacos

La predicción de actividad biológica de compuestos orgánicos posee un papel fundamental para la síntesis de nuevos fármacos. En estos momentos, el desarrollo de la industria farmacéutica internacional ha requerido del uso de herramientas que hagan cada vez más eficiente la búsqueda de nuevos fármacos para el tratamiento de las enfermedades. El método tradicional de búsqueda de nuevos principios activos basado en el sistema de “prueba y error” a través de ensayos masivos de gran número de sustancias químicas, es cada vez más ineficiente, siendo necesario ensayar sobre 10 000 compuestos para encontrar el deseado. A estos datos se unen que el costo del desarrollo de un nuevo medicamento desde su obtención en el laboratorio hasta su uso en terapéutica es de 360-500 millones de USD y en un periodo promedio de 12-15 años. Las vías alternativas en que la industria farmacéutica ha depositado su confianza, y también una gran cantidad de recursos, son la síntesis combinatoria y el diseño computacional de fármacos. Aunque el número de compuestos conocidos sobrepasa los 26 millones, y un gran número de estos está disponible en diferentes bases de datos químicas, muchos de ellos no han encontrado todavía aplicaciones farmacológicas o de otro tipo, lo cual es consecuencia de la diferencia que existe entre la velocidad a la cual las nuevas moléculas son obtenidas y el número de ellas que pueden ser evaluadas en ensayos farmacológicos, toxicológicos y farmacocinéticos. [1]

El descubrimiento de los fármacos esta sustentado en las propiedades de los compuestos que lo conforman y una característica determinante para los efectos deseados es la relación entre la estructura química y la actividad biológica de los mismos.

Uno de los propósitos más ambiciosos de la química moderna es encontrar esta relación entre la estructura molecular de productos orgánicos y la función biológica que cumplen. Los químicos trabajaron

en forma paralela con las líneas de investigación médica, sobre todo a partir de la década del 20, y han obtenido resultados trascendentales en el campo de las vitaminas, hormonas, proteínas, ácidos nucleícos y otros compuestos farmacológicos asociados a la vida.

1.2 El cálculo de descriptores como forma de describir la estructura química

Una forma de describir el comportamiento y la estructura que caracterizan a compuestos orgánicos, con un alto nivel de precisión, es el cálculo de descriptores. En la actualidad existen una gran cantidad de descriptores, cuyo número sobrepasa las centenas y están distribuidos según el tipo. Poseen varios enfoques y principalmente han sido empleados en el diseño de fármacos y en estudios de relación estructura-propiedad. Entre los tipos de descriptores más conocidos se pueden citar los derivados de mecánica molecular, los químico-cuánticos, los topológicos y los topográficos. Entre los métodos más comunes para el establecimiento de esas relaciones están los basados en la similitud-diferencia entre moléculas, los tridimensionales o 3D QSAR y los químico-físicos, entre otros. Todos estos métodos y descriptores pueden ser aplicables a moléculas, átomos y fragmentos en dependencia de la investigación que se este desarrollando y las preferencia o conocimientos de los especialistas. Con su empleo se genera una lista de resultados que permite caracterizar a las moléculas. El empleo de los estudios QSAR en investigaciones de las propiedades biológicas apoyadas en la teoría de grafos ha sido utilizada para la obtención de modelos aditivos y de regresión para la predicción de propiedades químicas, físicas y biológicas de forma efectiva. [1,3]

En el presente trabajo se emplean los descriptores topológicos e híbridos moleculares, los cuales fueron creados por químicos extranjeros y cubanos para describir la relación cuantitativa de Estructura – Actividad.

1.3 Descriptores

Se conocen como descriptores, los cuantificadores matemáticos que relacionan la estructura molecular y las propiedades físico-químicas de los compuestos a partir de parámetros estructurales simples. Los descriptores son utilizados para caracterizar la estructura química de un compuesto y la calidad de los mismos condiciona la calidad de los modelos matemáticos que describan los fenómenos biológicos.

Dentro de los descriptores más utilizados se encuentran los Índices topológicos que se basan únicamente en la estructura 2D o topología de la molécula, los cuales se derivan generalmente de las matrices de conectividad o de distancia del grafo molecular.

1.3.1 Descriptores Topológicos Moleculares.

Índice de Randic. [4]

El índice de conectividad de Randic ${}^{p}\chi$ se define por: ${}^{p}\chi = \Sigma[\delta(Vi)\ \delta(Vj)]^{-1/2}$. Donde el grado del vértice $\delta(Vi)$ es el número de vértices con los cuales el átomo i está enlazado directamente, y la suma es sobre todo los vértices adyacentes en la molécula.

Índice de Valencia

El índice de conectividad de Valencia ${}^{r}\chi$ se define por: $P_x = \Sigma[V_i\ V_j]$. Es una generalización del índice χ para la trayectoria de orden P. El índice fue modificado por Kier y Hall al definir subgrafos Gj de tipo árboles en el grafo de G que contiene los enlaces. A cada vértice i del grafo G se le asocia un término σ i (por ejemplo σ i = υ i). Después de esto, para cada subgrafo Gi con vértices j1..., jh+1 se calcula la magnitud:

$$F(\delta_{i_j}, \ldots, \delta_{j(h+l)}) = \prod \delta_{i_j}^{-1/2}$$

Donde. $\delta_i = Zv_i - h_i$ ó $\boldsymbol{\delta}_i = \dfrac{Z_i^v - h_i}{Z_i - Z_i^v - \mathbf{1}}$

Donde Zi es el número total de electrones, Zi^v el número de electrones de valencia y hi el número de átomos de hidrógeno enlazados al átomo i.

Índice de Partición de la Refractividad Molecular. [5]

Este índice emplea el algoritmo de Randic en su definición y su diferencia con este es que parte de la matriz del grafo químico completo. Es decir, considera para el cálculo los átomos de hidrógeno de la molécula y se define según la expresión:

$$^{P}MR_{x} = \sum[\delta^{MR}(v_i)\ \delta^{MR}(v_i)]^{-1/2} \qquad i \neq j$$

Índices Revisitados. [6]

Después de varios estudios y análisis experimentales realizados por Estrada, este sugirió hacer una sustitución en el grado de la ecuación para el cálculo del Índice de Randic de -1/2 a -1/3 para resolver el problema de la degeneración de los índices en el caso de los isómeros en familias de compuestos.

Índices Híbridos

Momentos Espectrales. [7,8]

Para obtener los valores de estos índices se parte de la matriz de conectividad de las aristas del grafo químico, se eleva a una potencia determinada y se calcula la suma de los elementos de la diagonal principal:

$$\mathbf{M_k = t_r(E^k)}$$

Como se puede apreciar existen varios descriptores y los resultados que se obtienen, con sus cálculos, brindan la posibilidad de emplearse en la obtención de modelos predictivos de la relación estructura-actividad. Por ello surge la tarea de seleccionar una técnica de inteligencia artificial para el desarrollo de un módulo que permita obtener un modelo matemático predictivo de la actividad biológica a partir de elementos estructurales de las sustancias.

1.4 Técnicas de Predicción

El proceso de predicción de clases de un objeto dado, se desarrolla para la clasificación del mismo, en donde el objeto de estudio del aprendizaje y la clasificación, es el descubrimiento de modelos y familias de funciones que permitan representar apropiadamente la pertenencia que tiene el objeto a una clase. Para

lograr este objetivo se realiza el desarrollo de algoritmos que permitan crear estos modelos a partir de datos.

En general los métodos de aprendizaje y clasificación son extensiones de técnicas clásicas tales como el análisis de discriminante, los métodos basados en árboles de decisión, los métodos de estimación de densidades o las técnicas jerárquicas de formación de grupos. Atendiendo a la naturaleza de los métodos de aprendizaje y clasificación se pueden organizar para su estudio en métodos estadísticos, modelos o algoritmos matemáticos para el reconocimiento de patrones, estrategias basadas en árboles de decisión y sistemas basados en el conocimiento. [9]

Dentro de los métodos estadísticos existe el discriminante lineal, logístico y cuadrático como extensiones del discriminante además de los cluster jerárquicos que pertenecen a técnicas de jerarquías de la formación de grupos. También dentro de la estadística moderna se encuentran la regresión adaptativa multivariada, redes causales y poli-árboles. Estos métodos estadísticos poseen un modelo de probabilidades como base que da lugar más que a una simple clasificación a la probabilidad de que un objeto pertenezca a cada una de las clases. [10]

Los sistemas basados en el conocimiento de la inteligencia artificial en los últimos años han experimentado un aumento en su aplicación y en la actualidad son muy empleados en sistemas de predicción de varias ramas científicas. Estos sistemas se caracterizan por tener una base de conocimiento que posee la información primaria de donde se hará un aprendizaje a través de la máquina de inferencia haciendo un análisis de la base de conocimiento. De aquí que la máquina de inferencia constituya el método de solución y un aspecto fundamental al igual que la base de conocimiento dentro del sistema. En los sistemas basados en el conocimiento están los sistemas basados en probabilidades, en reglas, en casos, en pesos y sistemas de inferencia borrosa entre otros. [11]

Dentro de la inteligencia artificial existen dos enfoques principales, el enfoque Simbólico o Top-down y el enfoque Subsimbólico o bottom-up. El enfoque Subsimbólico se caracteriza por crear sistemas con capacidad de aprendizaje. Éste se puede obtener a nivel de individuo imitando el cerebro y es lo que se conoce como Redes Neuronales, o a nivel de especie, imitando la evolución, lo que se ha denominado Computación Evolutiva, además de estas dos técnicas también esta la Lógica Difusa. [12]

Por la importancia en el desarrollo de este trabajo analizaremos la computación evolutiva. Esta constituye un método de búsqueda y optimización estocástica, ciega y múltiple de posibles soluciones a un problema determinado. Este proceso es guiado por los principios que rigen la evolución natural. Combina los conceptos de adaptación y supervivencia de los más aptos para producir soluciones subóptimas al problema dado. Esto es logrado mediante la aplicación de operadores de reproducción y cruce entre los individuos de una población que representa soluciones potenciales.

Existen varios modelos de computación evolutiva, los cuales reciben el nombre genérico de algoritmos evolutivos. Estos poseen como característica común su inspiración en la simulación de poblaciones de individuos y sobre el esquema general de los algoritmos evolutivos los principales modelos se clasifican en algoritmos genéticos, programación evolutiva, clasificadores genéticos, programación genética. [13]

La programación genética es un concepto más avanzado y más reciente que surge a comienzo de la década del noventa por estudios realizados por John Koza, sustentados sobre la base de los modelos de algoritmos genéticos y programación evolutiva. [14] Es entonces la disciplina de la computación evolutiva que provee un modo para realizar una búsqueda en el espacio de programas para encontrar aquel programa que sea más apto en la resolución de un problema dado, todo esto de modo independiente al dominio del problema. [15]

A diferencia de los algoritmos genéticos en la programación genética los individuos representan programas computacionales y las estructuras que sufren cambios adaptativos durante la programación genética son activas y capaces de ser ejecutadas en su forma actual. No son codificaciones pasivas a un problema, como lo son los strings de caracteres de longitud fija de los algoritmos genéticos. [15]

La programación genética no sólo se ha utilizado para generar programas, también se emplea en la búsqueda de cualquier tipo de soluciones cuya estructura sea similar a la de un programa. Por ejemplo, circuitos electrónicos, fórmulas matemáticas. Estas últimas estructuras son las más usadas en el desarrollo del algoritmo empleado para dar solución al problema en cuestión.

Principales Conceptos:

Para encontrar un modelo por programación genética se necesita definir una serie de elementos que son fundamentales para resolver el problema antes de entrar a la secuencia de pasos lógicos para el desarrollo del algoritmo. Estos son:

1. El conjunto de terminales: Los terminales son las hojas de los árboles que corresponden a variables o a valores constantes. Es decir, son las entradas de datos al programa. El conjunto de terminales (junto con el conjunto de funciones) son los ingredientes a partir de los cuales se construye un programa de computador para solucionar total, o parcialmente, un problema. En nuestro caso, el conjunto de terminales o datos de entrada son los descriptores moleculares, utilizados como método para describir la estructura química (Xs del problema).

2. El conjunto de funciones primitivas: Las funciones se usan junto con los terminales para generar la expresión matemática que trata de satisfacer la muestra finita de datos dados. En el conjunto de funciones primitivas pueden incluirse:

- Operaciones aritméticas (+, -, *, etc.).
- Funciones matemáticas (seno, coseno, log, etc.).
- Operadores boléanos (and, or, not, etc.).
- Operadores condicionales (if-then-else).
- Funciones que causen iteración (do-until).
- Funciones que causen recursión.
- Cualquier otro tipo de función específica del dominio que sea definida.

Un programa de computador (árbol de análisis gramatical) es una composición de funciones del conjunto F de funciones y terminales del conjunto T de terminales. Cada una de las funciones del conjunto F debe ser capaz de aceptar, como sus argumentos, cualquier valor y tipo de dato que pueda ser retornado por cualquier función del conjunto de funciones, y cualquier valor y tipo de dato que pueda tomar por cualquier terminal del conjunto T. Esto es, que el conjunto de funciones y el conjunto de terminales deben tener la propiedad de clausura. En este problema de predicción basta con utilizar funciones aritméticas y algunas funciones matemáticas.

3. La medida de la aptitud: Cada programa de computador individual en la población, se mide en términos de que tan bien se comporta en el ambiente del problema particular. Esta medida se llama medida de aptitud. La naturaleza de la medida de aptitud varía con el problema. Para muchos problemas, la aptitud de un programa se mide calculando el error que se produce al evaluar la función. El mejor programa de computador o función tendrá un error cercano a cero.

4. Los parámetros para controlar la corrida: Después de que se efectúan las operaciones genéticas sobre la población actual, la población de hijos reemplaza la generación anterior, a cada individuo de la nueva población se le mide su aptitud, y el proceso se repite por muchas generaciones. Es decir, la solución del problema consiste en un proceso combinatorio de optimización de funciones.

En cada etapa de este proceso secuencial controlado localmente, se determina la aptitud de cada uno de los individuos, es decir, se halla el ajuste de las funciones a los datos experimentales y se pondera esa calidad de ajuste. Los parámetros que controlan normalmente la ejecución de la PG son: el tamaño de la población; el número de generaciones; las probabilidades de aplicación de los operadores genéticos de cruce y mutación; y la estrategia de selección de los individuos para la nueva generación.

5. El método para seleccionar un resultado y el criterio para terminar la ejecución del programa: Un método posible de selección del resultado consiste en designar como tal al mejor individuo (función) que haya aparecido en cualquier generación, para lo cual es necesario colocarlo en memoria durante la ejecución. Otro método alternativo es el de seleccionar como resultado al mejor individuo de la población en el momento que termina la ejecución. La coincidencia entre ambos métodos ocurre con relativa frecuencia. Obviamente, no es necesario colocar al individuo en memoria en este último caso.

Un criterio de terminación para la programación genética es fijar un número máximo G de generaciones en una ejecución. Otro es cuando se alcance algún predicado de éxito específico del problema (como por ejemplo, encontrar una solución 100% correcta).

Posterior a la definición de los elementos que son fundamentales para resolver los problemas de PG pasamos a exponer los pasos lógicos para el desarrollo del algoritmo:

1. Generar una población inicial de composiciones aleatorias de funciones y terminales del problema (programas de computadora).
2. Ejecutar iterativamente los siguientes pasos hasta que se satisfaga el criterio de terminación:
    a) Ejecutar cada programa en la población y asignarle un valor de aptitud de acuerdo a la calidad en la solución del problema.

b) Crear una nueva población de programas de computadora aplicando las siguientes dos operaciones primarias. Las operaciones son aplicadas a los programas de computadora en base a cierto criterio de selección.

c) Copiar programas de computadora existentes en la nueva población.

d) Crear nuevos programas de computadora recombinando genéticamente partes aleatorias seleccionadas de dos programas ya existentes.

3. El mejor programa de computadora (función) que haya aparecido en cualquier generación es seleccionado como el resultado de la programación genética. Este resultado puede ser una solución perfecta o aproximada al problema. Como resultado se devolverá una función matemática de mejor ajuste, o mejor individuo. Este mejor individuo será el mejor modelo encontrado y cuando se requiera analizar una nueva molécula o una familia de compuestos, se desfragmentan, se les calculan los descriptores, luego se pasan por el modelo correspondiente que describe la actividad que se quiere analizar y como resultado se obtendrá el por ciento de ese tipo de actividad, que está presente en cada una de las moléculas analizadas.

1.4.1 Campos de aplicación de la programación genética [16]

Regresión simbólica

La regresión simbólica o identificación de funciones involucra encontrar una expresión matemática, en forma simbólica, que provea un ajuste bueno o perfecto entre una muestra finita de valores de variables independientes y los valores asociados de las variables dependientes. Por lo tanto, la regresión simbólica involucra encontrar un modelo que se ajuste a una muestra de datos dada. La regresión simbólica involucra encontrar tanto la forma funcional como así también los coeficientes numéricos del modelo. La regresión simbólica difiere de las regresiones convencionales lineal, cuadrática o polinómica, las cuales involucran meramente encontrar los coeficientes numéricos para una función cuya forma ya ha sido especificada. En cualquier caso, la expresión matemática que es buscada en la regresión simbólica puede ser vista como un programa de computadora que toma los valores de las variables independientes como sus entradas y produce los valores de las variables dependientes como sus salidas.

Descubrimiento empírico y pronóstico

El descubrimiento empírico involucra encontrar un modelo que relacione un conjunto dado de valores de variables independientes y los valores asociados (usualmente dispersos) de las variables dependientes para un determinado sistema del mundo real. Una vez que se haya encontrado un modelo para datos empíricos, el mismo puede ser utilizado para pronosticar los valores futuros de las variables del sistema. El modelo que es buscado en problemas de descubrimiento empírico puede ser visto como un programa de computadora que toma varios valores de las variables independientes como sus entradas y produce los valores observados de las variables dependientes como sus salidas.

Inducción de árboles de decisión

Un árbol de decisión es una manera de clasificar un objeto de un universo dado en una clase particular, utilizando sus atributos como base para dicha clasificación. La inducción de un árbol de decisión es, por lo tanto, un enfoque para la clasificación. Un árbol de decisión corresponde a un programa de computadora que consiste en funciones que prueban los atributos del objeto. La entrada de dicho programa consiste en valores de ciertos atributos asociados a un punto de datos dado. Su salida es la clase en la que se clasifica el punto de datos dado.

Estos son solo unos ejemplos de posibles aplicaciones de la programación genética existen muchos problemas de campos variados en los que la interrelación entre las variables sea desconocida y pueden ser reformulados como la búsqueda de un programa que produzca una determinada salida ante ciertas entradas. De aquí se demuestra que el espacio de búsqueda es el espacio de todos los posibles programas de computadora compuestos por funciones y terminales apropiados al dominio del problema. Las funciones pueden ser operaciones aritméticas, funciones matemáticas, funciones lógicas o funciones específicas del dominio. Dependiendo del problema en particular, el programa de computadora puede retornar un valor lógico, un valor entero, un valor real, un valor complejo, un vector, un valor simbólico o valores múltiples.

1.5 Software de Predicción [17]

Existen varios software de predicción de actividad biológica que utilizan diferentes métodos para lograr sus objetivos como el sistema experto APEX-3D implementado sobre Silicon Graphics que utiliza teoría lógica combinatoria, para el establecimiento de las relaciones estructura-actividad de grupos de compuestos que constituyen la base de datos suministrada por el usuario. Este sistema experto está

insertado dentro del paquete de programas Insight II y tuvo su predecesor en el sistema experto OREX implementado sobre IBM PC 80286. Este último se basa en la descomposición topológica de las moléculas en fragmentos estructurales y su asociación a las actividades biológicas reportadas en una base de datos interna de alrededor de 15 000 compuestos. OREX, de una manera rápida, permite con una probabilidad general del 75% predecir simultáneamente las posibles actividades biológicas que puede presentar la sustancia objeto de estudio.

El sistema experto PASS parte también de la fragmentación topológica de la estructura química para la predicción de actividad. En su versión 4.20 predice las probabilidades de presencia/ausencia de 114 actividades biológicas simultáneamente, a partir de un conjunto de entrenamiento de 9314 compuestos. Versiones en desarrollo incrementaron este número hasta alrededor de 70 000 sustancias biológicamente activas y algo más de 600 tipos de respuestas biológicas que cubren casi todo el espectro de la farmacología conocida hasta la fecha según el criterio de los autores. Este sistema fue mejorado con la incorporación de un diccionario de actividades (IBIAC) que permite identificar relaciones causales y de agrupación entre diferentes respuestas biológicas y mecanismos.

Otro sistema conocido es el ADAPT (Automated Data Analysis using Pattern Recognition Toolkit). Este es un sistema de programas que le permite al usuario el desarrollo de relaciones estructura-actividad y estructura-propiedad. Brinda al usuario la facilidad de entrada gráfica y almacenamiento de estructuras moleculares y sus datos asociados, generación de estructuras 3-D, cálculo de descriptores moleculares y análisis de estos empleando estadística multivariada, reconocimiento de patrones o redes de neuronas para construir modelos predictivos. Los enfoques estadísticos incluyen regresión lineal múltiple, análisis cluster, discriminante y redes neuronales. Se ejecuta sobre estaciones de trabajo bajo sistema operativo UNIX.

1.6 Importancia del Desarrollo del Software

Con el presente trabajo de investigación se logrará un gran impacto en la esfera científico- tecnológica ya que la aplicación de los resultados alcanzados permitirá introducir en la esfera médico-farmacéutica cubana una herramienta capaz de identificar y seleccionar los mejores candidatos a fármacos, en las etapas iníciales del proceso de desarrollo de principios activos, posibilitando un ahorro sustancial de

recursos humanos y materiales, los cuales son muy importantes dentro del desarrollo científico cubano actual.

1.7 Tecnologías y Tendencias Actuales

Uno de los aspectos fundamentales en la elaboración de un software es seleccionar las tecnologías y tendencias que mayores beneficios aporten. Para el desarrollo de este modulo se realizó un estudio de las posibles herramientas a utilizar en su construcción, teniéndose en cuenta la tendencia actual y las novedades de cada una de ellas.

Procesos de Desarrollo [18]

Existen varios procesos de desarrollo de software cada uno con sus propias características y particularidades aunque sus objetivos o propósitos son los mismos. Ejemplos de estos procesos son: RUP, XP, FDD.

RUP

El “Proceso Unificado de Desarrollo” es el resultado final de tres décadas de desarrollo y uso práctico. Esta es una de las causas que conlleva a que sea la metodología que mejor se ajusta a las necesidades que existen actualmente en el desarrollo de software, ya que propone una metodología iterativa e incremental que va eliminando los errores cometidos en las iteraciones previas, logrando que al final del proceso se obtenga como resultado un producto de calidad, muy acorde con la naturaleza cambiante de los requisitos en muchos proyectos.

RUP es un proceso capaz de ser aplicado a cualquier proyecto sin importar la magnitud del mismo, esta estructurado y permite adaptarse a cada proyecto, incluidos proyectos que no sean solo de software. Las fases por las que esta conformado son cuatros y se mencionan a continuación:

1. Inicio (puesta en marcha)
2. Elaboración (definición, análisis, diseño)
3. Construcción (implementación)
4. Transición (fin del proyecto y puesta en producción)

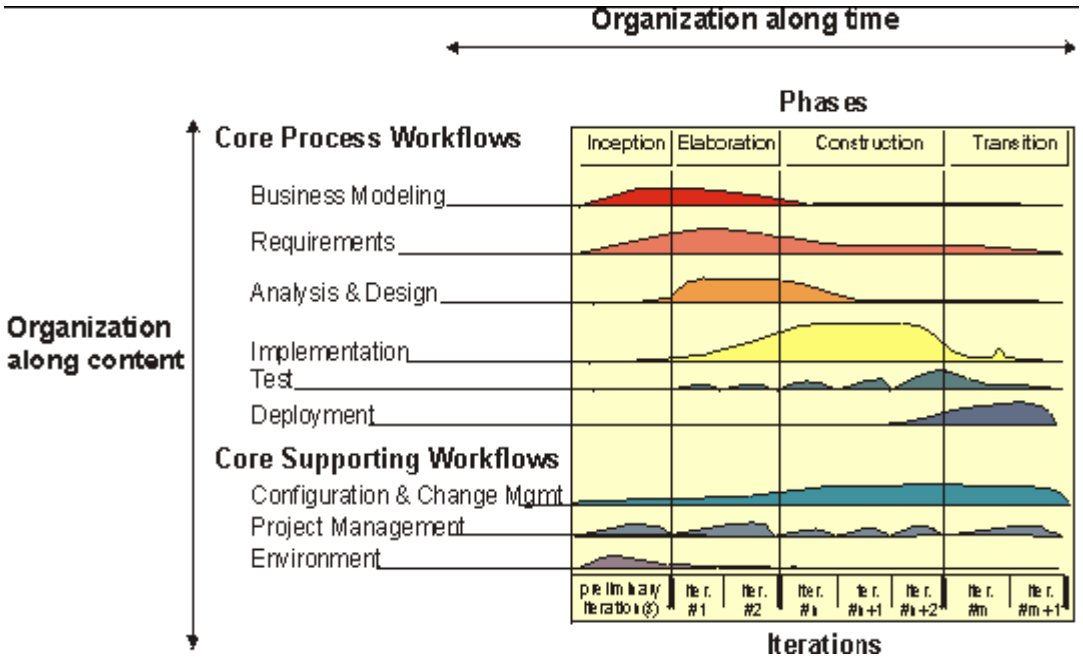


Figura 1.1 Vista general de RUP

RUP define nueve actividades a realizar en cada fase del proyecto

1. Modelado del negocio
2. Análisis de requisitos
3. Análisis y diseño
4. Implementación
5. Test
6. Distribución
7. Gestión de configuración y cambios
8. Gestión del proyecto
9. Gestión del entorno

RUP es dirigido por los casos de uso para describir lo que se espera del software y centrado en la Arquitectura del sistema, documentándose lo mejor posible, basándose en UML (Unified Modeling Language) como herramienta principal.

Resumen de puntos claves:

RUP

1. Pesado
2. Dividido en cuatro fases
3. Las fases se dividen en iteraciones
4. El discurrir del proyecto se define en Flujos de Trabajo
5. Los artefactos son el objetivo de cada actividad
6. Se basa en roles
7. UML
8. Muy organizativo
9. Mucha documentación

Por las características de RUP de ser un proceso muy organizativo, orientado a objetos y bien documentado permite ver de forma más descriptiva el proceso de desarrollo, por lo que se llega a la conclusión de que este proceso es el correcto y más indicado para llevar a cabo la descripción de este software.

Lenguaje de Modelado.

UML [4]

El “Lenguaje Unificado de Modelado” (UML) se presentó oficialmente cuando Rumbaugh, Booch y Jacobson unificaron sus estudios con una semántica y notación, para lograr compatibilidad en el análisis y diseño orientado a objetos, permitiendo que los proyectos se asentaran en un lenguaje de modelado maduro, permitiendo a los constructores de herramientas enfocarse en producir características más útiles. Es un lenguaje para visualizar, especificar, construir y documentar los artefactos de un sistema que involucra una gran cantidad de software.

La decisión de utilizar UML como notación para el desarrollo del software esta apoyada en que se ha convertido en un estándar con muchas características favorable como las siguientes:

1. Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
2. Permite especificar todas las decisiones de análisis y diseño, construyéndose así modelos precisos, no ambiguos y completos.
3. Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
4. Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
5. A pesar de tener gran expresividad esta notación es fácil de aprender.
6. UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

Herramienta CASE [19]

CASE (Computer-Aided Software Engineering) es una filosofía que se orienta a la mejor comprensión de los modelos de empresa, sus actividades y el desarrollo de los sistemas de información. Esta filosofía involucra además el uso de programas que permiten:

1. Construir los modelos que describen la empresa,
2. Describir el medio en el que se realizan las actividades,
3. Llevar a cabo la planificación,
4. El desarrollo del Sistema Informático, desde la planificación, pasando por el análisis y diseño de sistemas, hasta la generación del código de los programas y la documentación.

Los principales objetivos que poseen las CASE son:

1. Aumentar la productividad de las áreas de desarrollo y mantenimiento de los sistemas informáticos.
2. Mejorar la calidad del software desarrollado.
3. Reducir tiempos y costes de desarrollo y mantenimiento del software.
4. Mejorar la gestión y dominio sobre el proyecto en cuanto a su planificación, ejecución y control.

5. Mejorar el archivo de datos (enciclopedia) de conocimientos (know-how) y sus facilidades de uso, reduciendo la dependencia de analistas y programadores.
6. Automatizar:

- El desarrollo del software
- La documentación
- La generación del código
- El chequeo de errores
- La gestión del proyecto

7. Permitir

- La reutilización del software
- La portabilidad del software
- La estandarización de la documentación

8. Integrar las fases de desarrollo de la ingeniería del software con las herramientas CASE
9. Facilitar la utilización de las distintas metodologías que desarrollan la propia ingeniería del software.

Rational Rose [3]

Rational Rose es la herramienta de modelación visual que provee el modelado basado en UML. La Corporación Rational ofrece un Proceso Unificado (RUP) para el desarrollo de los proyectos de software. Para cada una de las etapas desde la Ingeniería de Requerimientos hasta la de pruebas existe una herramienta de ayuda en la administración de los proyectos, Rose es la herramienta del Rational para la etapa de análisis y diseño de sistemas.

Rose es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros de equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Una de las grandes ventajas de Rose es que utiliza la notación estándar en la arquitectura de software UML, la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común, además, los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

Visual Paradigm [20]

Visual Paradigm UML Community es una de las principales compañías de herramientas CASE. Tiene disponible distintas versiones: Enterprise, Professional, Standard, Modeler, Personal y Community la cual es gratuita. La compañía facilita licencias especiales para fines académicos.

Visual Paradigm es una herramienta que sirve para realizar modelado UML siguiendo el estándar UML. Esta herramienta posee características gráficas muy cómodas que facilitan la realización de los diagramas de modelado que sigue el estándar de UML que son:

1. Diagramas de clase, Casos de Uso, Comunicación, Secuencia, Estado,
2. Actividad, Componentes, etc.
3. Entre otras características importantes que se tienen tenemos:
4. Integración con diversas IDE's como son:
    - NetBeans(de Sun)
    - JDeveloper(de Oracle)
    - Eclipse (de IBM)
    - JBuilder (de Borland)
5. Ingenieria Inversa para:
    - JAVA
    - NET
    - XML
    - Hibernate
6. Exportación de imágenes jpg, png y svg (w3g estándar)

La herramienta está diseñada para una amplia gama de usuarios, incluyendo Ingenieros de Software, Analistas de Sistemas, Analistas de Negocios y Arquitectos de Sistemas que estén interesados en la creación de Grandes Sistemas de Software de manera confiable a través del paradigma Orientado a Objetos.

Visual Paradigm es la herramienta case que se utiliza en la modelación de este proyecto por todas las características que posee y beneficios que brinda.

Java [20]

Java es un lenguaje de programación con el que podemos realizar cualquier tipo de programa. En la actualidad es un lenguaje muy extendido y cada vez cobra más importancia tanto en el ámbito de Internet como en la informática en general. Está desarrollado por la compañía Sun Microsystems con gran dedicación y siempre enfocado a cubrir las necesidades tecnológicas más punteras.

Una de las principales características por las que Java se ha hecho muy famoso es que es un lenguaje independiente de la plataforma. Eso quiere decir que si hacemos un programa en Java podrá funcionar en cualquier ordenador del mercado. Es una ventaja significativa para los desarrolladores de software, pues antes tenían que hacer un programa para cada sistema operativo, por ejemplo Windows, Linux, Apple, etc. Esto lo consigue porque se ha creado una Máquina de Java para cada sistema que hace de puente entre el sistema operativo y el programa de Java y posibilita que este último se entienda perfectamente.

La independencia de plataforma es una de las razones por las que Java es interesante para Internet, ya que muchas personas deben tener acceso con ordenadores distintos. Pero no se queda ahí, Java está desarrollándose incluso para distintos tipos de dispositivos además del ordenador como móviles, agendas y en general para cualquier cosa que se le ocurra a la industria.

Java fue pensado originalmente para utilizarse en cualquier tipo de electrodoméstico pero la idea fracasó. Uno de los fundadores de Sun rescató la idea para utilizarla en el ámbito de Internet y convirtieron a Java en un lenguaje potente, seguro y universal gracias a que lo puede utilizar todo el mundo y es gratuito. Una de los primeros triunfos de Java fue que se integró en el navegador Netscape y permitía ejecutar programas dentro de una página Web, hasta entonces impensable con el HTML. Actualmente Java se utiliza en un amplio abanico de posibilidades y casi cualquier cosa que se puede hacer en cualquier lenguaje se puede hacer también en Java y muchas veces con grandes ventajas. Para lo que nos interesa a nosotros, con Java podemos programar páginas Web dinámicas, con accesos a bases de datos, utilizando XML, con cualquier tipo de conexión de red entre cualquier sistema. En general, cualquier aplicación que deseemos hacer con acceso a través web se puede hacer utilizando Java.

Hemos decidido usar este lenguaje para la realización de la aplicación por las ventajas de ser un lenguaje multiplataforma y un software libre, razón por la que aparecen gran cantidad de aplicaciones realizadas en dicho lenguaje actualmente que van desde móviles, la Web hasta aplicaciones de desktop.

Entornos de Desarrollo

NetBeans 5.5 [21]

NetBeans es un entorno de desarrollo integrado OpenSource y de distribución gratuita que proporciona herramientas muy cómodas y de fácil uso para el desarrollo de aplicaciones sobre la plataforma JAVA. Entonces se pueden desarrollar bajo esta IDE soluciones J2SE, J2ME y J2EE.

Entre las características que proporciona tenemos:

1. Desarrollo de aplicaciones multiplataforma sobre: MacOS, Windows, Linux.
2. Add-ons para desarrollo Móvil, desarrollo Web grafico, integración con SOA, optimización de aplicaciones y desarrollo con C y C++.
3. Cliente CVS integrado
4. Crecimiento de plataforma por medio de plugins. Entre los plugins que existen se tienen los siguientes:
   - Herramientas java que sirven para la mejora de desarrollo de aplicaciones
   - Herramientas de modelado UML
   - Herramientas XML, etc.
5. Struts, JSF, EJB, WebServices, etc.

Eclipse [20]

Eclipse es una plataforma de desarrollo extensible, basada en Java y de tipo open-source (CPL). Originalmente fue desarrollada por Alphaworks, laboratorio de desarrollo de IBM, pero actualmente está manejado por un consorcio de varias empresas, (Borland, Hitachi, etc.). Lo interesante de Eclipse, es que conforma una suerte de estándar o framework. Muchos vendedores de software basan sus ofertas de IDE en eclipse, al que suman conjuntos de plug-ins.

Este entorno de desarrollo integrado ofrece, el control del editor de código, del compilador y del depurador desde una única interfaz de usuario. Su misión consiste en evitar tareas repetitivas, facilitar la escritura de código correcto, disminuir el tiempo de depuración e incrementar la productividad del desarrollador. Estas tareas pueden realizarse de muchas maneras distintas: mediante la inclusión de asistentes para las tareas

más habituales y mecánicas, de editores que completen automáticamente el código y señalen los errores sintácticos, de gestores de archivos fuente, etc. Eclipse no es un IDE más a añadir a la lista, el objetivo de IBM ha sido crear una plataforma de desarrollo modular que cualquier herramienta de desarrollo pueda usar con cualquier lenguaje de programación.

Conclusiones

Después de un estudio detallado se puede llegar a la conclusión de que se desea desarrollar un módulo de predicción de la relación estructura-actividad anticancerígena de compuestos orgánicos a partir de descriptores topológicos e híbridos, como forma de describir la molécula.

Se ha escogido la programación genética como técnica de inteligencia artificial para desarrollar la predicción.

El lenguaje utilizado para el modelado fue UML y la herramienta CASE escogida es Visual Paradigm. Se ha escogido el lenguaje de programación Java por las posibilidades multiplataforma que este brinda, además por su posibilidad de uso gratuito. El editor seleccionado para la programación de este fue el Eclipse, por sus grandes potencialidades y sus posibilidades de extensión mediante el uso de plug-ins.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

En este capítulo se describe la solución propuesta utilizando los componentes del modelo de dominio de la metodología RUP, se tendrán en cuenta la definición de las entidades y los conceptos principales, así como su representación gráfica y definición de las reglas del negocio. Incluye la descripción de los requisitos funcionales, no funcionales del sistema, los actores que intervienen en el sistema, el diagrama de casos de uso del sistema y la descripción detallada de cada uno de los casos de uso del sistema.

2.1 Modelo del dominio

Un Modelo del Dominio captura los tipos de objetos más importantes en el contexto del sistema. Tiene como objetivo fundamental la comprensión y descripción de las clases más importantes en el sistema.

2.1.1 ¿Por que Modelo del dominio?

Debido a la poca estructuración de los procesos del negocio que tienen que ver con el objeto de estudio y para poder entender el contexto en que se emplaza el sistema necesitamos definir conceptos que podemos agrupar en un Modelo de Dominio.

2.1.2 Breve descripción del problema

El investigador del Centro de Química Farmacéutica desea analizar una familia de compuestos orgánicos para verificar el grado de actividad biológica de compuestos entrados por el investigador. Luego de estar definida la muestra que se desea analizar y escogida la actividad biológica se procede a evaluar los compuestos de dicha muestra en los modelos matemáticos que se encuentran en la base de conocimientos pertenecientes a la actividad biológica seleccionada por el investigador. Terminada la evaluación el sistema devuelve los compuestos con su por ciento de actividad biológica correspondiente ordenados de forma descendente. Si se desea realizar una nueva evaluación para otra actividad biológica con la misma muestra se selecciona una nueva actividad biológica y se realiza una nueva corrida.

2.1.3 Glosario de términos para el dominio

Compuestos Orgánicos: compuestos cuya composición fundamental es sobre la base del elemento químico carbono.

Actividad Biológica: actividad que caracteriza el comportamiento biológico en compuestos químicos.

Modelo Matemático: expresión numérica que simula la realidad.

Predicción: valor estimado que caracteriza una propiedad o fenómeno obtenido de un modelo.

Concentración Efectiva: cantidad de sustancia concentrada por unidades de masa o volumen.

Descriptores: es una forma de describir las moléculas mediante expresiones matemáticas que permite describir la estructura química o una propiedad de la molécula.

2.1.4 Diagrama Modelo del Dominio

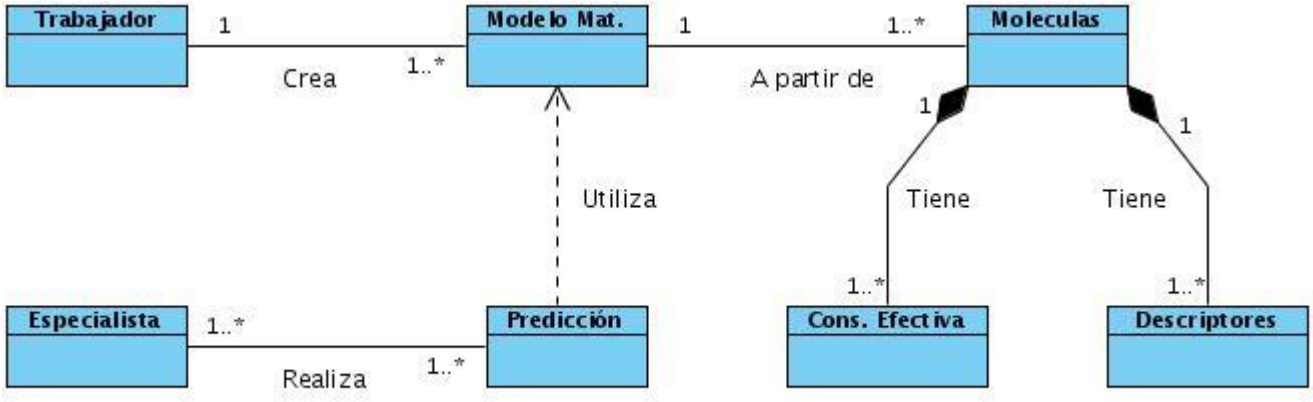


Figura 2.1 Modelo de dominio.

2.2 Especificación de los requerimientos de software

Requerimientos de software no son más que las condiciones o capacidades que tiene que tener un sistema para satisfacer un contrato o documento formal, se realizará un análisis del Modelo de Sistema, donde se verán cada uno de los casos de usos de este y la descripción de cada uno de ellos.

2.2.1. Requerimientos Funcionales

Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir, tareas que debe de realizar el sistema para satisfacer las necesidades del usuario. Definen los límites de la aplicación, proveen las bases sobre las cuales se planificará los contenidos de las iteraciones, permiten estimar el costo y el tiempo de desarrollo.

R1: Crear Modelo.

R2: Guardar los modelos en la base de conocimiento del sistema.

R3: Predecir Actividad Biológica.

R4: Cargar los ficheros de las moléculas.

R5: Cargar modelos matemáticos.

R6: Evaluar modelo matemático.

2.2.2. Requerimientos no Funcionales

Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener. Debe pensarse en estas propiedades como las características que hacen al producto atractivo, usable, rápido o confiable.

1. Requisito de funcionalidad

El sistema debe someterse a una etapa de adiestramiento en la que los usuarios se familiaricen con la aplicación y sean detectados los posibles errores, o puedan surgir posibles cambios en la interfaz para que los usuarios queden totalmente complacidos.

2. Apariencia o interfaz externa

El sistema debe contar con una interfaz amigable, fácil de comprender y usar, donde el usuario pueda orientarse fácilmente.

3. Usabilidad

El sistema le ofrecerá al investigador la posibilidad de realizar predicciones y analizar los resultados de las predicciones, en este sentido se centra el diseño de la aplicación y en específico de las interfaces que harán posible el intercambio de datos de manera que le resulte al usuario de fácil entendimiento.

4. Rendimiento

La eficiencia del producto estará determinada en gran medida por el aprovechamiento de los recursos que se disponen en el modelo Cliente/Servidor, y la velocidad de las consultas en la Base de Datos. La herramienta propuesta debe ser rápida y el tiempo de respuesta debe ser el mínimo posible, adecuado a la rapidez con que el cliente requiere la respuesta a su acción.

5. Soporte

Terminada la aplicación será sometida a pruebas con el objetivo de comprobar su funcionalidad.

6. Requerimientos de Portabilidad

La herramienta propuesta podrá ser usada bajo cualquier sistema operativo, para su implementación se usaron Herramientas de Programación y Gestión de Bases de Datos que son multiplataforma.

7. Software

Se debe disponer de sistemas operativos Linux, Windows 95 o superior para la instalación de la aplicación. Debe tenerse instalado el Java Runtime Environment (JRE) versión 1.5 o superior.

8. Hardware

Para el desarrollo y puesta en práctica del proyecto se requieren máquinas con los siguientes requisitos:

- Procesador Pentium 3 o superior
- 256 Mb de RAM
- 50 mb de capacidad del disco duro

2.2.3 Definición de los casos de uso

Este subepígrafe proporciona la definición de los actores que intervienen con el sistema, muestra el diagrama de caso de uso del sistema y describe textualmente los casos de uso del sistema.

Actores del sistema

El término *actor* significa el rol que algo o alguien ocupa cuando interactúa con el sistema.

Actores	**Justificación**
Especialista	Cualquier usuario que necesite realizar cierta predicción, es el encargado de solicitar que se realice la predicción.
Administrador	Es el encargado de actualizar el sistema creando nuevos modelos para la predicción utilizando programación genética.

Diagrama de Casos de Uso del Sistema

Este diagrama representa de forma gráfica como será concebido el sistema para una mejor comprensión.

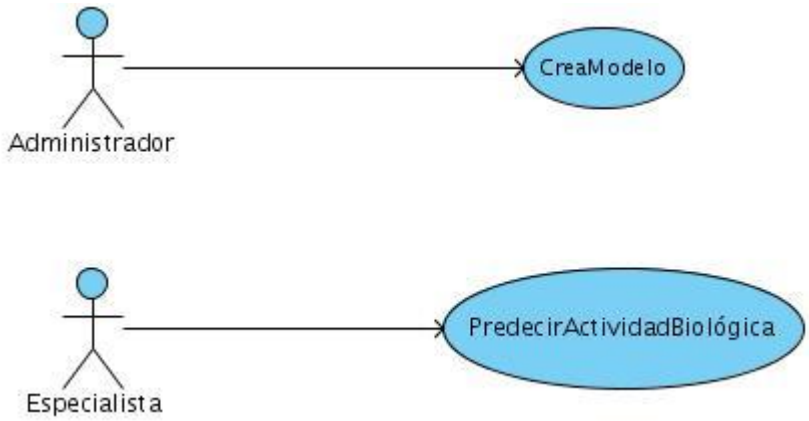


Figura 2.2 Diagrama de caso de uso del sistema

Descripción de los Casos de uso.

Con la posterior descripción de los casos de uso que se componen para el sistema, se obtendrá claramente la idea de cómo se realizarán las operaciones, cuándo y quienes intervienen en ellas.

Caso de Uso	Crear Modelo
Actores:	Administrador
Resumen:	El caso de uso inicia cuando el Administrador solicita Crear Modelo.
Referencia:	R1, R2
CU asociados:	**-**
Precondiciones:	Existencia de una base de datos molecular con resultados de sus descriptores.
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El administrador solicita Crear Modelo.	1. El sistema pide la entrada de la BDMD.
2. El administrador especifica la dirección donde pueden encontrar estos datos.	2.1 Se comprueba que los datos estén correctos. 2.2 Se crea el modelo matemático. 2.3 Se guardan los modelos en la base de conocimiento del sistema.
Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**
Poscondiciones:	
Prioridad:	Crítico
Especificaciones Complementaria:	
Puntos de Extensión	

Caso de Uso:	Predecir Actividad Biológica
Actores:	Especialista
Resumen:	El caso de uso se inicia cuando el Especialista solicita Realizar Predicción. Posteriormente se realizan los cálculos con los modelos existentes devolviendo los resultados de la predicción al Especialista.
Referencia:	R3, R4, R5, R6
CU asociados:	**-**
Precondiciones:	Que exista la base de modelos matemáticos.

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
1. El especialista gestiona cargar los ficheros de moléculas.	1. El sistema carga los ficheros de las moléculas
2. El especialista solicita Obtener Predicción	2.1 El sistema realiza los cálculos de los descriptores. 2.2. Selecciona Modelo matemático. 2.3. El sistema realiza la predicción. 2.4. El sistema devuelve la predicción.
3. El especialista Solicita guardar la información en un fichero.	3.1. El sistema guarda el fichero.

Flujos Alternos	
Acción del Actor	**Respuesta del Sistema**

Poscondiciones:	
Prioridad:	Crítico
Especificaciones Complementaria:	
Puntos de extensión	

Conclusiones

En el capítulo resultó concluida la fase de descripción del negocio, debido a la poca estructuración de los procesos del negocio y para una mejor comprensión se definieron en este capítulo conceptos, que fueron relacionados mediante un diagrama de Modelo del Dominio para modelar, de forma general, como se desarrolla este proceso y así brindar una clara definición de los requisitos que debe cumplir el sistema. Fueron seleccionados los actores y los casos de uso con los que constará dicho sistema, estos últimos con sus descripciones correspondientes. Se ganó claridad en cuanto a la concesión del sistema a construir y se sentaron las bases para las restantes fases del proceso de análisis, diseño e implementación.

CAPÍTULO 3: ANÁLISIS Y DISEÑO DEL SISTEMA

En este capítulo se describe la solución propuesta para el trabajo, se presentan los diagramas de clases de análisis organizados por casos de uso, con el propósito de una mejor comprensión del diseño. Después de realizar un estudio profundo sobre el funcionamiento del sistema y todas las clases definidas en el diagrama de clases del análisis, se pasa a la fase de diseño. Se muestran las características de la interfaz del usuario, los posibles errores que pueden cometer el usuario y la respuesta que le dará el sistema a cada uno de ellos.

3.1 Diagrama de clases del análisis

Una vez que se posee el modelo de dominio y la definición de los casos de uso del sistema se procede a realizar un refinamiento de los mismos, chequeando que exista un balance correcto entre los diferentes artefactos que se van obteniendo. En este flujo de trabajo se realizarán las definiciones de las clases de análisis que intervienen en el proceso, a partir de las clases del dominio que se poseen, de las interfaces que se poseen para el intercambio con los actores del sistema, así como de la necesidad de tener controlados cada uno de los procesos que se desarrollan.

Una vez definidas todas las clases del análisis se modelan las asociaciones existentes entre ellas a través del diagrama de clases del análisis correspondiente a cada caso de uso.

Caso de uso: Predecir actividad.

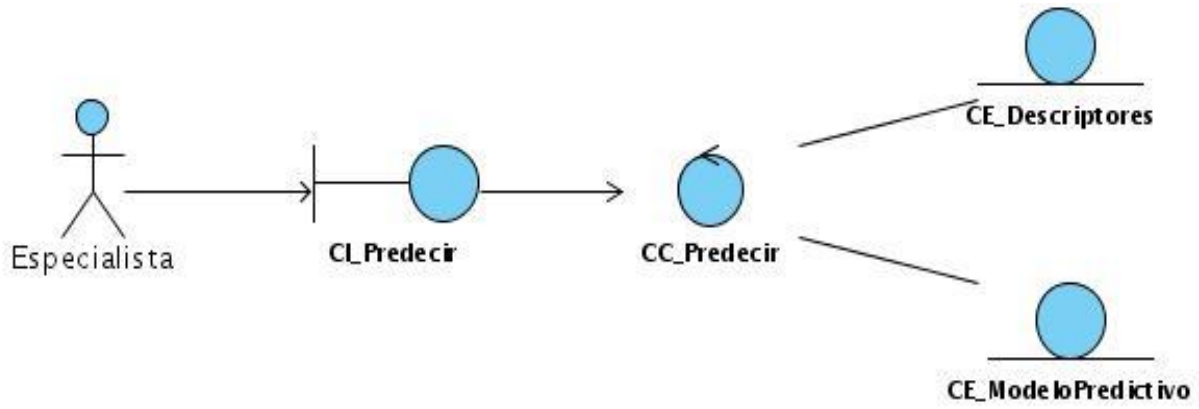


Figura 3.1 Diagrama de clases correspondiente al CU Predecir actividad.

Caso de uso: Crear modelo.

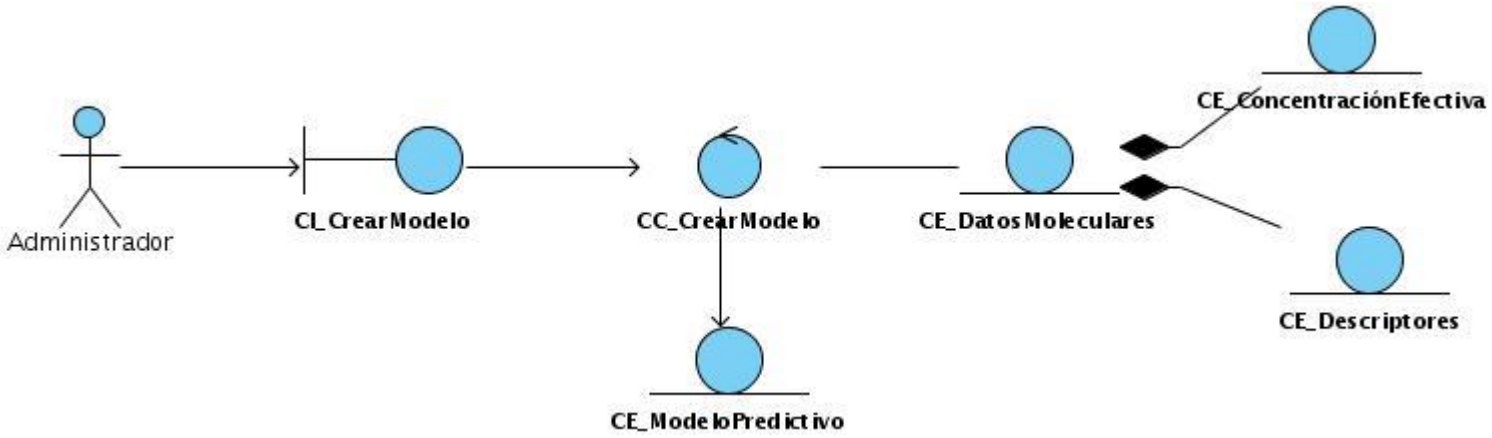


Figura 3.2 Diagrama de clases correspondiente al CU Crear modelo.

3.2 Diagrama de interacción.

Los Diagramas de Interacción son de dos tipos: Secuencia y Colaboración. Los diagramas de secuencia destaca el orden temporal de los mensajes que se intercambian entre Objetos, ofrece una visión clara del flujo a lo largo del tiempo mientras que los diagramas de colaboración destaca la organización de los objetos que participan, es un grafo donde los nodos son los objetos y los actos los mensajes que se mandan.

Con la idea de dar una visión gráfica de las interacciones de los actores con el sistema, se utilizan los diagramas de secuencia del sistema (DSS), los cuales muestran qué hace el sistema ante el medio, sin explicar el cómo.

Para cada uno de los casos de uso se define un diagrama de secuencia del sistema. En estos intervienen los actores del caso de uso, un objeto que representa al sistema, y se muestran los eventos que envía cada actor al sistema. Ver **Anexo # 1.**

3.3 Diagrama de clases del diseño

Después de realizar un estudio profundo sobre el funcionamiento del sistema y todas las clases definidas en los diagramas de clases del análisis, se pasa a la fase de diseño. En el **Anexo # 2**, se muestra los diagramas de clases del diseño.

3.4 Descripción de las clases del diseño

En el **Anexo # 3**, se muestra las descripciones detalladas de las principales clases que componen los diagramas de clases del diseño, con sus atributos y métodos. A continuación se muestra una descripción más general de todas las clases que componen los diagramas de clases del diseño para un mejor entendimiento de sus funcionalidades.

Nombre: BinaryOperator	
Tipo de clase: Entidad	
Modificador: abstract	
Padre: Operator	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
argc()	int
calc(int rowIndex, double v1, double v2)	abstract Number
calc(int rowIndex, Node[] childs)	Number
getName()	String
Descripción:	
La clase BinaryOperator representa a los operadores matemáticos de tipo binario. Su principal objetivo consiste en tener los métodos que identifican al operador y los métodos para definir la operación.	

Nombre: Column	
Tipo de clase: Entidad	
Modificador: -	
Padre: Leaf	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
Column(GeneticProg genprog)	-
calc(int row)	Number
clone(Node parent)	Node
equals(Object o)	boolean
getNodeType()	Node.NodeType
muteIt()	void
print(PrintWriter out)	void
Descripción:	
La clase Column representa a las Variables del modelo matemático. Su principal objetivo consiste en tener los métodos que identifican y devuelven a la variable y los métodos para definir las operaciones principales como evaluar, mutar e imprimir.	

Nombre: Constant	
Tipo de clase: Entidad	
Modificador: -	
Padre: Leaf	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
Constant(GeneticProg genprog)	-
calc(int row)	Number
clone(Node parent)	Node
equals(Object o)	boolean
getNodeType()	Node.NodeType
muteIt()	void
print(PrintWriter out)	void
Descripción:	
La clase Constant representa a las Constantes Numéricas del modelo matemático. Su principal objetivo consiste en tener los métodos que identifican y devuelven a la constante y los métodos para definir las operaciones principales como evaluar, mutar e imprimir.	

Nombre: Function	
Tipo de clase: Entidad	
Modificador: -	
Padre: Node	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
Function(GeneticProg genprog, src.GeneticProg.Shuttle n)	-
calc(int row)	Number
clone(Node parent)	Node
equals(Object o)	boolean
getNodeType()	Node.NodeType
muteIt()	void
getChildAt(int index)	Node
getChildCount()	int
toString()	String
print(PrintWriter out)	void
Descripción:	
La clase Function representa a las funciones formadas por un Operador y sus parámetros correspondientes, estas funciones son las que conforman el modelo matemático. Su principal objetivo consiste en tener los métodos que identifican y los métodos para definir las operaciones principales como evaluar, mutar e imprimir.	

Nombre: Leaf	
Tipo de clase: Entidad	
Modificador: abstract	
Padre: Node	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
getChildAt(int index)	Node
getChildCount()	int
Descripción:	
La clase Leaf representa a las hojas del árbol que se emplea para representar el modelo matemático.	

Nombre: Node	
Tipo de clase: Entidad	
Modificador: abstract	
Padre: -	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
calc(int row)	abstract Number
clone(Node parent)	abstract Number
countDescendant()	int
equals(Object o)	abstract boolean
getAllNodes()	Vector<Node>
getChildAt(int index)	abstract Node
getChildCount()	abstract int
getGeneticProg()	GeneticProg
getNodeType()	abstract Node.NodeType
getParent()	Node
getRoot()	Node
muteIt()	abstract void
print(PrintWriter out)	abstract void
toString()	String
Descripción:	
La clase Node representa a los nodos del árbol que se emplea para representar el modelo matemático. Los nodos pueden ser de tres tipos (COLUMN, CONSTANT, FUNCTION)	

Nombre: Operator	
Tipo de clase: Entidad	
Modificador: abstract	
Padre: -	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
argc()	abstract int
calc(int rowIndex, Node[] childs)	abstract Number
equals(Object o)	boolean
getName()	abstract String
getSpreadSheet()	GeneticProg
hashCode()	int
toString()	String
Descripción:	
La clase Operator representa a los operadores matemáticos.	

Nombre: Solution	
Tipo de clase: Entidad	
Modificador: -	
Padre: -	
Implementa: Serializable, Cloneable, Comparable<Solution>	
Principales Responsabilidad	
Nombre	**Tipo**
Solution(Node node, int generation)	-
calc()	double[]
Calculo()	double[]
clone()	Object
compareTo(Solution src)	int
equals(Object obj)	boolean
getGeneration()	int
getGeneticProg()	GeneticProg
getNode()	Node
getScore()	Double
mute()	void
toString()	String
Descripción:	
La clase Solution representa a las soluciones o modelos. Su principal objetivo consiste en tener los métodos que identifican al modelo y los métodos para definir las operaciones principales como evaluar, comparar e imprimir.	

Nombre: SVGPane	
Tipo de clase: vista	
Modificador: -	
Padre: JPanel	
Implementa:.ImageObserver, MenuContainer, Serializable, accessibility.Accessible	
Principales Responsabilidad	
Nombre	**Tipo**
SVGPane()	-
getSolution()	Solution
setSolution(Solution sol)	void
paintComponent(Graphics g1d)	void
Descripción:	
La clase SVGPane tiene como objetivo graficar los resultados observados contra los estimados en un eje de coordenadas cartesiana.	

Nombre: UnaryOperator	
Tipo de clase: Entidad	
Modificador: abstract	
Padre: Operator	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
argc()	int
calc(int rowIndex, double value)	abstract Number
calc(int rowIndex, Node[] childs)	Number
getName()	String
Descripción:	
La clase UnaryOperator representa a los operadores matemáticos de tipo unitario. Su principal objetivo consiste en tener los métodos que identifican al operador y los métodos para definir la operación.	

Nombre: GeneticProg	
Tipo de clase: Controladora	
Modificador: -	
Padre: -	
Implementa: Serializable	
Principales Responsabilidad	
Nombre	**Tipo**
GeneticProg()	-
getColumnCount()	int
getGenerationPerExplorer()	int
getMaxNANPercent()	double
getMinmax()	double[]
getNormalizedResultAt(int row)	Double
getNumberOfExplorer()	int
getNumberOfRun()	double
getRandom()	Random
getResultAt(int row)	Double
getRowCount()	int
getTable()	JTable
getValueAt(int row, int col)	Double
max_nodes_in_a_tree()	int
num_extra_parents()	int
num_parents()	int
proba_create_leaf()	double
proba_mutation()	double
read(BufferedReader in)	void
rnd()	double
run()	void
SalvarModeloSeriado()	void
SalvarModeloTexto()	void
setGenerationLabel(.JLabel generationLabel)	void
setTable(JTable table)	void
setTableModel(table.DefaultTableModel model)	void
setTableModelDefault(table.DefaultTableModel model)	void
setTextArea(JTextArea textarea)	void
setTextSVGPane(SVGPane svgPanes)	void
UpdateEXTRA_COLUMNS()	Void

UpdateParam(int maxNodeInATree, int numberOfParent, double probaMutation, double probaLeaf, double maxNANPercent, int numberOfExplorer, int numberOfGeneration, double numberOfRun, int numSum, int numResta, int numMult, int numDiv, int numSQRT, int numLog, int numExp)	Void
Descripción:	
La clase GeneticProg representa a la Programación Genética (PG). Su principal objetivo consiste en desarrollar y controlar el algoritmo de solución.	

Nombre: GenerarModeloPG	
Tipo de clase: vista	
Modificador: -	
Padre: -	
Implementa: Runnable	
Principales Responsabilidad	
Nombre	**Tipo**
GenerarModeloPG()	-
main(String[] args)	static void
run()	void
Descripción:	
La clase GenerarModeloPG es la vista principal en la generación de modelo. Tiene como objetivo controlar las acciones solicitadas por el usuario y mostrarles los resultados de las mismas.	

Nombre: Predecir	
Tipo de clase: vista	
Modificador: -	
Padre: -	
Implementa: Runnable	
Principales Responsabilidad	
Nombre	**Tipo**
Predecir()	-
main(String[] args)	static void
read(BufferedReader in)	void
run()	void
Descripción:	
La clase Predecir es la vista principal en la predicción de la actividad. Tiene como objetivo controlar las acciones solicitadas por el usuario y mostrarles los resultados de las mismas.	

3.5 Modelo de clases persistentes

Las clases persistentes representan información de larga duración o que persiste en el tiempo, es decir, es la información que se requiere almacenar para gestionarla en cualquier momento. En este caso la clase persistente se convierte en un fichero, lo cual permitirá tener un registro de los modelos generados.

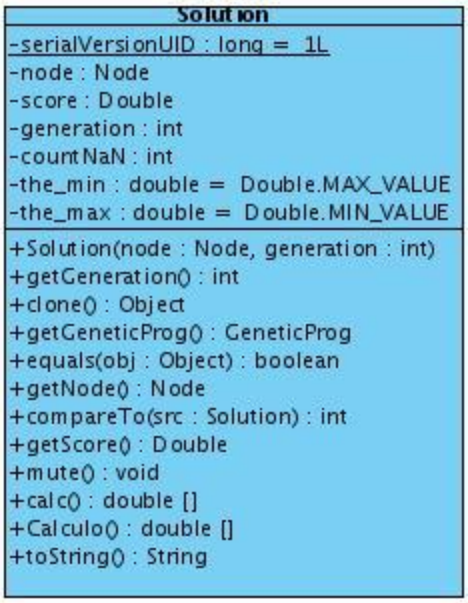


Figura 3.3 Diagrama de clases persistente.

3.6 Principios de diseño

3.6.1 Interfaz de usuario

Toda la aplicación debe seguir un Standard de diseño, la letra utilizada es *MS San Serif* de estilo regular y tamaño 10, cada uno de los botones tiene las mismas dimensiones, la aplicación se diseña para que el especialista químico que la use no tenga problema alguno al interactuar con las facilidades que brinda la herramienta. El sistema muestra un menú donde aparecen todas las opciones de trabajo. Ver **Anexo 4.**

3.6.2 Formato de salida de los reportes

Los reportes de las predicciones realizadas se harán en formato de tabla y se mostrarán en pantalla. Ver **Anexo 5.**

3.6.3 Tratamiento de errores

En el diseño de la interfaz se debe tener en cuenta el tratamiento de errores logrando que los mensajes de error que emita el sistema sean de fácil comprensión para el usuario y lo más descriptivos posibles y además debe alertarlos de posibles riesgos de las operaciones que realice, debe emitir un error si trata de abrir un fichero que no exista o alguna acción no valida para el sistema, esto se hace mediante el levantamiento de excepciones en el lenguaje java.

Se mostrará el siguiente icono de error cada vez que se encuentre un error.

Icono de error

Además de los mensajes de error existen otro tipo de mensajes que ayudan al usuario en el empleo de la aplicación, estos son los informativos y condicionales.

Los informativos dan a conocer el estado de la aplicación y acciones a realizar y estarán representados por el icono.

Icono de información

Los condicionales dan a conocer el próximo estado de la aplicación y posibilita el pase a esta o la permanencia en el estado actual y serán representados por el icono.

Icono de condición

3.6.4 Patrones de diseño utilizados

Los patrones de diseño, más comúnmente conocidos como "Design Patterns", son soluciones simples y elegantes a problemas específicos y comunes del diseño orientado a objetos. Son soluciones basadas en la experiencia y que se ha demostrado que funcionan. En el desarrollo de multitud de aplicaciones hay problemas de diseños que se repiten o que son análogos, es decir, que responden a un cierto patrón. Con el uso de patrones los diseños serán mucho más flexibles, modulares y reutilizables. Estos han revolucionado el diseño orientado a objetos y todo buen arquitecto de software debe conocerlos.

Para el resultado de la investigación se emplea un grupo de patrones relacionados con el diseño de software, llamados patrones GRASP (General Responsibility Asignment Software Patterns) los que describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Este grupo de patrones esta muy relacionado con los problemas básicos del diseño. La asignación correcta de las responsabilidades en el diseño orientado a objetos garantiza la alta cohesión de las clases y el bajo acoplamiento de los mismos, lo que posibilita más extensibilidad, adaptabilidad y menos tiempo para el mantenimiento del diseño. [Graig Larman en su libro UML y Patrones].

Los patrones GRASP son los siguientes:

1. Experto: Asignar una responsabilidad al experto en información, la clase que cuenta con la información necesaria para cumplir la responsabilidad. Es un patrón que se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos.
2. Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento. Indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada.
3. Alta Cohesión: Asignar una responsabilidad de modo que la cohesión siga siendo altas. Es un principio que debemos tener presente en todas las decisiones de diseño: es la meta principal que ha de buscarse en todo momento. Es un patrón evaluativo que el desarrollador aplica a1 valorar sus decisiones de diseño.

4. Bajo Acoplamiento: Estimula asignar una responsabilidad de modo que su colocación no incremente el acoplamiento tanto que produzca los resultados negativos propios de un alto acoplamiento. Soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecientan la oportunidad de una mayor productividad. No puede considerarse en forma independiente de otros patrones como Experto o Alta Cohesión, sino que más bien ha de incluirse como uno de los principios del diseño que influyen en la decisión de asignar responsabilidades. El acoplamiento tal vez no sea tan importante, si no se busca la reutilización.

5. Controlador: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema. Ofrece una guía para tomar decisiones apropiadas que generalmente se aceptan.

3.7 Arquitectura empleada

"La Arquitectura del Software es la organización fundamental de un sistema formado por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución".

3.7.1 Patrón arquitectónico empleado

Los patrones arquitectónicos especifican un conjunto predefinido de subsistemas con sus responsabilidades y una serie de recomendaciones para organizar los distintos componentes.

Patrón Modelo – Vista – Controlador

El patrón Modelo – Vista – Controlador está catalogado como un patrón de diseño de software donde:

> **Modelo:** Representación específica del dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio. La lógica de dominio añade significado a los datos. El modelo encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.
>
> **Vista:** Presenta el modelo en un formato adecuado para interactuar, usualmente un elemento de interfaz de usuario. Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

Controlador: Responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista. Los eventos son traducidos a solicitudes de servicio (“service requests”) para el modelo o la vista.

Muchas aplicaciones utilizan un mecanismo de almacenamiento persistente (como puede ser una base de datos) para almacenar los datos. Modelo – Vista – Controlador no menciona específicamente ésta capa de acceso a datos.

3.8 Diagrama de despliegue

Los Diagramas de Despliegue muestran la disposición física de los distintos nodos que componen un sistema y el reparto de los componentes sobre dichos nodos. La vista de despliegue representa la disposición de las instancias de componentes de ejecución en instancias de nodos conectados por enlaces de comunicación.

Para aplicaciones como la que se propone, que se ejecuta en una sola máquina y todos los dispositivos con que se relaciona son los estándares (teclado, mouse), el diagrama de despliegue va a estar constituido por un nodo, por lo que no se considera relevante incluirlo en la investigación.

Conclusiones

En este capítulo se hizo una descripción de la solución propuesta, utilizando la programación orientada a objeto, se definieron un total de 14 clases, se presentó el diagrama de clases organizado por casos de uso, además de mostrar el diagrama de clases general, se presentan algunas características en cuanto al diseño de la interfaz de la aplicación, tipo de letra, tamaño y se mencionan los patrones de diseño más utilizados para el desarrollo de la aplicación.

CAPÍTULO 4: IMPLEMENTACIÓN Y VALIDACIÓN DEL MODELO

En el presente capítulo se describe cómo los elementos del modelo de diseño se implementan en términos de componentes, para esto se muestra el diagrama de componentes. Además se validan los modelos difusos generados, mediante técnicas estadísticas, comparando los valores obtenidos con valores experimentales, con el objetivo de determinar el grado de precisión con que predicen los mismos.

4.1 Implementación

El flujo de trabajo de implementación describe cómo los elementos del modelo del diseño se implementan en términos de componentes y cómo estos se organizan de acuerdo a los nodos específicos en el modelo de despliegue.

4.1.1 Diagrama de Componentes.

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

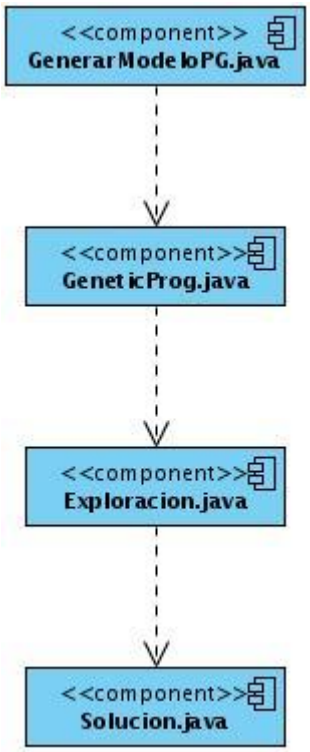


Figura 4.1 Diagrama de componente caso de uso CrearModelo .

Figura 4.2 Diagrama de componente caso de uso Predecir.

4.2 Validación de Modelos.

La validación de los resultados de un software está adquiriendo gran importancia debido a que cada vez se hace más patente la necesidad de obtener datos objetivos que permitan evaluar, predecir y mejorar la calidad del mismo, así como el tiempo y costo de desarrollo del mismo.

Un modelo es una invención, algo que se desarrolla para explicar la relación entre una serie de datos que se posee con el fenómeno al cual están relacionados. Para que un modelo sea útil, tiene que permitir que los datos ajusten de forma coherente, es decir, tiene que poder explicar lo que pasa de una manera lógica. Son muy útiles cuando se quiere estudiar fenómenos o sistemas complejos. Representa lo que se desea estudiar de modo más simple, centrándose en los aspectos que se consideran importantes del fenómeno. Aunque no hay que olvidar que los modelos no son el fenómeno sino solo esquemas que lo explican, y que representan por lo general sólo la parte que se eligió estudiar.

La construcción de modelos predictivos consiste en la creación de un modelo de clasificación a partir de un conjunto de entrenamiento. Los registros del conjunto de entrenamiento tienen que pertenecer a un pequeño grupo de clases predefinidas, cada clase corresponde a un valor de la etiqueta. El modelo inducido (clasificador) consiste en una serie de patrones que son útiles para distinguir las clases. Una vez que se ha inducido el modelo se puede utilizar para predecir automáticamente la clase de otros registros no clasificados.

La efectividad, en nuestro caso, depende entre otros factores de la naturaleza de los descriptores moleculares seleccionados para caracterizar la estructura química. Las variables independientes empleadas son los descriptores de Randic, Valencia y Partición de la Refractividad Molecular calculados a 2000 moléculas seleccionadas del ensayo NCI Yeast Anticancer Drug Screen, en una relación de 1x1 entre molécula activas e inactivas. La muestra de prueba esta compuesta por 500 moléculas con una relación también de 1x1 entre moléculas activas e inactivas.

Con la aplicación se generaron varios modelos de clasificación, con parámetros específicos en cada caso y se le realizo la prueba a cada uno de ellos. A continuación se muestran tres de los modelos obtenidos con los resultados de sus pruebas.

Primer Modelo Obtenido

La obtención del primer modelo estuvo regida por los parámetros siguientes:

Parámetros de Control	
Máx. Nodos de un Árbol	23
Número de Individuos	7
Probabilidad de Mutación	0.95
Probabilidad de Nodo Hoja	0.4
Número de Exploraciones	3
Número de Generaciones	50
Número de Corridas	10
Parámetros Matemáticos	
Suma	20
Resta	20
Multiplicación	20
División	20
Raíz Cuadrada	1
Log	1
Potencia	1

El modelo obtenido fue el siguiente

Res(VPC34 , exp(Res(Div(Div(Sum(Mul(Mul(Res(Div(R3 , Res(PR9 , Sum((0.0771195003970967) , Mul(Div((-3.0) , (0.313022218557664)) , exp(Res(R1 , VPC23))))) , (-0.423835729696165)) , PRPC14) , V8) , (-36.0)) , Div(Sum((-45.0) , PR11) , Res((42.0) , (0.44935767824700124)))) , Res(Sum((0.15291662923673555) , V11) , Mul((0.12176443507457146) , (0.39870678210133426)))) , VPC23)))

Con un error 0.4231577 obtenido en la generación 472. Con el modelo se clasificó la muestra de entrenamiento con un nivel de acierto de 57.21 % y la muestra de prueba con un nivel de acierto de 69.47%.

Segundo Modelo Obtenido

La obtención del segundo modelo estuvo regida por los parámetros siguientes:

Parámetros de Control	
Máx. Nodos de un Árbol	50
Número de Individuos	15
Probabilidad de Mutación	0.95
Probabilidad de Nodo Hoja	0.4
Número de Exploraciones	3
Número de Generaciones	50
Número de Corridas	10
Parámetros Matemáticos	
Suma	20
Resta	20
Multiplicación	20
División	20
Raíz Cuadrada	5
Log	1
Potencia	5

El modelo obtenido fue el siguiente

Res(Div(Div(R14 , Sum(exp(Mul((-0.37770993898957594) , Mul(R1 , Res((-22.0) , Div((31.0) , (34.0))))))) , Div(Mul(Sum((-0.20528314329749586) , RPC14) , Res(Mul(Res(Sum(PRPC24 , (0.41381226967870066)) , Div(Div(R4 , Sum((1.0) , Div(Res(Sum((-0.20528314329749586) , RPC14) , (18.0)) , PRPC13))) , (-35.0))) , R1) , (26.0))) , PRPC13))) , (-17.0)) , exp(Sum(Div(exp(Mul(Sum((0.1770115004076953) , exp(Mul(PRPC14 , (-48.0)))) , Mul(VPC23 , R3))) , (-41.0)) , Div(exp(Mul((14.0) , PR10)) , (-47.0)))))

Con un error 0.3804 obtenido en la generación 456. Con el modelo se clasificó la muestra de entrenamiento con un nivel de acierto de 57.92 % y la muestra de prueba con un nivel de acierto de 70.09%.

Tercer Modelo Obtenido

La obtención del tercer modelo estuvo regida por los parámetros siguientes:

Parámetros de Control	
Máx. Nodos de un Árbol	90
Número de Individuos	30
Probabilidad de Mutación	0.95
Probabilidad de Nodo Hoja	0.3
Número de Exploraciones	3
Número de Generaciones	80
Número de Corridas	10
Parámetros Matemáticos	
Suma	20
Resta	20
Multiplicación	20
División	20
Raíz Cuadrada	10
Log	1
Potencia	10

El modelo obtenido fue el siguiente

exp(Div(Mul(Sum((-0.37304040350763135) , Res(Sum(exp(PR3) , PR2) , (-11.0))) , exp(Mul(Mul(Sum(exp(PR4) , RPC13) , Sum(PRPC23 , Mul(VC3 , Res(Mul(R13 , Mul(Div(Div((19.0) , (-0.3106957875591979)) , Sum(R4 , Res(Mul(PR8 , Res(Sum(exp((-0.03263100734541258)) , Mul(Sum(exp((0.4777269173422336)) , Div(Mul(Sum((-0.11387287941992086) , Res(Sum(exp(PR3) , (4.0)) , Mul(PR8 , Res(Sum(R5 , Mul(Sum(exp(Mul(Sum(exp(PR4) , RPC13) , Sum(PRPC23 , Mul(VC3 , Res(Mul(PR11 , Mul(Div(Div((19.0) , (-0.3106957875591979)) , Sum(R4 , Res(Sum(PR8 , Res(Div(sqrt(exp(exp(exp((-38.0))))) , Div(exp(Sum((-37.0) , (0.02708734837791127))) , (-9.0))) , Sum(PR2 , exp((-0.1195329664738185))))) , exp(RPC23)))) , exp(Sum(Div(R8 , PRPC13) , (0.19444322801299097))))) , exp(RPC33)))))) , Div(Mul(Sum(Mul((0.3931381634441873) , R2) , Res(PR15 , RPC13)) , exp(Mul(Mul(R2 , Sum(exp(Sum(VPC14 , VPC14)) , Mul(VC3 , Res(Mul(R13 , Mul(Mul(Res((19.0) , V14) , Sum(R10 , Res(Mul(PR8 , Res(Div(R8 , PRPC13) , Sum(Mul(exp((0.036006990868200583)) , Sum(PR1 , (-0.08578932512878124))) , exp((0.47126702178783875))))) , exp(VC4)))) , exp(V2))) , PR5)))) , exp(PR15)))) , (-0.08688078576381342))) , (-0.16088201283541415))) , Mul(Div(Res(RC3 , Mul(Div(exp(PR3) , PR2) , (-49.0))) , Res(RPC13 , (34.0))) , R10))))) , exp(Mul(Mul(R2 , Sum(exp(Sum(VPC14 , VPC14)) , Mul(VC3 , Res(Mul(R13 , Mul(Mul(Res((19.0) , V14) , Sum(R10 , Res(Mul(PR8 , Res(V13 , Sum(Mul(exp((0.036006990868200583)) , Sum(PR1 , V8)) , V5))) , exp(VC4)))) , exp(V2))) , exp(R11))))) , exp(PR11)))) , (-0.08688078576381342))) , (-0.16088201283541415))) , Sum(PR2 , exp((-0.1195329664738185))))) , exp(RPC23)))) , exp(Sum(Div(R8 , PRPC13) , (0.19444322801299097))))) , exp(RPC33))))) , exp(Div(Div(Div(Sum(RPC34 , R11) , (-27.0)) , exp(Div((47.0) , (-14.0)))) , sqrt((0.06370813012370724))))))) , (-0.08688078576381342)))

Con un error 0.3536 obtenido en la generación 782. Con el modelo se clasificó la muestra de entrenamiento con un nivel de acierto de 60.35 % y la muestra de prueba con un nivel de acierto de 67.08%.

Después de la realización de las pruebas podemos afirmar que con el empleo de parámetros medios (utilizados en la segunda prueba) su obtuvo un acertado modelo de clasificación. Ver **Anexo 6** con imágenes de salida.

Conclusiones

Se utilizaron diagramas de componentes para estructurar el modelo de implementación en términos de subsistemas de implementación y mostrar las relaciones entre sus elementos. El uso más importante de estos diagramas es mostrar la estructura de alto nivel del modelo de implementación, especificando los subsistemas correspondientes y sus dependencias a la hora de importar código.

Se demostró la efectividad de los modelos de las muestras seleccionadas de entrenamiento y prueba a partir de las pruebas realizadas al sistema.

CONCLUSIONES GENERALES

- Se analizo, diseño e implemento una aplicación informática para la predicción de actividad anticancerigena de compuestos orgánicos a partir de descriptores topológicos e híbridos utilizando programación genética.
- Se elaboro un plug-in de la aplicación informática desarrollada para su incorporación al visualizador de la Plataforma.
- Las pruebas realizadas a los modelos obtenidos por la aplicación en una clasificación de dos subclases de compuestos anticancerigenos, fueron satisfactorias.

RECOMENDACIONES

Para aumentar las prestaciones que brinda la aplicación implementada se recomienda:

- Definir un conjunto de teclas de acceso rápido a las funcionalidades de generación de modelo, predicción, exportación y demás funcionalidades.
- Realizar un mayor número de prueba al sistema con diferentes ensayos para garantizar aun más la calidad de los modelos que se obtienen.
- Implementar una funcionalidad que permita exportar los modelos obtenidos en diferentes formatos, buscando una mayor portabilidad del sistema.
- Implementar una funcionalidad que permita diferentes reportes de la predicción así cono graficas de los mismos en varios formatos.

REFERENCIAS BIBLIOGRÁFICAS

[1] ROMERO ZALDIVAR, C. R. Un Novedoso Enfoque para el Diseño 'Racional In-Silico' de Fármacos Disponible en: http://www.forum.villaclara.cu/ponencias

[2] Sheridan, R. P.; Venkataraghavan, R. (1987) Acc. Chem. Res. 20, 322-329.p.

[3] FERNANDEZ, N. R. Sistema "GRATO (GRAph-TOol)" para la Visualización Molecular y el Cálculo de Descriptores, ISJAE, 2006. p.

[4] URIARTE, E. Recent advances on the role of topological indeces in drug discovery research, 2001.

[5] J. A. PADRÓN, R. C., R.F.PELLÓN Molecular descriptor based on a molar refractivity partition using Randic-type graph-theoretical invariant Nov 2001.

[6] ESTRADA, E. Graph Theoretical Invariant of Randic Revisited, may 1995.

--- Spectral Moment of Edge Adjacency Matrix in Molecular Graphs, 1995.

[7] E. Estrada, "Spectral Moment of Edge Adjacency Matrix in Molecular Graphs," 1995.

[8] ROGER, E. E. Estudios sobre nuevos modelos Grafo-Teoricospara el diseño de molecular en química orgánica. Centro de Bioactivos Químicos. Santa Clara, UCLV, 1996. p.

[9] PÉREZ, P. Y. P. Un modelo para el aprendizaje y la clasificación automática basado en técnicas de softcomputing., 2005. 13. p.

[10] ---. Un modelo para el aprendizaje y la clasificación automática basado en técnicas de softcomputing., 2005. 14. p.

[11] ---. Un modelo para el aprendizaje y la clasificación automática basado en técnicas de softcomputing., 2005. 16-17. p.

[12] ---. Notas sobre Computación Evolutiva. Disponible en: http://www.redcientifica.com/gaia/ce/ceno_c.htm

[13] ---. Un modelo para el aprendizaje y la clasificación automática basado en técnicas de softcomputing., 2005. 16-17. p.

[14] ---. Una aplicación de programación genética al área de control. Buenos Aires, Univercidad de Belgrano, 2005. 13. p.

[15] ---. Una aplicación de programación genética al área de control. Buenos Aires, Univercidad de Belgrano, 2005. 23. p.

[16] ---. Una aplicación de programación genética al área de control. Buenos Aires, Univercidad de Belgrano, 2005. 16. p.

[17] ROMERO, A. V. G. SISTEMA PARA PREDICCIÓN ACTIVIDAD BIOLÓGICA DE COMPUESTOS ORGÁNICOS, ISJAE, 2006. 30. p.

[18] MOLPECERES, A. Procesos de Desarrollo RUP, XP y FDD 2002. [Disponible en: http://www.willydev.net/descargas/articulos/general/cualxpfddrup.PDF

[19] ---. Introducción a los Sistemas y Herramientas CASE. Disponible en: http://ceds.nauta.es/informes/case01.htm

[20] ---. Notas técnicas de java., 2003. [Disponible en: http://www.teknoda.com/tips/java/java01.pdf

[21] ALTAMIRANO, A. V. Comparativo de Entornos de Desarrollo Integrados (IDE's). Disponible en: http://www.ubicuos.com/files/downloads/ComparativoIDES.pdf

BIBLIOGRAFÍA

1. IVAR JACOBSON; GRADY BOOCH*, et al. El Proceso Unificado de Desarrollo de Software.* 1999. vol. volumen 1.
2. JAVA, T. D. *Características de Java* [Consultado el: 19 de enero de 2007]. Disponible en: http://www.cica.es/formacion/JavaTut/Intro/tabla.html.
3. PARADIGM, V. *Documentation* [Consultado el: 18 de enero de 2007]. Disponible en: http://www.visual-paradigm.com/.
4. SANCHEZ, M. A. M. *Metodologías de Desarrollo de Software* [Consultado el: 17 de enero de 2007]. Disponible en: http://64.233.167.104/search?q=cache:lzgHcDIV3CEJ:www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf
5. SCHIAFFINO, P. D. S. *Descubrimiento de Conocimiento en Bases de Datos* [Consultado el: 17 de enero de 2007]. Disponible en: http://www.exa.unicen.edu.ar/catedras/dbdiscov/clase2c.pdf.
6. ZAMORANO, J. P. *Clasificacion de patrones: Métodos no supervisados* [Consultado el: 23 de enero de 2007]. Disponible en: http://www.iula.upf.es/materials/050418porta_5.pdf.
7. Livingstone, D.J. (2000) “The characterization of chemical structures using molecular properties. A survey”, J. Chem. Inf. Comput. Sci. 40, 195–209.
8. Tounge, B.A., Pfahler, L.B. and Reynolds, C.H. (2002) “Chemical information based scaling of molecular descriptors: a universal chemical scale for library design”, J. Chem. Inf. Comput. Sci. 42, 879–884.
9. Lucic, B. and Trinajstic, N. (1999) “Multivariate regression outperforms several robust architectures of neural networks in QSAR modeling”, J. Chem. Inf. Comput. Sci. 39, 121–132.
10. Whitley, D.C., Ford, M.G. and Livingstone, D.J. (2000) “Unsupervised forward selection: a method for eliminating redundant variables”, J. Chem. Inf. Comput. Sci. 40, 1160–1168.

11. Rogers, D. and Hopfinger, A.J. (1994) “Application of genetic function approximation to quantitative structure–activity relationships and quantitative structure–property relationships”, J. Chem. Inf. Comput. Sci. 34, 854–866.

12. Cho, S.J. and Hermsmeier, M.A. (2002) “Genetic algorithm guided selection: variable selection and subset selection”, J. Chem. Inf. Comput. Sci. 42, 927–936.

13. Izrailev, S. and Agrafiotis, D. (2001) “A novel method for building regression tree models for QSAR based on artificial ant colony systems”, J. Chem. Inf. Comput. Sci. 41, 176–180.

14. UML y Patrones. Introducción al análisis y diseño orientado a objetos, Craig Larman, Mexico, 1999, ISBN: 970-17-0261-1. Disponible en http://biblioteca.uci.cu/titdigitales.htm#pro

15. Estadística, Primera Parte, Juan L. Cué Muñiz, Ernestina Castell Gil, José M. Hernandez Carratalá, Ciudad de la Habana, 1987, Universidad de la Habana, Facultad de Matemática Cibernética. Disponible en http://biblioteca.uci.cu/titdigitales.htm#pro

16. [J. C. Escalona et.al,]J.C. Escalona, R. Carrasco, J. A. Padrón, *Introducción al diseño de Fármacos,* Folleto para la docencia de la asignatura de Farmacia, Universidad de Oriente.[Kurup, 2002] A Kurup, R. Garg, S. B. Mekapati, C. Hans, Bioorg. Med. Chem., in press, 2002.

17. UCITEVE, P.; D. D. C. AUDIOVISUAL, et al. Fase de Elaboración. Análisis - Diseño, 2005

18. UCITEVE, P.; D. D. C. AUDIOVISUAL, et al. Fase de inicio. Levantamiento de requisitos., 2005.

19. Algoritmos genéticos. Http://eddyalfaro.galeon.com/geneticos.html. ISSN: 1579-0223.

20. Kandid 1.0, arte genético en Java. 26/1/2005. http://gimp.hispalinux.es/article.php?storyid=23

21. Genetic Programming Inc. sep 18, 2006. [Disponible en: http://www.genetic-programming.com

22. Genetic Programming: On the Programming of Computers by Means of Natural Selection. p. 0-262-11170-5

23. Intel.ligència Artificial, Facultat d'Informàtica de Barcelona. Disponible en: http://www.lsi.upc.es/~bejar/ia/ia.html

24. Introducción a la Inteligencia Artificial. Disponible en: http://cruzrojaguayas.org/inteligencia/Que%20es%20IA.htm

25. Java API for genetic algorithms 21.05.2004. [Disponible en: http://www.jaga.org/

26. GONZÁLEZ, I. L. A. Arte genético: función de aptitud estética, Julio de 2005 [Disponible en: http://www.monografias.com/trabajos33/arte-genetico/arte-genetico.shtml

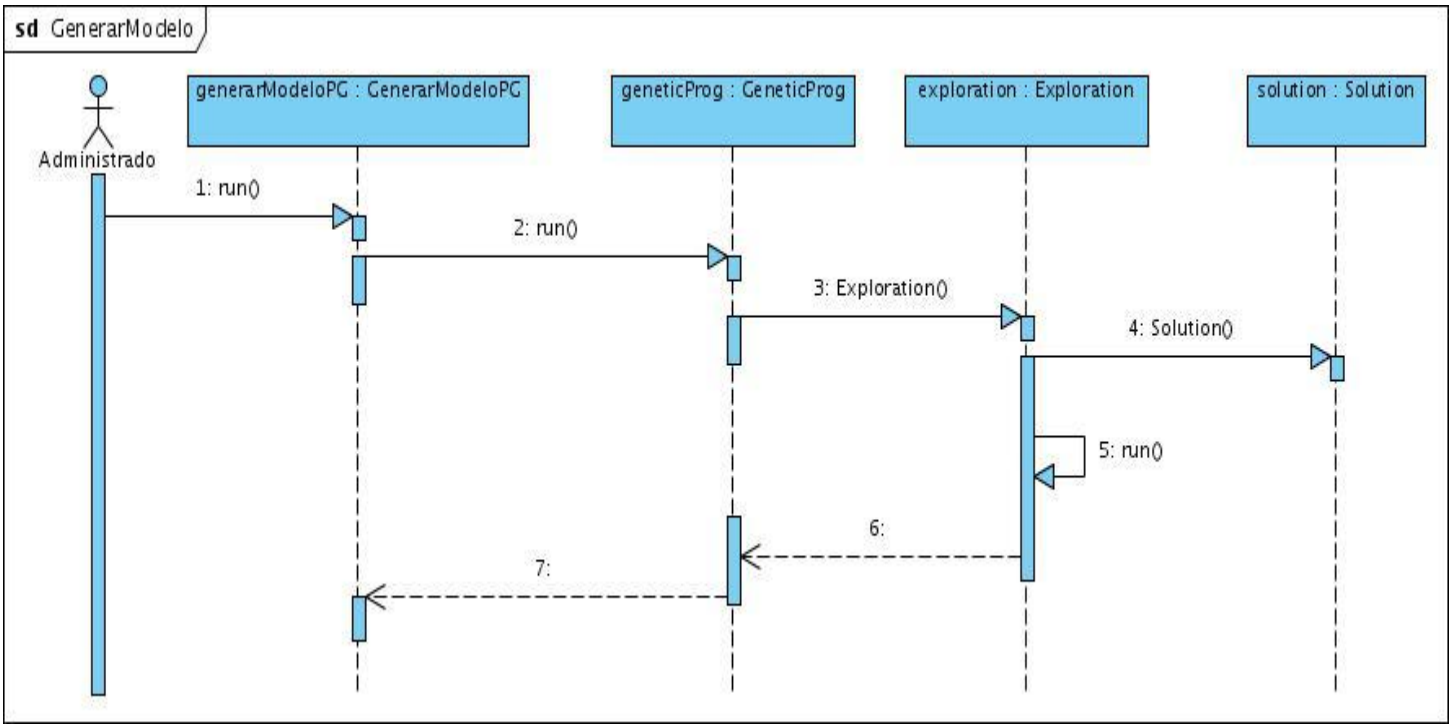
27. LUNT, E. Simple Symbolic Regression Using Genetic Programming à la John Koza. Disponible en: http://alphard.ethz.ch/gerber/approx/default.html

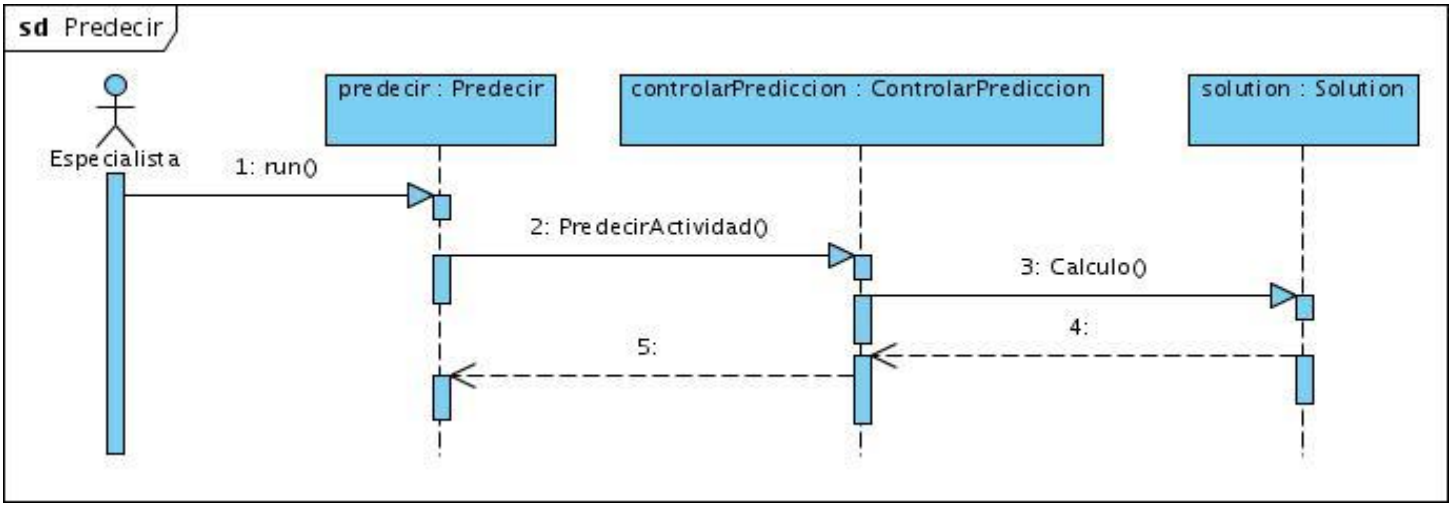
28. MANZANARES, E. M. Programación Genética 2004-11-02 [Disponible en: http://www.genetic-programming.com/published/computerbits0696.html

29. ROCÍO, R., ROSSANA, RAFAEL Programación Genética, 2002/05/28. [Disponible en: http://www.depi.itch.edu.mx/apacheco/expo/html/ai14/gp.html

ANEXOS

Anexo # 1 Diagramas de secuencia correspondientes a los caso de uso del sistema.





Anexo # 2: Diagramas de clases.

Diagrama de clases general

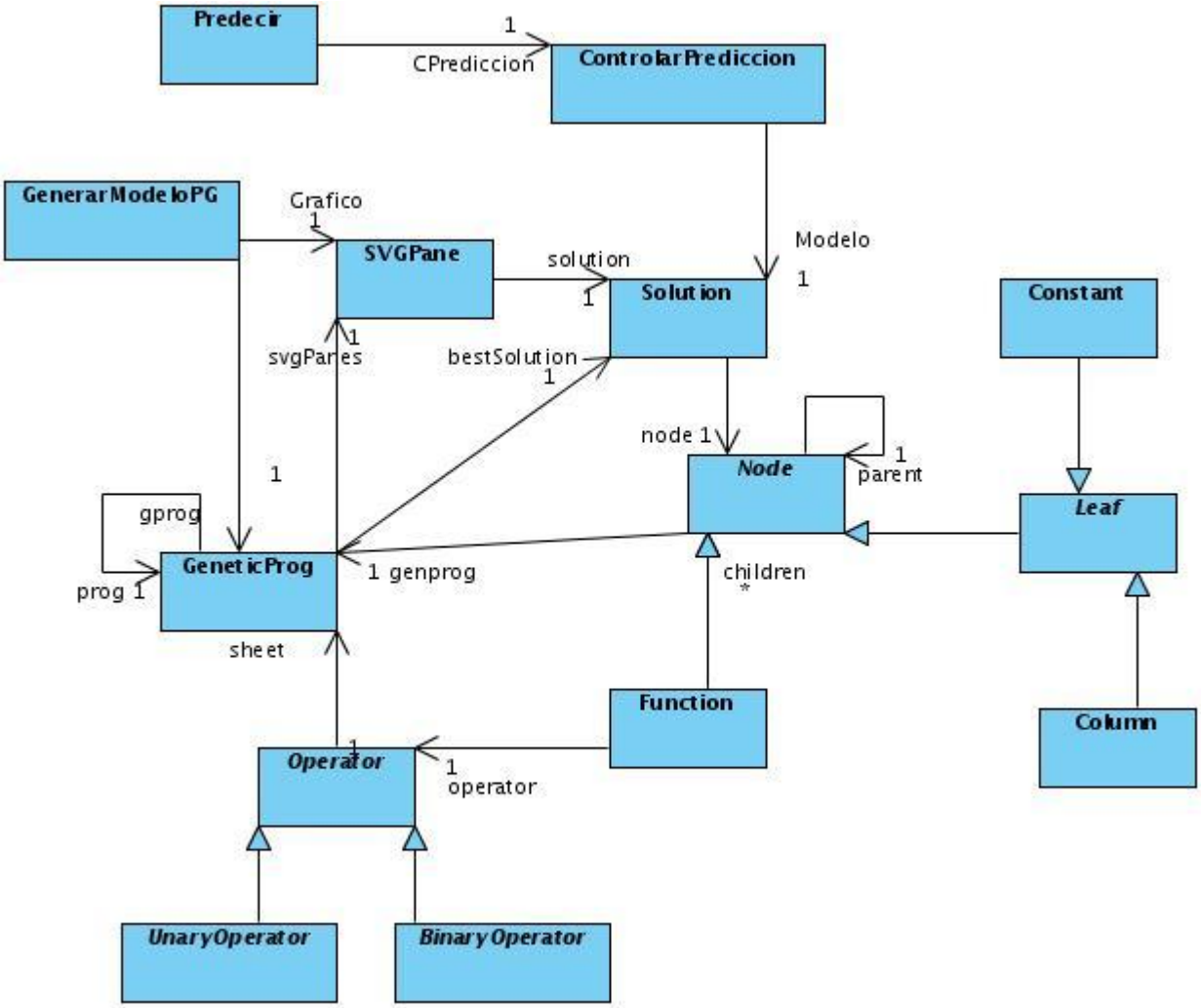


Diagrama de clases para el caso de uso CrearModelo

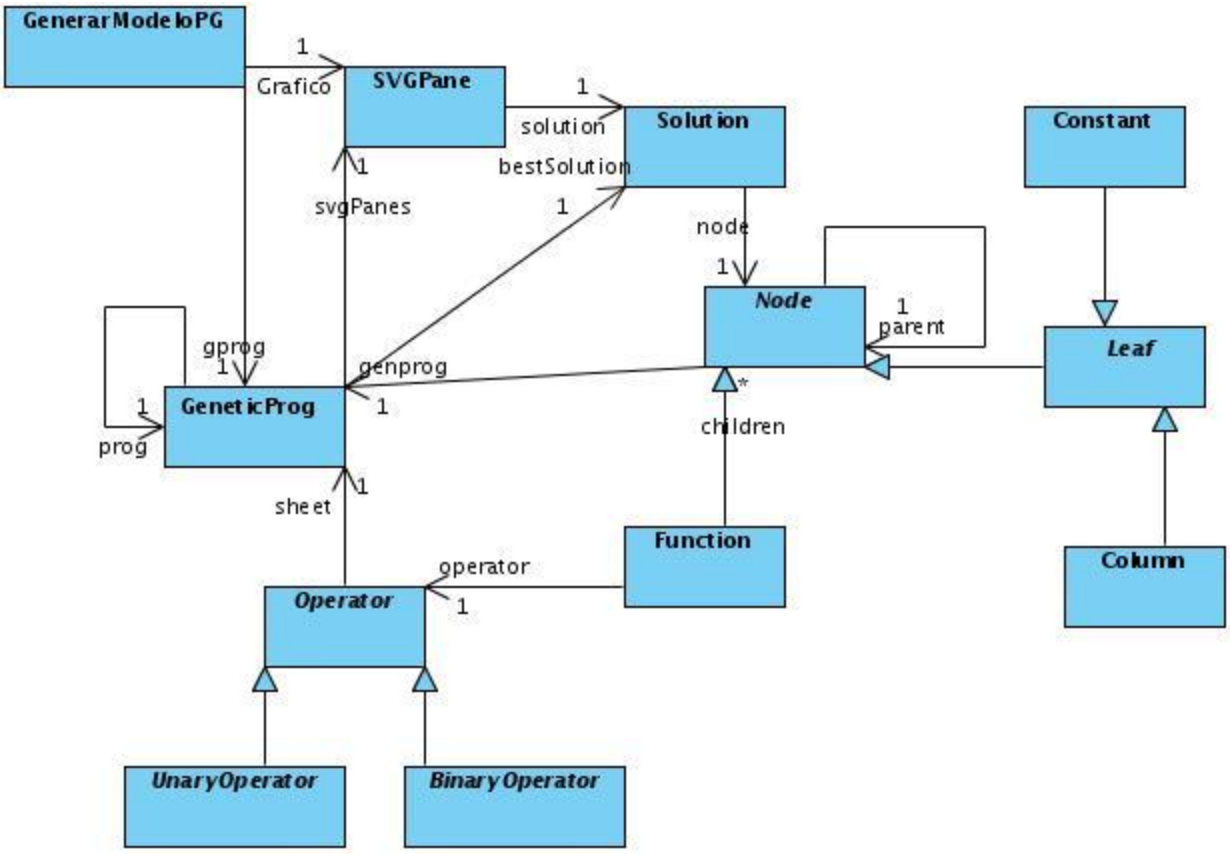
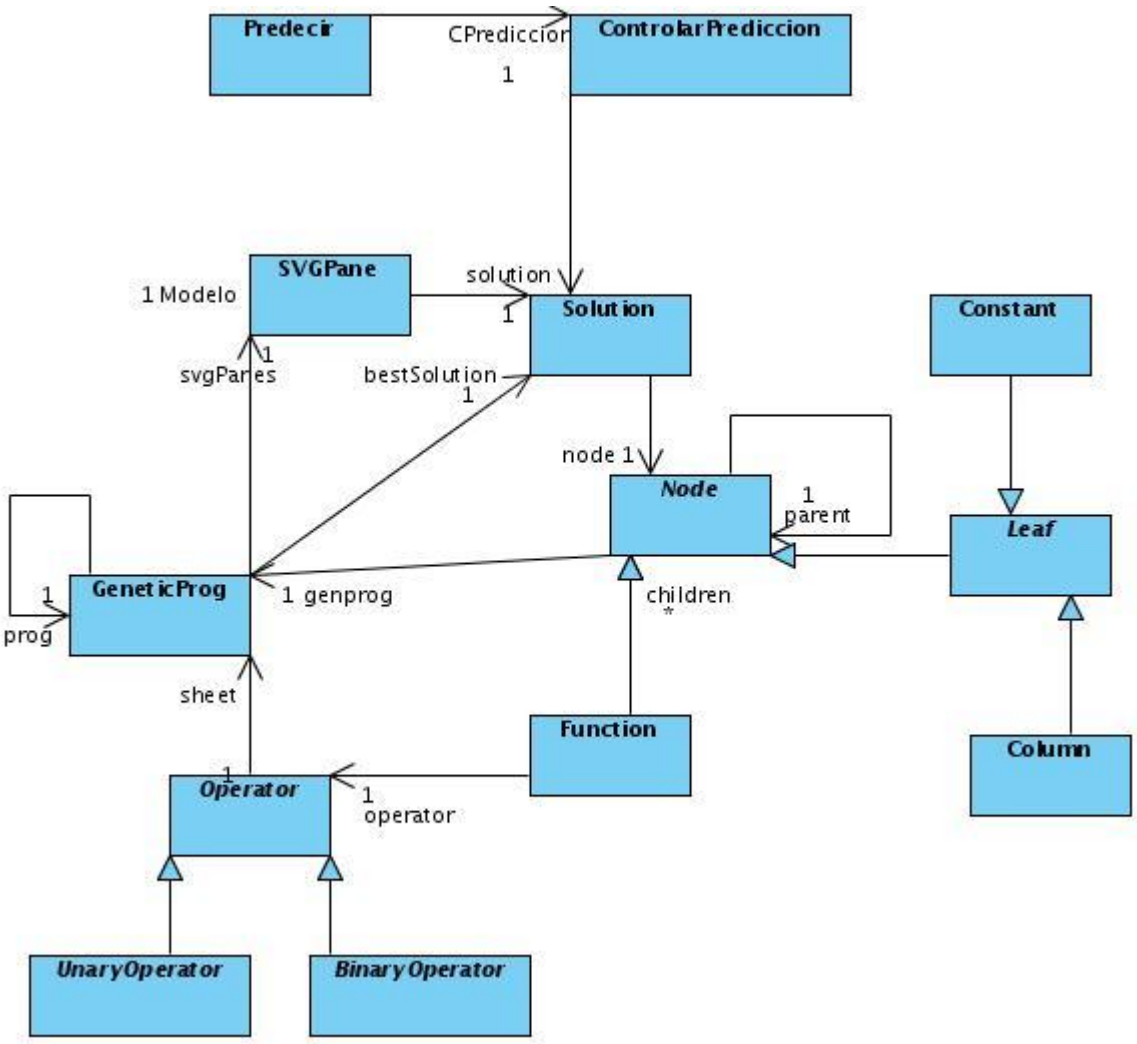
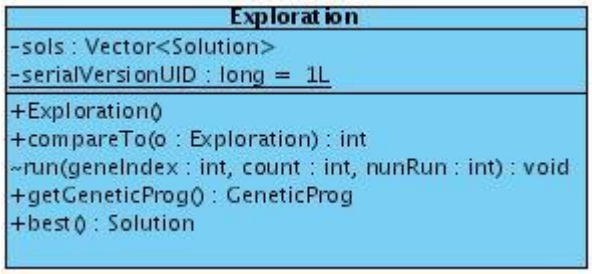
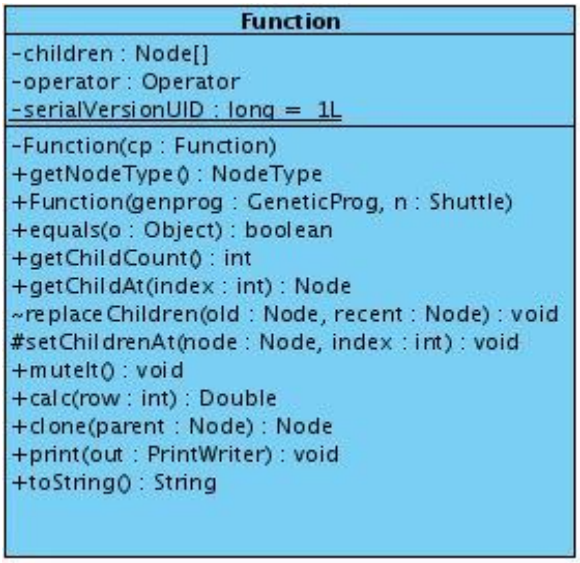


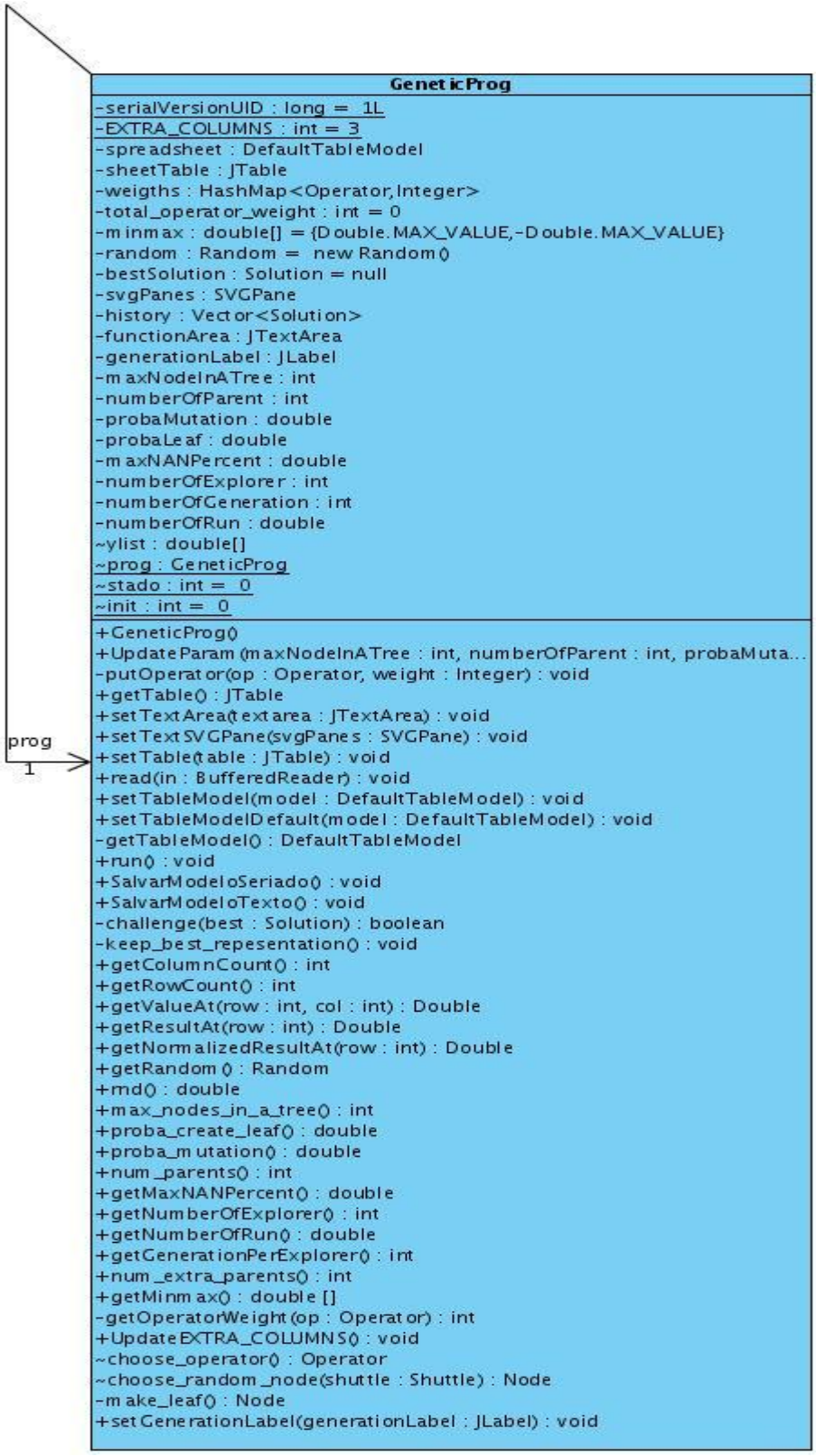
Diagrama de clases para el caso de uso Predecir

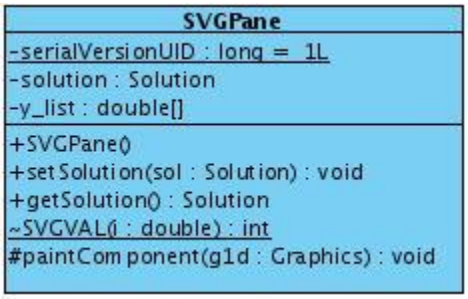


Clases que componen los diagramas anteriores con los atributos y métodos correspondientes.

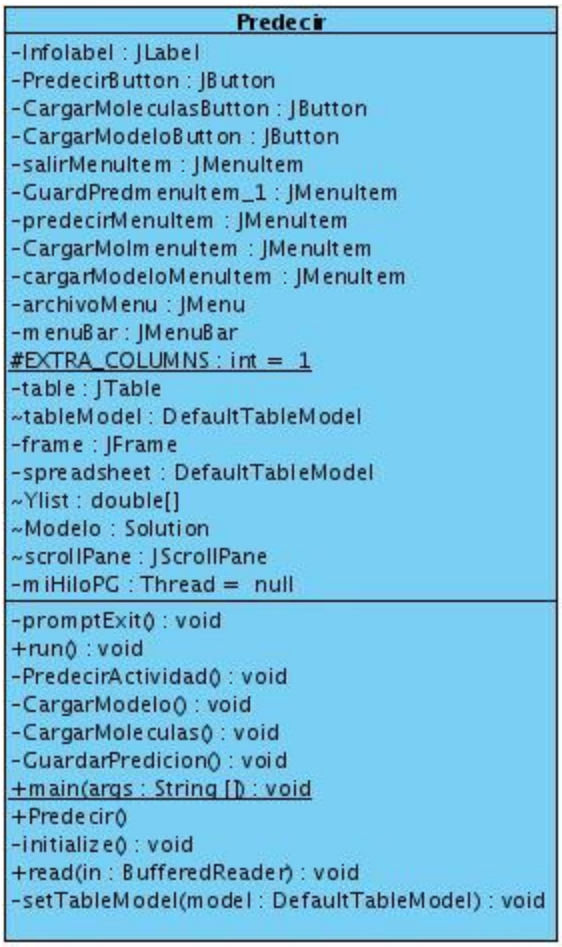


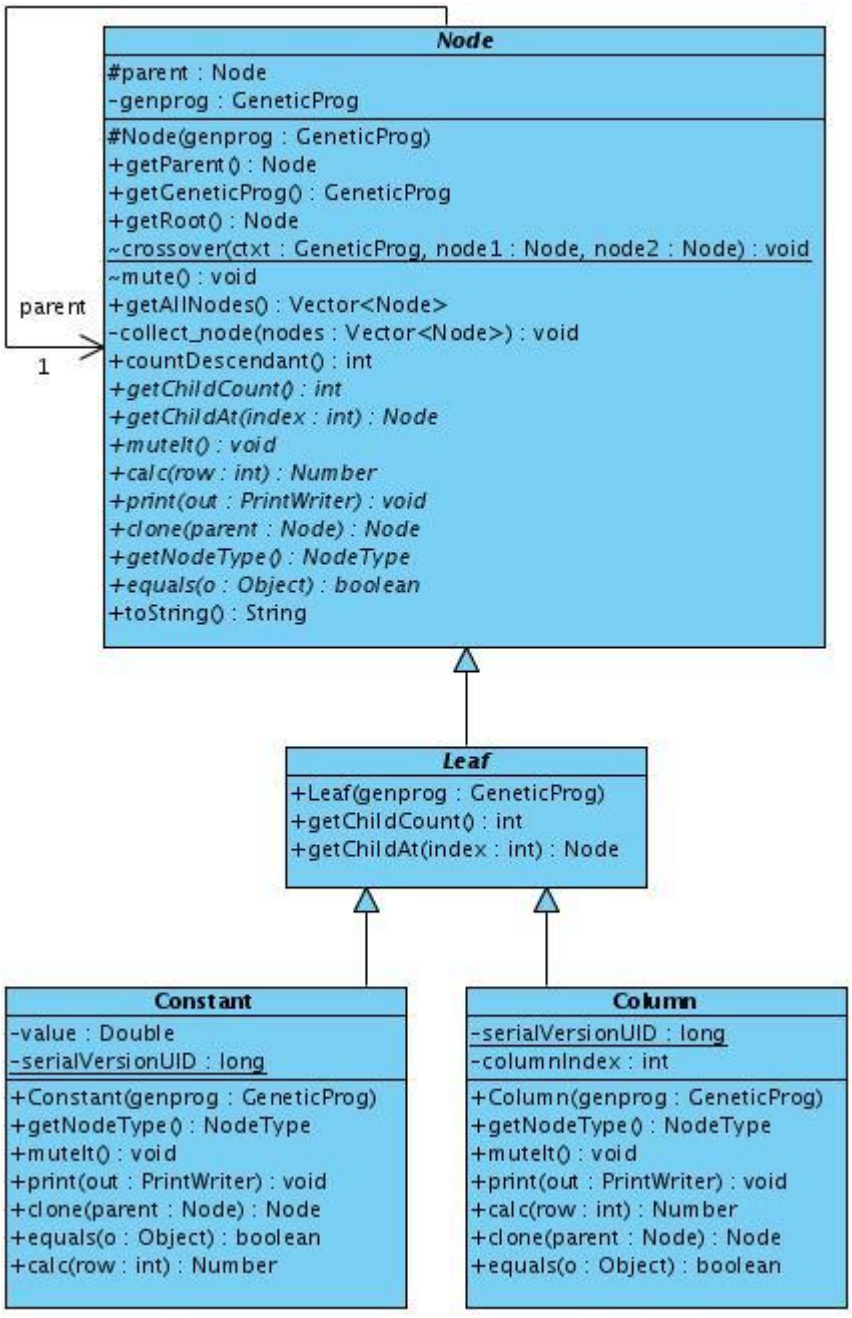


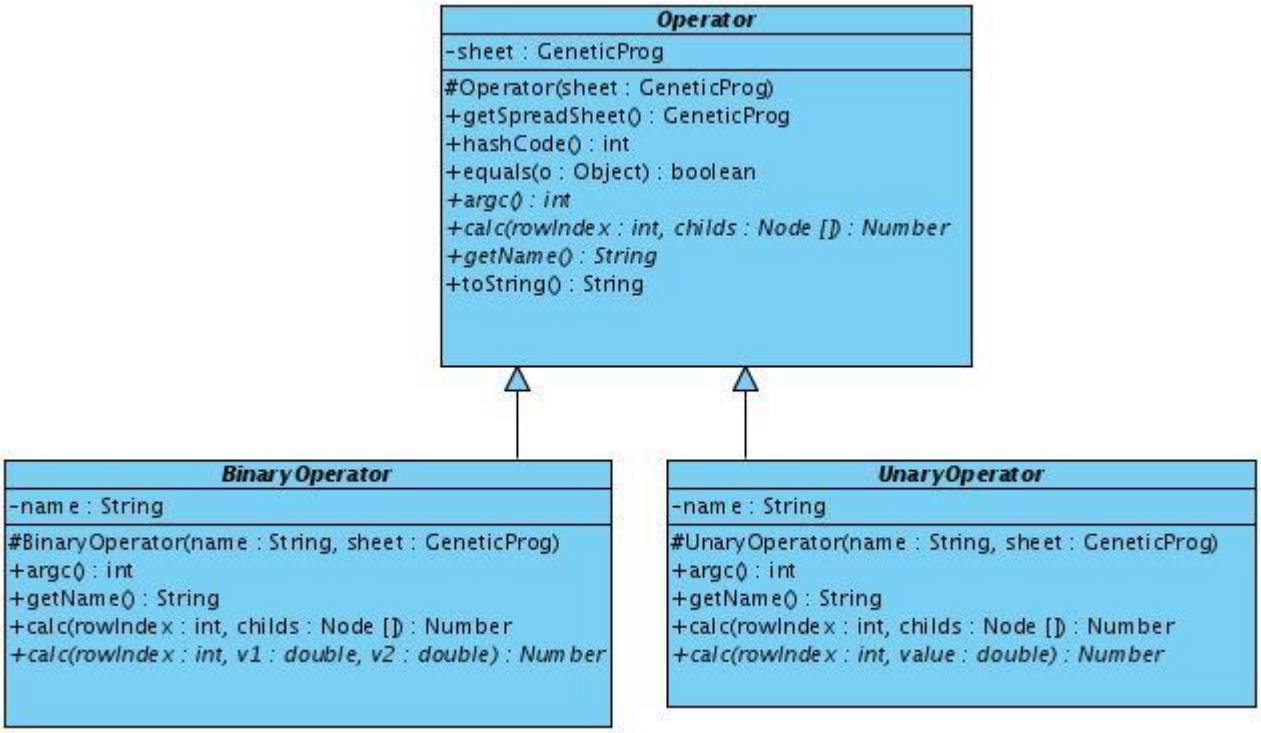

GeneticProg
-serialVersionUID : long = 1L
-EXTRA_COLUMNS : int = 3
-spreadsheet : DefaultTableModel
-sheetTable : JTable
-weigths : HashMap<Operator,Integer>
-total_operator_weight : int = 0
-minmax : double[] = {Double.MAX_VALUE,-Double.MAX_VALUE}
-random : Random = new Random()
-bestSolution : Solution = null
-svgPanes : SVGPane
-history : Vector<Solution>
-functionArea : JTextArea
-generationLabel : JLabel
-maxNodeInATree : int
-numberOfParent : int
-probaMutation : double
-probaLeaf : double
-maxNANPercent : double
-numberOfExplorer : int
-numberOfGeneration : int
-numberOfRun : double
~ylist : double[]
~prog : GeneticProg
~stado : int = 0
~init : int = 0
+GeneticProg()
+UpdateParam(maxNodeInATree : int, numberOfParent : int, probaMuta...
-putOperator(op : Operator, weight : Integer) : void
+getTable() : JTable
+setTextArea(textarea : JTextArea) : void
+setTextSVGPane(svgPanes : SVGPane) : void
+setTable(table : JTable) : void
+read(in : BufferedReader) : void
+setTableModel(model : DefaultTableModel) : void
+setTableModelDefault(model : DefaultTableModel) : void
-getTableModel() : DefaultTableModel
+run() : void
+SalvarModeloSeriado() : void
+SalvarModeloTexto() : void
-challenge(best : Solution) : boolean
-keep_best_repesentation() : void
+getColumnCount() : int
+getRowCount() : int
+getValueAt(row : int, col : int) : Double
+getResultAt(row : int) : Double
+getNormalizedResultAt(row : int) : Double
+getRandom() : Random
+rnd() : double
+max_nodes_in_a_tree() : int
+proba_create_leaf() : double
+proba_mutation() : double
+num_parents() : int
+getMaxNANPercent() : double
+getNumberOfExplorer() : int
+getNumberOfRun() : double
+getGenerationPerExplorer() : int
+num_extra_parents() : int
+getMinmax() : double []
-getOperatorWeight(op : Operator) : int
+UpdateEXTRA_COLUMNS() : void
~choose_operator() : Operator
~choose_random_node(shuttle : Shuttle) : Node
-make_leaf() : Node
+setGenerationLabel(generationLabel : JLabel) : void
prog
1

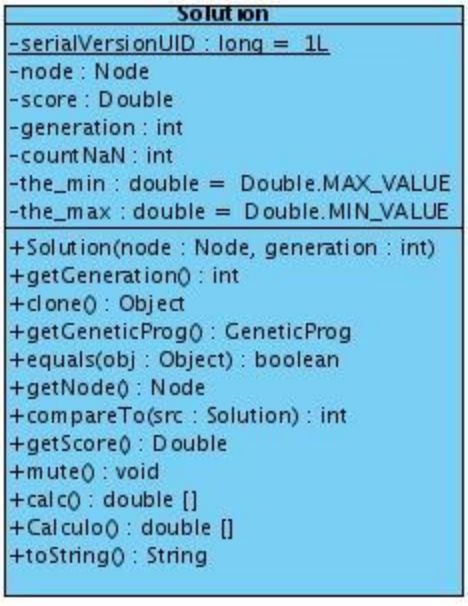

SVGPane
-serialVersionUID : long = 1L
-solution : Solution
-y_list : double[]
+SVGPane()
+setSolution(sol : Solution) : void
+getSolution() : Solution
~SVGVAL(i : double) : int
#paintComponent(g1d : Graphics) : void


GenerarModeloPG
-textModelo : JTextArea
-scrollPane_1 : JScrollPane
-detenerModeloMenuItem : JMenuItem
-salirMenuItem : JMenuItem
-obtenerModeloMenuItem : JMenuItem
-exportarModeloMenuItem : JMenuItem
-guardarModeloMenuItem : JMenuItem
-cargarDatosMenuItem : JMenuItem
-archivoMenu : JMenu
-InformacionLabel : JLabel
-Graf : JPanel
-Grafico : SVGPane
-ExponencialSpinner : JSpinner
-LogSpinner : JSpinner
-RaizCuadSpinner : JSpinner
-DivSpinner : JSpinner
-MultipSpinner : JSpinner
-RestaSpinner : JSpinner
-SumaSpinner : JSpinner
-NumCorridasSpinner : JSpinner
-NumGenrExplSpinner : JSpinner
-NumExplSpinner : JSpinner
-MaxNanSpinner : JSpinner
-ProbHojaSpinner : JSpinner
-ProbMutaSpinner : JSpinner
-NumIndvSpinner : JSpinner
-MaxNarbolspinner : JSpinner
-tabladatos : JTable
-springLayout : SpringLayout
-frame : JFrame
-miHiloPG : Thread = null
~gprog : GeneticProg
+main(args : String []) : void
+GenerarModeloPG()
-DetenerCalculo() : void
+run() : void
-geticon() : Icon
-CreaGP() : void
-CargarDatos() : void
-promptExit() : void
-initialize() : void


Predecir
-Infolabel : JLabel
-PredecirButton : JButton
-CargarMoleculasButton : JButton
-CargarModeloButton : JButton
-salirMenuItem : JMenuItem
-GuardPredmenuItem_1 : JMenuItem
-predecirMenuItem : JMenuItem
-CargarMolmenuItem : JMenuItem
-cargarModeloMenuItem : JMenuItem
-archivoMenu : JMenu
-menuBar : JMenuBar
#EXTRA_COLUMNS : int = 1
-table : JTable
~tableModel : DefaultTableModel
-frame : JFrame
-spreadsheet : DefaultTableModel
~Ylist : double[]
~Modelo : Solution
~scrollPane : JScrollPane
-miHiloPG : Thread = null
-promptExit() : void
+run() : void
-PredecirActividad() : void
-CargarModelo() : void
-CargarMoleculas() : void
-GuardarPredicion() : void
+main(args : String[]) : void
+Predecir()
-initialize() : void
+read(in : BufferedReader) : void
-setTableModel(model : DefaultTableModel) : void


Node
#parent : Node
-genprog : GeneticProg
#Node(genprog : GeneticProg)
+getParent() : Node
+getGeneticProg() : GeneticProg
+getRoot() : Node
~crossover(ctxt : GeneticProg, node1 : Node, node2 : Node) : void
~mute() : void
+getAllNodes() : Vector<Node>
-collect_node(nodes : Vector<Node>) : void
+countDescendant() : int
+getChildCount() : int
+getChildAt(index : int) : Node
+mutelt() : void
+calc(row : int) : Number
+print(out : PrintWriter) : void
+clone(parent : Node) : Node
+getNodeType() : NodeType
+equals(o : Object) : boolean
+toString() : String
parent
1
Leaf
+Leaf(genprog : GeneticProg)
+getChildCount() : int
+getChildAt(index : int) : Node
Constant
-value : Double
-serialVersionUID : long
+Constant(genprog : GeneticProg)
+getNodeType() : NodeType
+mutelt() : void
+print(out : PrintWriter) : void
+clone(parent : Node) : Node
+equals(o : Object) : boolean
+calc(row : int) : Number
Column
-serialVersionUID : long
-columnIndex : int
+Column(genprog : GeneticProg)
+getNodeType() : NodeType
+mutelt() : void
+print(out : PrintWriter) : void
+calc(row : int) : Number
+clone(parent : Node) : Node
+equals(o : Object) : boolean


Operator
-sheet : GeneticProg
#Operator(sheet : GeneticProg)
+getSpreadSheet() : GeneticProg
+hashCode() : int
+equals(o : Object) : boolean
+argc() : int
+calc(rowIndex : int, childs : Node []) : Number
+getName() : String
+toString() : String
BinaryOperator
-name : String
#BinaryOperator(name : String, sheet : GeneticProg)
+argc() : int
+getName() : String
+calc(rowIndex : int, childs : Node []) : Number
+calc(rowIndex : int, v1 : double, v2 : double) : Number
UnaryOperator
-name : String
#UnaryOperator(name : String, sheet : GeneticProg)
+argc() : int
+getName() : String
+calc(rowIndex : int, childs : Node []) : Number
+calc(rowIndex : int, value : double) : Number


Solution
-serialVersionUID : long = 1L
-node : Node
-score : Double
-generation : int
-countNaN : int
-the_min : double = Double.MAX_VALUE
-the_max : double = Double.MIN_VALUE
+Solution(node : Node, generation : int)
+getGeneration() : int
+clone() : Object
+getGeneticProg() : GeneticProg
+equals(obj : Object) : boolean
+getNode() : Node
+compareTo(src : Solution) : int
+getScore() : Double
+mute() : void
+calc() : double []
+Calculo() : double []
+toString() : String

Anexo # 3 Descripción de las principales clases del diseño.

Nombre: GeneticProg	
Tipo de clase : Controladora	
Atributo	**Tipo**
Responsabilidades:	
Nombre:	UpdateParam(**int** maxNodeInATree ,**int** numberOfParent, **double** probaMutation, **double** probaLeaf,**double** maxNANPercent, **int** numberOfExplorer, **int** numberOfGeneration , **double** numberOfRun , **int** numSum, **int** numResta, **int** numMult, **int** numDiv,**int** numSQRT, **int** numLog, **int** numExp)
Descripción:	Es el método responsable de actualizar los parámetros de ejecución.
Nombre:	read(BufferedReader in)
Descripción:	Es el método responsable de leer los datos de entrada para obtener el modelo. se le pasa como parámetro un el BufferedReader de lectura.
Nombre:	run()
Descripción:	Es el método responsable de la creación de las exploraciones y la ejecución de las mismas para que estas obtengan la solución.
Nombre:	SalvarModeloSeriado()
Descripción:	Es el método responsable de realizar el seriado de la solución y guardarlo.
Nombre:	SalvarModeloTexto()
Descripción:	Es el método responsable de realizar la impresión de la solución.
Nombre:	challenge(Solution best)
Descripción:	Es el método responsable de realizar la sustitución de una solución por una con mayor score.
Nombre:	getNormalizedResultAt(**int** row)
Descripción:	Es el método responsable de realizar la normalización los resultados de la fila que se le pase como parámetro.
Nombre:	choose_operator()
Descripción:	Es el método responsable de elegir un operador aleatorio.
Nombre:	choose_random_node(Shuttle shuttle)
Descripción:	Es el método responsable de elegir un nodo aleatorio
Nombre:	make_leaf()
Descripción:	Es el método responsable de construir un nodo hoja aleatorio para la solución.

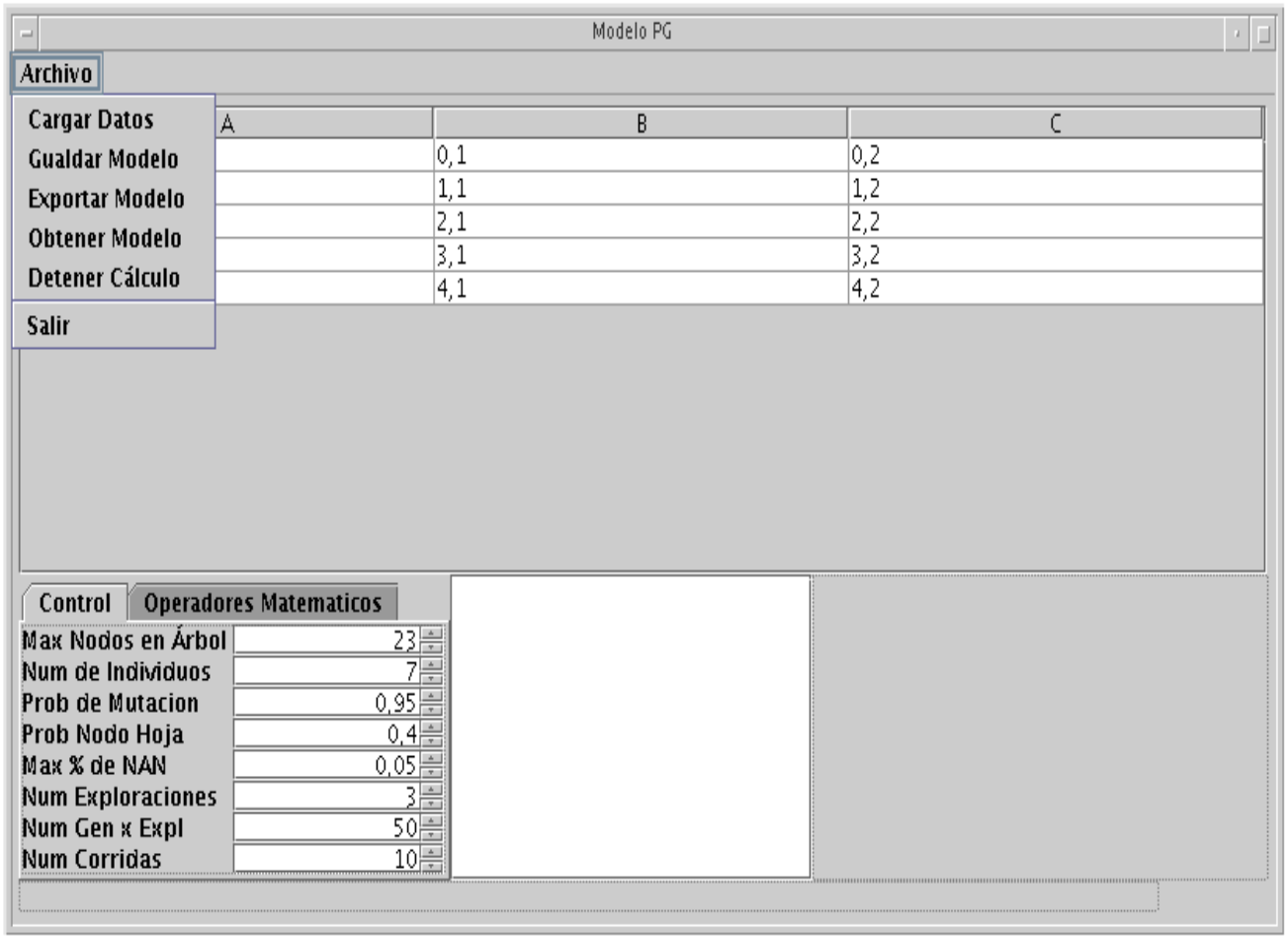
Nombre: ControlarPrediccion	
Tipo de clase : controladora	
Atributo	**Tipo**
Responsabilidades:	
Nombre:	PredecirActividad()
Descripción:	Es el método responsable de obtener la evaluación del modelo y guardar una lista con los resultados.
Nombre:	CargarModelo()
Descripción:	Es el método responsable de cargar el modelo desde un fichero seriado binario.
Nombre:	getYlist()
Descripción:	Es el método responsable de devolver los resultados de la Predicción.

Nombre: Exploration	
Tipo de clase : controladora	
Atributo	**Tipo**
Responsabilidades:	
Nombre:	compareTo()
Descripción:	Es el método responsable de comparar una Exploración con otra de acuerdo a cual tiene la mejor solución.
Nombre:	run()
Descripción:	Es el método responsable de la ejecución de la exploración para obtener las soluciones.
Nombre:	best()
Descripción:	Es el método responsable de devolver la mejor solución de la exploración

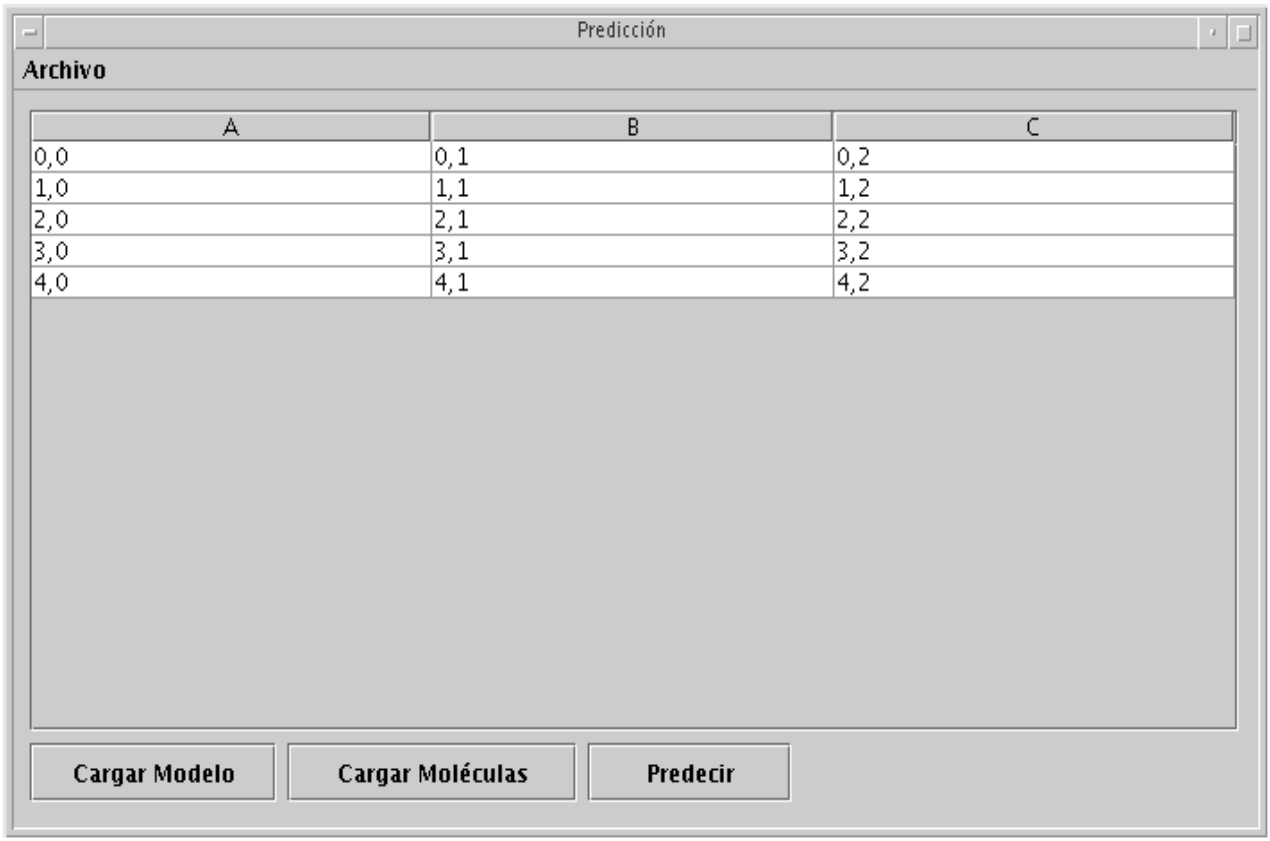
Nombre: Solution	
Tipo de clase : Entidad	
Atributo	**Tipo**
node	Node
score	Double
generation	**int**
countNaN	**int**
the_min	**double**
the_max	**double**
Responsabilidades:	
Nombre:	calc()
Descripción:	Es el método responsable de evaluar la solución y normalizar los resultados y obtener el score de la solución. Devuelve una lista con los resultados.
Nombre:	Calculo()
Descripción:	Es el método responsable de evaluar la solución y des normalizar el resultado.
Nombre:	toString()
Descripción:	Es el método responsable de convertir en una cadena de texto la solución.
Nombre:	compareTo()
Descripción:	Es el método responsable de comprara una solución con otra de acuerdo al score
Nombre:	mute()
Descripción:	Es el método responsable de realizar mutaciones en la solución.
Descripción:	Además de los get y set utilizados para el acceso de los atributos de la clase.

Anexo # 4 Prototipos de Interfaz

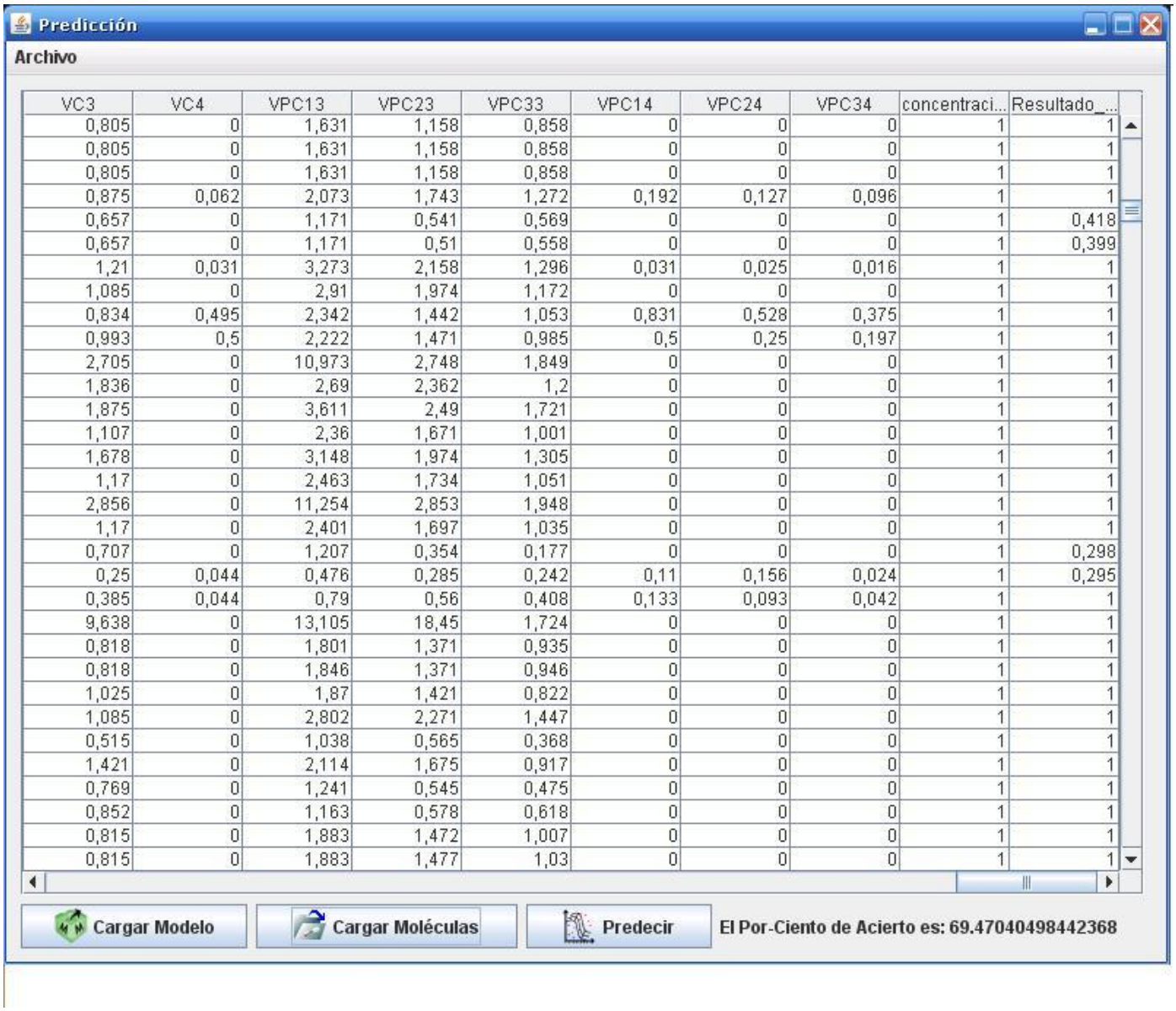
Interfaz Crear Modelo



Interfaz Predecir

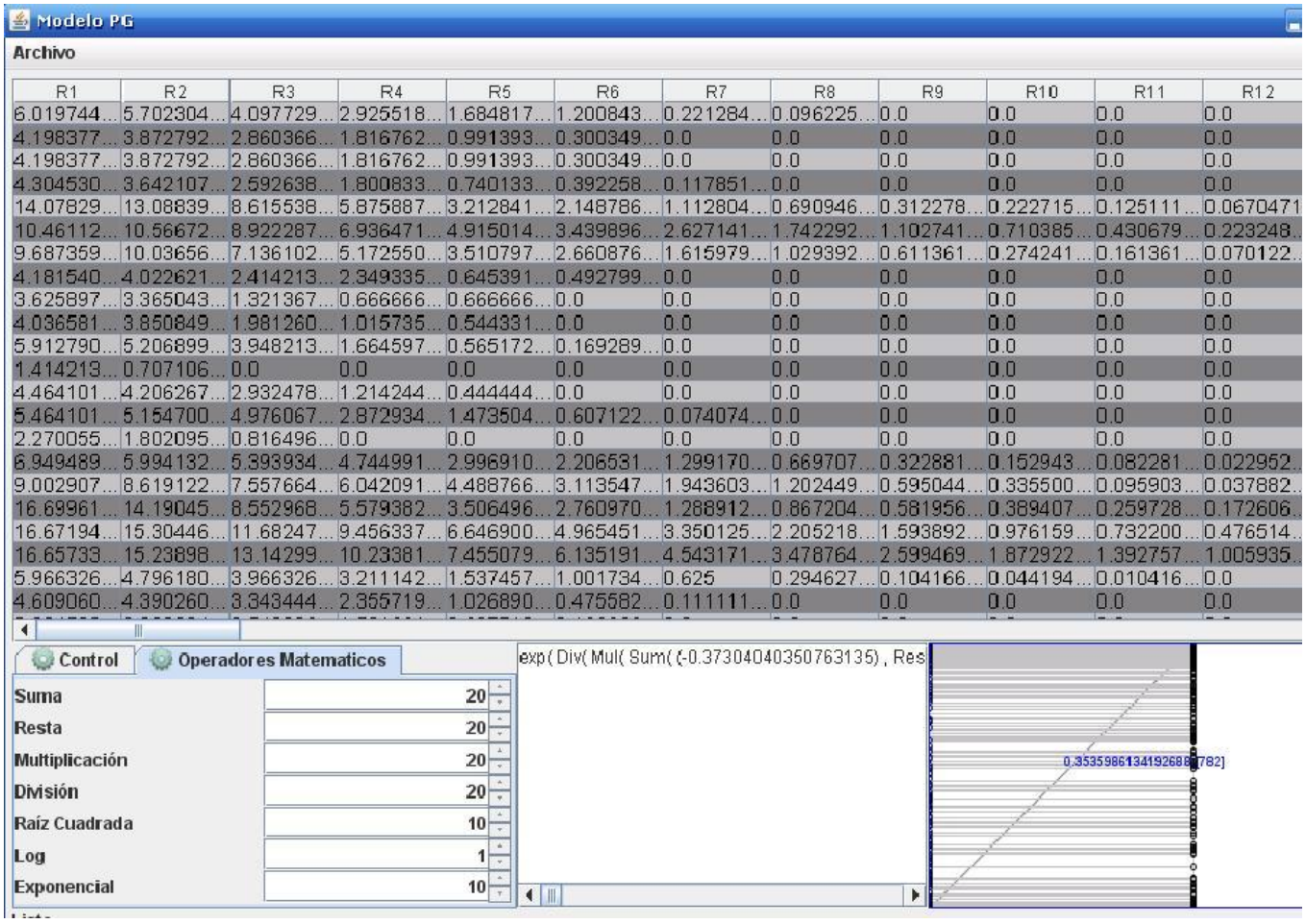


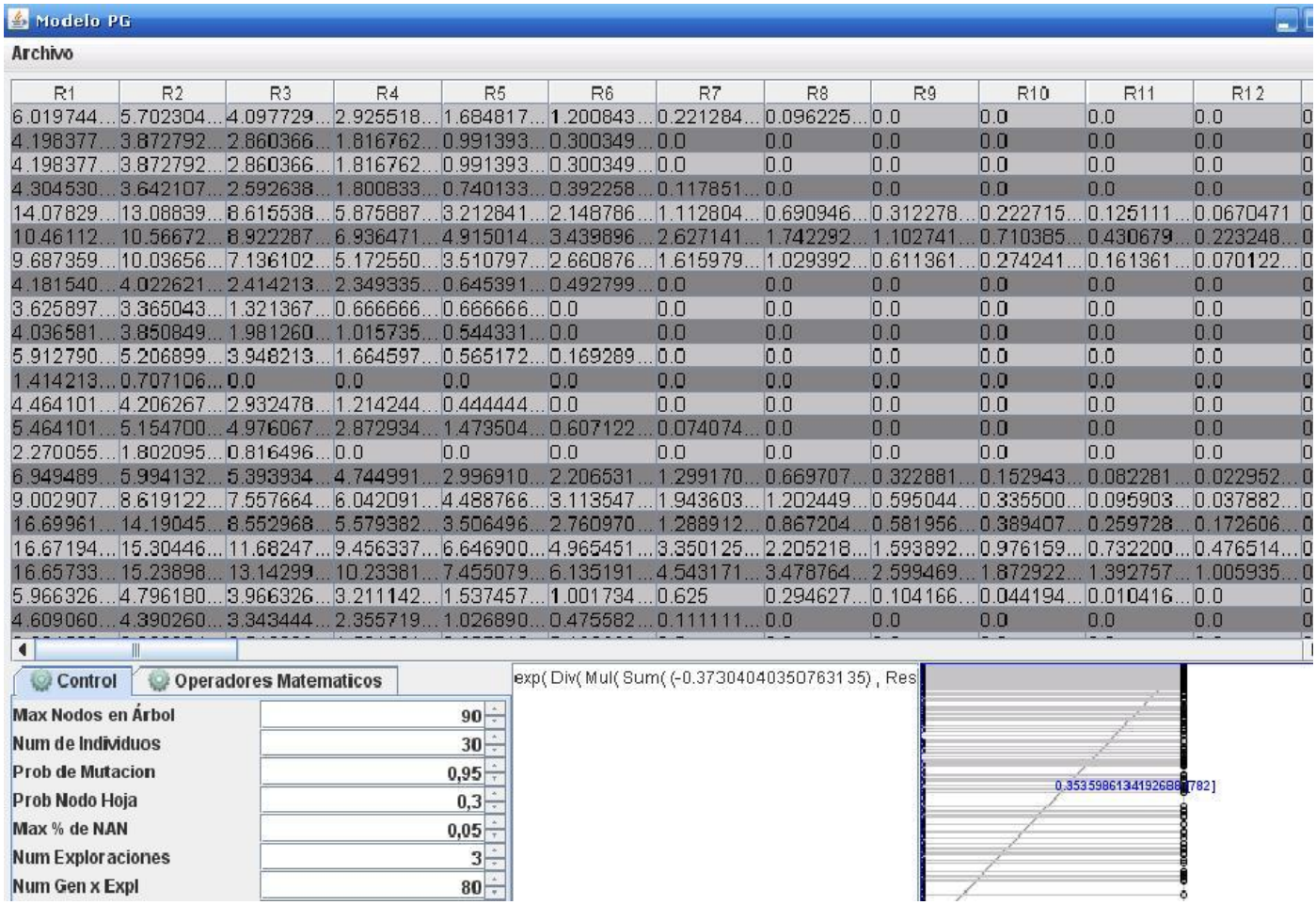
Anexo # 5 Vistas de Salida



VC3	VC4	VPC13	VPC23	VPC33	VPC14	VPC24	VPC34	concentraci...	Resultado_...
0,805	0	1,631	1,158	0,858	0	0	0	1	1
0,805	0	1,631	1,158	0,858	0	0	0	1	1
0,805	0	1,631	1,158	0,858	0	0	0	1	1
0,875	0,062	2,073	1,743	1,272	0,192	0,127	0,096	1	1
0,657	0	1,171	0,541	0,569	0	0	0	1	0,418
0,657	0	1,171	0,51	0,558	0	0	0	1	0,399
1,21	0,031	3,273	2,158	1,296	0,031	0,025	0,016	1	1
1,085	0	2,91	1,974	1,172	0	0	0	1	1
0,834	0,495	2,342	1,442	1,053	0,831	0,528	0,375	1	1
0,993	0,5	2,222	1,471	0,985	0,5	0,25	0,197	1	1
2,705	0	10,973	2,748	1,849	0	0	0	1	1
1,836	0	2,69	2,362	1,2	0	0	0	1	1
1,875	0	3,611	2,49	1,721	0	0	0	1	1
1,107	0	2,36	1,671	1,001	0	0	0	1	1
1,678	0	3,148	1,974	1,305	0	0	0	1	1
1,17	0	2,463	1,734	1,051	0	0	0	1	1
2,856	0	11,254	2,853	1,948	0	0	0	1	1
1,17	0	2,401	1,697	1,035	0	0	0	1	1
0,707	0	1,207	0,354	0,177	0	0	0	1	0,298
0,25	0,044	0,476	0,285	0,242	0,11	0,156	0,024	1	0,295
0,385	0,044	0,79	0,56	0,408	0,133	0,093	0,042	1	1
9,638	0	13,105	18,45	1,724	0	0	0	1	1
0,818	0	1,801	1,371	0,935	0	0	0	1	1
0,818	0	1,846	1,371	0,946	0	0	0	1	1
1,025	0	1,87	1,421	0,822	0	0	0	1	1
1,085	0	2,802	2,271	1,447	0	0	0	1	1
0,515	0	1,038	0,565	0,368	0	0	0	1	1
1,421	0	2,114	1,675	0,917	0	0	0	1	1
0,769	0	1,241	0,545	0,475	0	0	0	1	1
0,852	0	1,163	0,578	0,618	0	0	0	1	1
0,815	0	1,883	1,472	1,007	0	0	0	1	1
0,815	0	1,883	1,477	1,03	0	0	0	1	1

Anexo # 6 Imágenes de la aplicación generando modelo.




Modelo PG
Archivo
R1 R2 R3 R4 R5 R6 R7 R8 R9 R10 R11 R12
6.019744... 5.702304... 4.097729... 2.925518... 1.684817... 1.200843... 0.221284... 0.096225... 0.0 0.0 0.0 0.0
4.198377... 3.872792... 2.860366... 1.816762... 0.991393... 0.300349... 0.0 0.0 0.0 0.0 0.0 0.0
4.198377... 3.872792... 2.860366... 1.816762... 0.991393... 0.300349... 0.0 0.0 0.0 0.0 0.0 0.0
4.304530... 3.642107... 2.592638... 1.800833... 0.740133... 0.392258... 0.117851... 0.0 0.0 0.0 0.0 0.0
14.07829... 13.08839... 8.615538... 5.875887... 3.212841... 2.148786... 1.112804... 0.690946... 0.312278... 0.222715... 0.125111... 0.0670471
10.46112... 10.56672... 8.922287... 6.936471... 4.915014... 3.439896... 2.627141... 1.742292... 1.102741... 0.710385... 0.430679... 0.223248...
9.687359... 10.03656... 7.136102... 5.172550... 3.510797... 2.660876... 1.615979... 1.029392... 0.611361... 0.274241... 0.161361... 0.070122...
4.181540... 4.022621... 2.414213... 2.349335... 0.645391... 0.492799... 0.0 0.0 0.0 0.0 0.0 0.0
3.625897... 3.365043... 1.321367... 0.666666... 0.666666... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4.036581... 3.850849... 1.981260... 1.015735... 0.544331... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5.912790... 5.206899... 3.948213... 1.664597... 0.565172... 0.169289... 0.0 0.0 0.0 0.0 0.0 0.0
1.414213... 0.707106... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
4.464101... 4.206267... 2.932478... 1.214244... 0.444444... 0.0 0.0 0.0 0.0 0.0 0.0 0.0
5.464101... 5.154700... 4.976067... 2.872934... 1.473504... 0.607122... 0.074074... 0.0 0.0 0.0 0.0 0.0
2.270055... 1.802095... 0.816496... 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
6.949489... 5.994132... 5.393934... 4.744991... 2.996910... 2.206531... 1.299170... 0.669707... 0.322881... 0.152943... 0.082281... 0.022952...
9.002907... 8.619122... 7.557664... 6.042091... 4.488766... 3.113547... 1.943603... 1.202449... 0.595044... 0.335500... 0.095903... 0.037882...
16.69961... 14.19045... 8.552968... 5.579382... 3.506496... 2.760970... 1.288912... 0.867204... 0.581956... 0.389407... 0.259728... 0.172606...
16.67194... 15.30446... 11.68247... 9.456337... 6.646900... 4.965451... 3.350125... 2.205218... 1.593892... 0.976159... 0.732200... 0.476514...
16.65733... 15.23898... 13.14299... 10.23381... 7.455079... 6.135191... 4.543171... 3.478764... 2.599469... 1.872922... 1.392757... 1.005935...
5.966326... 4.796180... 3.966326... 3.211142... 1.537457... 1.001734... 0.625 0.294627... 0.104166... 0.044194... 0.010416... 0.0
4.609060... 4.390260... 3.343444... 2.355719... 1.026890... 0.475582... 0.111111... 0.0 0.0 0.0 0.0 0.0
Control
Operadores Matematicos
Max Nodos en Árbol 90
Num de Individuos 30
Prob de Mutacion 0,95
Prob Nodo Hoja 0,3
Max % de NAN 0,05
Num Exploraciones 3
Num Gen x Expl 80
exp(Div(Mul(Sum((-0.3730404035076313 5) , Res
0.3535986134192688 782]

GLOSARIO DE TÉRMINOS

CQF: Centro de Química Farmacéutica (CQF) es una institución para el desarrollo de investigaciones científico-tecnológicas dirigidas hacia la obtención de sustancias bio-activas para uso humano.

Bioinformática: es la aplicación de los ordenadores y los métodos informáticos en el análisis de datos experimentales y simulación de los sistemas biológicos.

CASE: Computer Aided Software Engineering (Herramientas de ingeniería de software asistida por computadora).

Plug-ins: aplicación informática que interactúa con otra aplicación para aportarle una función o utilidad específica.

Compuestos Orgánicos: compuestos cuya composición fundamental es sobre la base del elemento químico carbono.

Actividad Biológica: actividad que caracteriza el comportamiento biológico en compuestos químicos.

Modelo Matemático: expresión numérica que simula la realidad.

Predicción: valor estimado que caracteriza una propiedad o fenómeno obtenido de un modelo.

Concentración Efectiva: cantidad de sustancia concentrada por unidades de masa o volumen.

Descriptores: es una forma de describir las moléculas mediante expresiones matemáticas que permite describir la estructura química o una propiedad de la molécula.

Programación Genética: es un concepto que surge a comienzo de la década del noventa por estudios realizados por John Koza sustentados sobre la base de los modelos de algoritmos genéticos y programación evolutiva. Consiste en la evaluación automática de programas usando ideas basadas en la selección natural.