Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de diploma para optar por el título

Ingeniero en Ciencias Informáticas

FCM – Decision: MÓDULO DISEÑO AUTOMÁTICO DE MAPAS COGNITIVOS DIFUSOS.

Autor: Yosbel Fonseca Mendoza

Tutor: Ing. Oscar Reimar Cedeño Delgado

La Habana, junio de 2013.

"Año 55 de la Revolución"

"¿Casualidad o causalidad? La verdadera fuente de la vida se encuentra en tu propio destino y si crees que éste no es el que tú te mereces, no le des más vueltas porque por casualidad o causalidad tu destino está escrito".

Anónimo

DECLARACIÓN DE AUTORÍA

Declaro ser el único autor de la presente tesis y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

Para que así conste firmo la presente a los ____ días del mes de _____ del año 2013.

_____ _____

Firma del autor Firma del tutor

Yosbel Fonseca Mendoza Ing. Oscar Reimar Cedeño Delgado

DATOS DE CONTACTO:

Tutor:

Ing. Oscar Reimar Cedeño Delgado: Profesor graduado de Ingeniero en Ciencias Informáticas en el año 2008 en la UCI. Categoría: Instructor. Ha impartido las asignaturas Programación 2, Programación 4, Programación 5 y Práctica Profesional. Se desempeña como Jefe de Desarrollo del proyecto Contraloría General de la República (AUDAT).

Email: orcedeno@uci.cu

Autor: Yosbel Fonseca Mendoza

Universidad de las Ciencias Informáticas

La Habana, Cuba

Email: yfonseca@estudiantes.uci.cu

AGRADECIMIENTOS:

Es mi deseo, como sencillo gesto de gratitud, agradecer:

A Dios por acompañarme uno y cada uno de los días de la carrera proporcionándome fortaleza, fe, salud y perseverancia para alcanzar este anhelo que se vuelve una realidad tangible.

A mi abuela Viltrudis, que Dios te bendiga todos los días abuela, por permanecer conmigo, en mis contraseñas, desde la primera hasta la última, en la firma del correo, en mi corazón los días buenos y malos y en los que no fueron ni buenos ni malos. Muchas Gracias por tu tiempo, paciencia y amor, por compartir conmigo todos los valores que te hacen tan especial, por ayudarme a crecer y ser ejemplo en todos los sentidos y aspectos, capaz de darlo todo sin recibir nada, por todo esto y por ser mi Abuela: "Muchas Gracias".

A mis padres, quienes creyeron siempre en mí y permanentemente me aconsejaron con espíritu alentador, contribuyendo incondicionalmente a lograr mis metas y objetivos propuestos.

A mi hermana, por todas sus oraciones. Por ser especial, aceptarme a pesar de todas mis faltas y permanecer formándose para convertirse en la próxima profesional de la familia. Estoy muy orgulloso de ti mi hermana, sé que lo lograrás, te digo como me dijiste un día: "no temas, ni desmayes, porque Dios estará contigo dondequiera que vayas".

A mi familia, por brindarme apoyo en todo momento. Especialmente a mi tío Elio, por ser el motor impulsor que me brindó fuerzas cuando más la necesitaba. Al nuevo integrante, mi cuñado Pacheco, porque verdaderamente, sin él, este resultado habría sido más difícil de alcanzar.

A mis amistades más cercanas, del barrio: Eduardo Luis (Bolo) y Liudmila, y de la UCI: Yudelis, Maricel, Yanet y Graciela por su compañía, sus buenos consejos y por dedicar siempre una palabra de aliento cuando las he necesitado.

Agradecimientos

A mis compañeros, con quienes he compartido durante la carrera: los integrantes de los grupos 6101, 6205, 6406 y 6506, los muchachos del secretariado de la FEU, la gente del monte: Tito, Franklin, el Yoss y Flores, a PMM. A ustedes gracias por su confianza, amistad y por esos momentos maravillosos que disfrutamos.

A los maestros y profesores que me han acompañado durante el largo camino, brindándome siempre su orientación con profesionalismo ético en la adquisición de conocimientos y afianzando mi formación como estudiante. Especialmente al profe Omar Mar, por ser como un padre para mí, por brindarme su ayuda cuando la he necesitado. A los profesores del Consejo de Dirección del centro por haber compartido conmigo estos dos años de trabajo en la FEU. A mi tutor, mi oponente y al tribunal por sus valoraciones, observaciones y sugerencias durante el desarrollo de este trabajo.

Quiero hacer llegar un agradecimiento especial a varias personas que facilitaron la realización del presente trabajo, ellos son: Alien, Adrián Rosales, Jorge Deinier Sosa, Rolando, Jorge Ernesto, Yami y Baby. A ustedes un Terabyte de Gracias.

He querido dejar para el final este último párrafo, no porque sea menos importante, al contrario, quiero agradecerte a ti, Eilen, por ser novia, tutora, oponente, tribunal y pilar fundamental para afianzar este sueño, siendo partícipe activa de su creación, mil gracias Eilen, sin temor a equivocarme quiero expresar que este resultado también es tuyo.

*A todos, **¡MUCHAS GRACIAS!***

DEDICATORIA:

Dedico este resultado a mi abuela Viltrudis, a mi mamá y mi papá, por confiar en que lograría este sueño y ser el pilar fundamental en toda mi educación, tanto académica, como de la vida, por su apoyo absoluto perfectamente mantenido a través del tiempo, por sus consejos, su ejemplo de perseverancia y constancia, sus valores, por la motivación constante que me ha permitido ser una persona de bien, pero más que nada, por su amor incondicional.

Este resultado es, sin dudas, para y gracias, a ustedes.

RESUMEN

Los modelos causales son instrumentos empleados frecuentemente para comprender los sistemas complejos. Un marco de trabajo fundamental para la inferencia causal son los Mapas Cognitivos Difusos. En la actualidad se evidencian carencias en las funcionalidades de las herramientas que construyen modelos para el razonamiento causal automático basado en Mapas Cognitivos Difusos, incluyendo el soporte a su ciclo de vida: construcción, modelado y análisis. Para erradicar estas dificultades se desarrolló el módulo para el trazado de los Mapas Cognitivos Difusos a partir de una matriz de adyacencia importada por el usuario, lo cual proporciona un área de trabajo que permite visualizar los conceptos representados y analizar sus características principales. El módulo se desarrolló utilizando tecnologías libres, multiplataformas y como patrón arquitectónico el Modelo Vista Controlador. Se utilizó Java como lenguaje de programación, *Netbeans 7.2.1* como entorno de desarrollo y OPEN UP como metodología para guiar el proceso de desarrollo de software. Con el fin de garantizar la calidad que requiere el proceso de desarrollo de software del módulo se elaboraron pruebas para validar el correcto funcionamiento de las funcionalidades implementadas.

Palabras clave: mapas cognitivos difusos, modelado, razonamiento causal, simulación.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

La Inteligencia Artificial (IA) se encarga de modelar la inteligencia humana en sistemas computacionales. Es un área de investigación en la cual se desarrollan algoritmos y funciones para controlar procesos y objetos que proporcionan ayuda a las personas. Utiliza varias técnicas para incorporar a los sistemas informáticos conocimientos u otras características de seres inteligentes. Cuantiosas son las aplicaciones que se han desarrollado en líneas de investigaciones científicas, tales como, la robótica, la visión artificial, técnicas de aprendizaje y la gestión del conocimiento. Estas aplicaciones incorporan factores y relaciones del mundo real y del ámbito del conocimiento, constituyendo una base teórica que se elabora a través del razonamiento causal.

El razonamiento lógico o causal es una operación lógica mediante la cual, partiendo de una o más situaciones, se deriva la validez, posibilidad o falsedad de otra operación lógica distinta. Por lo general, las situaciones en que se basa un razonamiento expresan conocimientos adquiridos o, por lo menos, postulados como hipótesis. El razonamiento causal se utiliza para justificar o aportar razones en favor de lo que se conoce o se cree conocer y, en algunos casos, permite demostrar lo que se sabe. Es útil en la toma de decisiones por dos razones fundamentales: primero, es natural y fácil de entender; segundo, es convincente porque explica el por qué se llega a una conclusión particular. (1)

Para considerar la causalidad desde un punto de vista computacional existen dos marcos de trabajo fundamentales: las Redes Bayesianas (RB) y los Mapas Cognitivos Difusos (MCD). (2) Las RB se crearon como modelo probabilístico para el razonamiento con incertidumbre en Inteligencia Artificial, lo que permitió refutar las objeciones contra el uso de la probabilidad, construyendo un modelo de razonamiento causal con un sólido fundamento teórico. Permiten seleccionar sólo las variables que tienen relaciones causales para el cálculo de las probabilidades condicionadas. Estas ofrecen un modelo apropiado para caracterizar la causalidad en términos de probabilidades condicionales. Sin embargo, presentan limitaciones para manejar la existencia de ciclos en las relaciones causales y no permiten determinar de manera exacta las probabilidades.

Los MCD fueron ideados por el escritor y profesor de ingeniería eléctrica y derecho *Bart Kosko Andrew* (3) como una extensión de los mapas cognitivos. (4) Constituyen una estructura de grafo difuso utilizado para representar razonamiento causal. Su aplicación resulta recomendable cuando es necesario representar el

grado de influencia entre conceptos y resulta difícil encontrar una relación probabilística. Además son muy usados para los dominios donde los conceptos y las relaciones son fundamentalmente difusos, como la política, la historia y la planificación estratégica. Con la utilización de los MCD se obtienen los beneficios de modelado visual, la simulación y la predicción (4), modelando el sistema con grados difusos de causalidad en el intervalo [-1, 1].

En correlación con lo antes planteado los MCD proveen esquemas más realistas para la representación del conocimiento con respecto a las RB. (5) Entre los elementos que permiten una representación más realista del conocimiento se encuentra la posibilidad de representar retroalimentación, el tiempo, la vaguedad y la ambigüedad. Para la representación gráfica de los MCD se utilizan grafos dirigidos donde los nodos son conceptos causales y pueden modelar eventos, acciones, valores, metas o procesos; y los arcos indican relación causal.

Los MCD han sido aplicados en un elevado número de dominios, entre los cuales se puede destacar: el análisis de los fallos en la calidad del agua, la formulación de la estrategia financiera y la visión artificial. En el área de la ingeniería de software se destaca su empleo en la simulación de proyectos de desarrollo de software y el análisis de riesgos en el mantenimiento de la planificación de los recursos empresariales. La poca disponibilidad de herramientas, tanto comerciales como libres, que brinden soporte a esta técnica es una de sus limitaciones.

La representación de un MCD consiste en encontrar para cada vértice, una posición en el plano o área de trabajo. La automatización del dibujo de grafos tiene importancia en materias claves como: la Ingeniería de Software, la Visualización de Información, Minería de Datos y la Representación del Conocimiento. Para la implementación del esbozo de grafos se utilizan métodos algorítmicos; estos cumplen con criterios estéticos (minimizar los cruces, posicionamiento uniforme de vértices, longitud de las aristas, maximizar la simetría del trazado y minimizar el área de trabajo), con el fin de lograr una mejor visualización del mismo. Los algoritmos de visualización reciben como entrada el grafo a visualizar y brindan como salida el mismo grafo pero trazado estéticamente cumpliendo con los criterios anteriormente mencionados.

Una de las tendencias utilizadas para el dibujo de grafos es el uso de la matriz de adyacencia, es una de las formas más sencillas, donde las filas y columnas se identifican con los vértices del grafo; mientras que en las celdas se indica con un valor numérico el peso de la relación entre dos vértices.

En la actualidad existen herramientas que permiten la representación de grafos a partir de la matriz de adyacencia. Estas no cuentan con todas las funcionalidades para la construcción de los modelos para el razonamiento causal automático. En la Universidad de las Ciencias Informáticas (UCI) se encuentra en desarrollo la aplicación de escritorio *FCM – Decision* que tiene como objetivo modelar y analizar los MCD e incluye el soporte al ciclo de vida de los modelos.

FCM – Decision emplea en su desarrollo un lenguaje multiplataforma y cuenta con una interfaz principal para el modelado de los MCD cuya filosofía es: "dibujar, modelar y analizar". La aplicación brinda al usuario la posibilidad de esbozar libremente un grafo de forma manual, mostrando mensajes de advertencia en caso de operaciones satisfactorias u ocurrencia de errores. A partir del dibujo del grafo permite visualizar su matriz de adyacencia con las relaciones existentes entre cada uno de los nodos; posibilitando la edición de estos, así como del peso y la dirección de las aristas. Sin embargo con la visualización de esta matriz termina el ciclo de vida de los modelos, siendo imposible realizar la operación inversa y obtener el modelado de un MCD a través de una matriz de adyacencia. Como parte del modelado automático, resulta engorroso para los usuarios de *FCM – Decision* realizar el trazado manual de un grafo con más de 20 nodos y sus respectivas relaciones.

Analizando la situación anterior se identifica como problema de la investigación: ¿Cómo contribuir a la visualización efectiva de los Mapas Cognitivos Difusos a partir de su matriz de adyacencia?

Teniendo en cuenta el problema de la investigación se ha definido como objeto de estudio: Los Mapas Cognitivos Difusos.

El estudio se centra en el campo de acción definido por: Visualización de Mapas Cognitivos Difusos.

Con el propósito de solucionar los problemas encontrados se ha determinado como objetivo de la investigación: Desarrollar el módulo diseño automático de Mapas Cognitivos Difusos para la herramienta de modelado y análisis *FCM - Decision*.

Del mismo se derivan los siguientes objetivos específicos:

1. Realizar el análisis y diseño del módulo diseño automático de Mapas Cognitivos Difusos.
2. Implementar el módulo diseño automático de Mapas Cognitivos Difusos.
3. Evaluar el funcionamiento del módulo diseño automático de Mapas Cognitivos Difusos.

Para dar cumplimiento al objetivo planteado, se definen las siguientes tareas de la investigación:

1. Revisión bibliográfica para la selección de las herramientas, metodologías y mecanismos para el modelado de Mapas Cognitivos Difusos.
2. Realización del estudio de la teoría de grafos con el objetivo de adquirir los conocimientos necesarios para la implementación del módulo.
3. Definición del estándar de entrada para la matriz a graficar.
4. Implementación del módulo para el modelado de Mapas Cognitivos Difusos.
5. Realización de pruebas al módulo implementado.
6. Integración del módulo a la herramienta *FCM – Decision*.

Con el desarrollo del módulo diseño automático de Mapas Cognitivos Difusos, *FCM – Decision* contará con un flujo para la importación de datos y con funcionalidades para obtener a partir de una matriz de adyacencia, el modelado del mapa cognitivo difuso correspondiente.

Este documento se encuentra estructurado en tres capítulos, los cuales se describen a continuación:
CAPÍTULO 1: FUNDAMENTOS TEÓRICOS: En este capítulo se realiza un estudio de los conceptos y software relacionados con el modelado automático de grafos difusos. Como el desempeño de un proyecto está antecedido por la investigación de las tecnologías, se describen las herramientas y tecnologías utilizadas en el desarrollo de la investigación.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA: En este capítulo se realiza el modelado y estructuración del sistema a través del modelo de diseño, en el cual se aplican las buenas prácticas definidas en los patrones de diseño. Además se realiza, teniendo en cuenta la configuración y distribución de los nodos de cómputo, el modelo del sistema y el diagrama de despliegue.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA: En este capítulo se implementan las clases y subsistemas en términos de componentes y se presenta la propuesta de solución del sistema. De igual manera, se explica el proceso de validación del sistema a través de los casos de prueba aplicados al módulo.

CAPÍTULO 1: FUNDAMENTOS TEÓRICOS

En el presente capítulo se describen los conceptos básicos que se relacionan con el objeto de estudio y campo de acción de la investigación. Se caracterizan los diferentes sistemas de diseño automático de Mapas Cognitivos Difusos y se fundamenta la selección de la metodología, tecnologías y herramientas a utilizar para el desarrollo de la aplicación.

1.1- Principales conceptos

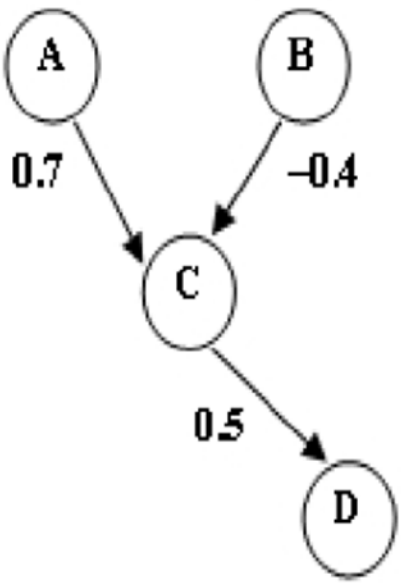
Inteligencia artificial: es la ciencia que estudia y analiza el comportamiento humano. De esta manera, las aplicaciones de la IA se sitúan principalmente en la simulación de actividades intelectuales del hombre. Es decir, imitar por medio de máquinas, normalmente electrónicas, tantas actividades mentales como sea posible, y quizás llegar a mejorar las capacidades humanas en estos aspectos.

Causalidad: es el fenómeno mediante el cual se relacionan causas con efectos. Es la conexión que existe entre las razones o las causas de ciertos fenómenos o procesos y los resultados o efectos de los mismos. La noción de causalidad implica una permanente relación entre un evento anterior y su continuación, además de formarse así un círculo infinito de conexión entre sucesos y eventos que se generan unos a otros. (6)

Mapas Cognitivos Difusos: fueron introducidos por Bart Kosko, quien le da significado a esta nueva representación como un grafo capaz de codificar conocimiento empleando lógica difusa. Constituyen una técnica de representación causal, compuesta por conceptos y relaciones causales, los cuales usan la teoría de lógica difusa para describir su estructura. Mejoran los mapas cognitivos al describir la fortaleza de la relación mediante el empleo de valores borrosos en el intervalo [-1,1]. (7)

Algoritmo: es una secuencia finita de instrucciones que se utiliza para ejecutar una tarea o resolver un problema determinado. (8)

Matriz de adyacencia: sea G un grafo dirigido con un conjunto de nodos o vértices V = {1, 2,…n} y un conjunto de arcos o aristas. La matriz de adyacencia de G es la matriz M: n x n en la cual el elemento en la fila i y columna j es el peso de la arista que va desde el vértice i al vértice j. Cada vértice representa un conjunto difuso o evento que ocurre en algún grado (figura 1).



	A	B	C	D
A	0	0	0.7	0
B	0	0	–0.4	0
C	0	0	0	0.5
D	0	0	0	0

Figura 1: Mapa Cognitivo Difuso y su matriz de adyacencia

En los MCD existen tres posibles tipos de relaciones causales entre conceptos:

- $W_{ij} > 0$, indica una causalidad positiva entre los conceptos C_i y C_j. Es decir, el incremento (o disminución) en el valor de C_i lleva al incremento (o disminución) en el valor de C_j.
- $W_{ij} < 0$, indica una causalidad negativa entre los conceptos C_i y C_j. Es decir, el incremento (o disminución) en el valor de C_i lleva a la disminución (o incremento) en el valor de C_j.
- $W_{ij} = 0$, indica la no existencia de relación entre Ci y Cj.

1.2- Sistemas vinculados al campo de acción

Antes de iniciar el desarrollo de un proyecto de software es muy importante realizar un estudio detallado de los sistemas existentes tanto nacionales como internacionales. En la actualidad existen numerosos sistemas que brindan soluciones informáticas para el modelado de Mapas Cognitivos Difusos. El estudio de estos sistemas constituye una guía para la investigación y a través del mismo se pueden realizar análisis en cuanto a sus funcionalidades y arquitectura. A continuación se muestran algunos de estos sistemas.

Sistemas nacionales:

1.2.1- Sistema para la reducción de grafos y búsqueda de camino mínimo del Proyecto SIG – Desktop

En el centro de Desarrollo de Geoinformática y Señales Digitales (GEySED) de la UCI, el proyecto SIG – Desktop cuenta con un Sistema de Información Geográfica (SIG), en el mismo se utiliza el modelo matemático grafo para realizar análisis de redes. El sistema está compuesto por dos módulos, uno para la obtención del grafo que representa una red de viales y otro para la reducción de grafos y búsqueda de caminos mínimos sobre el grafo, reducido de forma eficiente.

El módulo de obtención de grafos facilita un modelo matemático equivalente a la red de viales de un mapa, en este caso un grafo dirigido y no ponderado. A partir de la cartografía de entrada de este módulo se obtienen los nodos y las aristas del grafo, la misma debe estar almacenada en una base de datos PostgreSQL. Para dibujar el grafo, implementa un algoritmo ávido con el objetivo de encontrar las intersecciones entre las calles de la red y los puntos iniciales y finales de cada calle, obteniendo así los nodos del grafo. El sistema se desarrolló haciendo uso de herramientas libres. Como lenguaje de programación utiliza Java y para exportar una cartografía en formato shape a PostgreSQL utiliza la herramienta shp2psql.

1.2.2- Herramienta de creación de grafos de caminos para la biblioteca "SceneToolkit"

En el año 2008 el departamento de Entornos Virtuales de la facultad 5 en la UCI, presentó un trabajo de diploma titulado: Herramienta de creación de grafos de caminos para la biblioteca "*SceneToolkit*". La herramienta está implementada utilizando como el lenguaje de programación C++, siguiendo los principios de la programación orientada a objetos. Las interfaces gráficas basadas en *widgets* están diseñadas en la aplicación QT *Designer* y utiliza la biblioteca *SceneToolkit* para el tratamiento de entornos virtuales, la que a su vez hace uso de la biblioteca gráfica libre de más bajo nivel OpenGL. Para construir la aplicación visual utiliza la biblioteca de clases QT mientras que el *framework* STK Editor se encarga de la vinculación QT – STK. Para la representación de las estructuras de datos, hace uso de la STL de C++. El sistema permite además importar y exportar ficheros de mundo 3D y en formato xml respectivamente, gestiona nodos y aristas, convirtiéndolos en grafos dirigidos y no ponderados y aplica algoritmos de planificación de movimientos a grafo de caminos.

Sistemas internacionales:

1.2.3- MaGraDa

El paquete de software *MaGraDa* (Grafos para Matemática Discreta) es una aplicación informática programada en lenguaje Java utilizando como herramienta de desarrollo el Netbeans en su versión 6.9. Está diseñada específicamente para trabajar con grafos dirigidos, no dirigidos, ponderados y no ponderados. La filosofía de *MaGraDa* permite representar grafos con un máximo de 50 vértices. Este paquete es sencillo y cómodo de manejar, consta de dos pantallas de visualización: una en modo texto donde se trabaja con los datos del grafo y otra en modo gráfico en la que los grafos se pueden observar gráficamente.

Los grafos se crean a través de la interacción del usuario con la aplicación, quien puede crear un grafo manualmente o a través de una matriz de adyacencia y además puede abrir un grafo creado anteriormente con *MaGraDa*. Con esta se pueden averiguar una serie de características o propiedades básicas, tales como: grado de un vértice, matriz de adyacencia o pesos y ver aristas (o arcos) del grafo. Dispone de algoritmos muy conocidos en el mundo de los grafos tales como, *Dijkstra*, *Floyd – Warshall, Kruskal* y *Prim*. *MaGraDa*, por ser un proyecto sencillo, no se encarga de garantizar que los trazados de los grafos sean estéticamente buenos, por lo que pueden existir solapamientos de nodos y cruces entre aristas.

1.2.4- Técnicas de visualización para crawlers. Jwire, un caso práctico

Jwire está implementado en Java, utilizando el *Netbeans* 6.9 y la versión 6.0 de la Máquina Virtual de Java. Aprovecha, mediante la interfaz nativa de Java, de bibliotecas de funciones o *APIs* externas, implementadas en Java, o en otros lenguajes de programación, en particular en C y C++. Utiliza varias librerías para la visualización de los grafos representados.

El sistema posee dos módulos principales, Ejecución y Visualización. El módulo Ejecución se encarga de ejecutar y procesar salidas de los diferentes comandos de *WIRE* (proyecto desarrollado bajo licencia GPL por el Centro de Investigación de la Web, perteneciente al departamento de Ciencias de la Computación en la Universidad de Chile). Este es capaz por sí solo de rastrear y extraer información de internet, accediendo a múltiples sitios web, es decir, puede rastrear un dominio de un país completo si le brindan los recursos computacionales y el tiempo necesario. El objetivo del proyecto, es crear una aplicación para obtener información sobre la web. (9)

El módulo Visualización obtiene los datos recogidos por *WIRE* tras su ejecución y visualiza estos datos en forma de grafos no dirigidos y sin ponderación. Aporta herramientas para el análisis visual de los grafos representados y permite guardar o procesar los datos recogidos.

Los datos recogidos por *WIRE*, son recursos web extraídos de las redes rastreadas. Los recursos web accedidos son, en su mayor parte, páginas web o direcciones web relacionadas entre sí mediante hiperenlaces. En la representación del grafo, los nodos representarían los recursos web accedidos y las aristas serían los hiperenlaces que relacionan dichos recursos. (9) La aplicación realiza trazados de grafos, estéticamente buenos, aun cuando los grafos tienen más de 100 nodos.

1.2.4- Análisis de los sistemas estudiados

A continuación se enuncian los principales aspectos a considerar en los sistemas analizados para determinar si alguno de ellos cumple las condiciones necesarias para utilizarlo como solución al problema planteado.

1. **Modelo matemático grafo:** Refleja si la aplicación estudiada utiliza en sus funcionalidades el modelo matemático grafo.

2. **Tipo de grafo:** Recoge el tipo de grafo que utiliza la aplicación (dirigido, no dirigido, ponderado, no ponderado).

3. **Matriz de adyacencia:** Establece si la aplicación realiza el trazado del grafo a partir de una matriz de adyacencia.

4. **Máximo de nodos:** Establece la cantidad máxima de nodos que permite la aplicación en la construcción del grafo.

5. **Visualización:** Establece si el trazado del grafo posee estética o no, es decir, si existe solapamiento de nodos o cruces entre aristas.

6. **Herramienta de desarrollo:** Establece las herramientas con las cuales se desarrolló la aplicación estudiada.

7. **Lenguaje de programación:** Se especifica el lenguaje de programación con el cual se desarrolló la aplicación estudiada.

Sistemas\Criterios	1	2	3	4	5	6	7
A	Sí	Dirigido y No ponderado	No	Se desconoce	Sí	MPE Neo4j, extensión PostGIS de PostgreSQL	Java
B	Sí	No Dirigido, No ponderado	No	Se desconoce	Sí	QT Designer, biblioteca SceneToolkit, framework STK Editor	C++
C	Sí	Dirigido, No Dirigido, Ponderado y No ponderado	Sí	50	No	Netbeans 6.9	Java
D	Sí	No Dirigido, No ponderado	No	Mayor que 100	Sí	Netbeans 6.9	Java

Tabla 1. Análisis de los sistemas estudiados

Leyenda de la tabla:

- Los números representan los criterios comparativos descritos anteriormente.
- Las letras representan cada uno de los sistemas analizados como se muestra a continuación:
  A. Sistema para la reducción de grafos y búsqueda de camino mínimo del Proyecto SIG – Desktop.
  B. Herramienta de creación de grafos de caminos para la biblioteca “SceneToolkit”.
  C. MaGraDa.
  D. Técnicas de visualización para crawlers. Jwire, un caso práctico.

Los sistemas analizados ofrecen una guía para el desarrollo del módulo pues utilizan los grafos como modelo matemático para resolver determinado problema. Sin embargo luego de realizar el análisis de los sistemas estudiados se resume que ninguno cumple con las condiciones necesarias para ser integrado a la aplicación *FCM – Decision* y así solucionar a la problemática planteada en la investigación.

Luego de este análisis, con el objetivo de encontrar una solución a la problemática, se estudiaron dos librerías: *Prefuse* y JUNG, realizadas en Java para la creación interactiva de aplicaciones de visualización de la información. Ambas soportan un amplio conjunto de características para el modelado de datos, visualización e interacción. Proporcionan estructuras de datos optimizadas para tablas, grafos y árboles, técnicas de codificación visual y soporte para animación, consultas dinámicas, búsqueda integrada y conectividad de bases de datos.

La utilización de las librerías anteriormente mencionadas es una excelente elección si se fuese a realizar una aplicación desde el inicio pero en este caso fueron desechadas pues su uso en la realización del módulo cambiaría toda la arquitectura de la aplicación *FCM – Decision.*

1.3- Tendencias y tecnologías actuales a considerar

A continuación se describe la metodología, herramientas y tecnologías a utilizar, así como sus principales características.

El desempeño de un proyecto está antecedido por la investigación de las tecnologías de punta que se utilizan, así como las ventajas y desventajas de su aplicación. Para la realización de esta investigación fueron analizadas las tecnologías existentes que se ajustan a la solución a desarrollar.

1.3.1- Metodología de desarrollo del software

El desarrollo de software puede ser un reto para los desarrolladores, pues crear uno en el menor tiempo posible y con calidad, puede ser casi imposible si no se cuenta con un proceso que ayude a agilizar su desarrollo. La tendencia actual es el uso de metodologías, estas proponen un proceso disciplinado para el desarrollo de software con el fin de hacerlo más predecible y eficiente.

Dentro de las metodologías de desarrollo de software se encuentra Open UP seleccionada por ser una metodología ágil, diseñada para un equipo de desarrollo reducido. Además por ser extensible, con iteraciones cortas y permitir mantener la filosofía de *Rational Unified Process* (RUP) la cual es una de las más empleadas actualmente en la UCI. Esta metodología excluye la mayoría de las piezas opcionales de RUP, y muchos artefactos han sido fusionados. El resultado final es un proceso mucho más sencillo que sigue siendo fiel a los principios de RUP.

La metodología Open UP - en inglés: *Open Unified Process* - es una parte del marco de desarrollo Eclipse. Esta metodología es un proceso unificado que aplica propuestas de gestión ágil, tratando de ser manejable en relación con RUP pues mantiene sus características esenciales: centrado en la arquitectura, dirigido por Casos de Uso (CU) y desarrollo iterativo e incremental, dentro del ciclo de vida de un proyecto de software. No provee lineamientos para todos los elementos que se manejan en un proyecto, pero tiene los componentes básicos que pueden servir de base a procesos específicos. La mayoría de los elementos de Open UP están declarados para fomentar el intercambio de información entre los equipos de desarrollo y mantener un entendimiento compartido del proyecto, sus objetivos, alcance y avances. (10)

En un proyecto organizado sobre Open UP, los esfuerzos personales se convierten en micro incrementos. Estos proveen un ciclo de retroalimentación relativamente corto que permite flexibilidad y mejor adaptación a las decisiones tomadas dentro de cada iteración. Esta metodología divide el proyecto en iteraciones que son planeadas sobre intervalos de tiempo definidos en semanas. Estas iteraciones se centran dando cumplimiento a los objetivos definidos previamente en el plan para cada una por parte del equipo de desarrollo. En cada ciclo iterativo se debe concebir como resultado un demo o un ejecutable con funcionalidades específicas. (11)

Open UP en cada iteración del ciclo de vida, incrementa progresivamente los objetivos de las iteraciones anteriores, añadiendo nuevas funcionalidades a las versiones estables del software que se tiene hasta el momento. Dentro del ciclo de vida existen cuatro fases: Inicio, Elaboración, Construcción y Transición (ver figura 2).

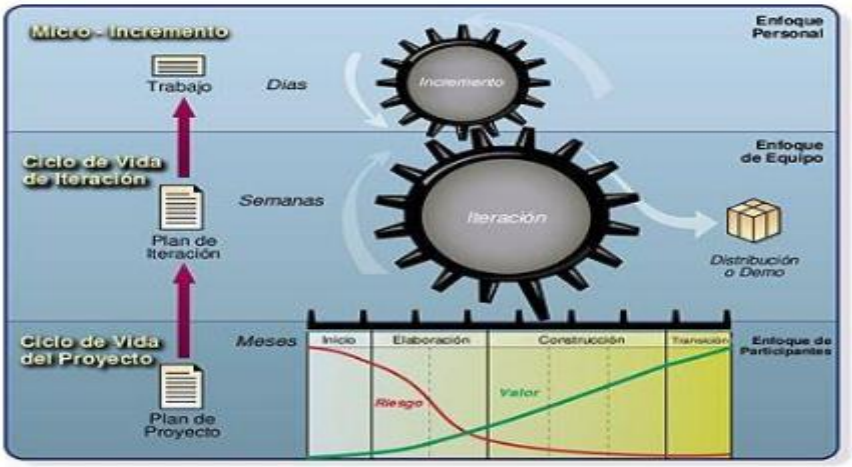


Figura 2: Ciclo de vida de OPEN UP

Beneficios de OPEN UP:

- Es apropiado para proyectos pequeños y de bajos recursos.
- Permite disminuir las probabilidades de fracaso en los proyectos pequeños e incrementar las probabilidades de éxito.
- Permite detectar errores tempranos a través de un ciclo iterativo.

- Evita la elaboración de documentación, diagramas e iteraciones innecesarios requeridos en la metodología RUP. (11)

Después de haber analizado las características que propone la metodología de desarrollo OPEN UP, se decide utilizarla pues proporciona una comprensión detallada del proyecto de software, beneficiando a clientes y desarrolladores sobre productos a entregar y su formalidad. Se centra en una arquitectura temprana para reducir al mínimo los riesgos y organizar el desarrollo.

Lenguaje de modelado:

Una exigencia de la mayoría de las instituciones es que los desarrollos de software se formalicen con un lenguaje estándar y unificado, es decir, se requiere que cada una de las partes que comprende el desarrollo de todo software, se visualice, especifique y documente con lenguaje común. Un lenguaje unificado que cumple con estos requisitos, es ciertamente UML - en inglés: *Unified Modeling Language* -, a continuación se ofrecen algunas de sus características.

UML es utilizado para visualizar, especificar, construir y documentar los artefactos del software. El modelado visual ayuda a mantener la consistencia entre los artefactos del sistema: requisitos, diseños e implementaciones, por tanto, ayuda a mejorar la capacidad del equipo para gestionar la complejidad del software.

UML es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad, es expresivo, claro y uniforme, que no garantiza el éxito de los proyectos pero si mejora sustancialmente el desarrollo de los mismos, al permitir una nueva y fuerte integración entre las herramientas, los procesos y los dominios. Utiliza diagramas y una semántica bien definida con el propósito de elaborar los artefactos de un sistema a través de las distintas etapas de su ciclo de vida, principalmente durante el análisis y diseño del mismo. Se puede aplicar en el desarrollo de software entregando variedad de formas para dar soporte a una metodología de desarrollo, pero no especifica en sí mismo qué metodología o proceso utilizar.

UML tiene como características:

- Tecnología orientada a objetos.
- Viabilidad en la corrección de errores.

- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura pruebas y versiones).
- Cubre las cuestiones relacionadas con el tamaño propio de los sistemas complejos y críticos.
- Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar. (12)

Se decidió utilizar UML como lenguaje de modelado, pues este, además de las ventajas mencionadas anteriormente, permite tener mayor rigor en la especificación, realizar la verificación y validación del modelo desarrollado, automatizar determinados procesos y generar código a partir de los modelos y a la inversa. Esto último permite que el modelo y el código estén actualizados.

1.3.2- Herramienta de modelado

El desarrollo de software es un proceso complejo, desde su concepción hasta el despliegue del producto final. La herramienta de modelado Visual Paradigm está basada en UML y se ha convertido en una parte esencial para reducir esa complejidad. A continuación se ofrecen algunas de sus ventajas.

Visual Paradigm 6.4: es una herramienta multiplataforma de modelado visual muy potente, fácil de utilizar y no emplea una metodología en específico. Soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientado a objetos, construcción, pruebas y despliegue. El software de modelado UML contribuye al desarrollo de aplicaciones con calidad.

Esta herramienta ofrece:

- Entorno de creación de diagramas para UML 2.1.
- Diseño centrado en casos de uso y enfocado al negocio que genera un software de mayor calidad.
- Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- Capacidades de ingeniería directa (versión profesional) e inversa.
- Modelo y código que permanece sincronizado en todo el ciclo de desarrollo.
- Disponibilidad de múltiples versiones, para cada necesidad.

- Disponibilidad de integrarse en los principales Entornos de Desarrollo Integrado (IDEs).
- Disponibilidad en múltiples plataformas. (13)

Además de las ventajas citadas anteriormente, es válido resaltar que se seleccionó Visual Paradigm 6.4 porque soporta las últimas versiones del Visual Paradigm, es una herramienta multiplataforma y se integra fácilmente con el lenguaje Java, aspecto importante en la realización del presente trabajo.

1.3.3- Entorno de Desarrollo Integrado (IDE)

Para mejorar la productividad del desarrollador son necesarios los IDEs, éstos ofrecen un conjunto de funcionalidades totalmente cohesionadas entre sí, a través de una interfaz gráfica de usuario.

Se define IDE como el conjunto de herramientas utilizadas por los programadores que incluye, por lo general, un buen editor de código, administrador de proyectos y archivos, enlace a compiladores e integración con sistemas controladores de versiones o repositorios, además de brindar facilidades para la construcción de interfaces gráficas de usuario. (14)

NetBeans 7.2.1:

NetBeans 7.2.1 es un entorno de desarrollo integrado distribuido por SUN Microsystems bajo licencia dual: la Licencia Pública General (GPL) y la Licencia Común de Desarrollo y Distribución (CDDL). Esto significa que es gratuito y de código abierto para desarrolladores de software. Permite a los programadores escribir, compilar, depurar y ejecutar programas. Es apoyado por una comunidad de desarrolladores con más de 100 desarrolladores y ofrece una amplia documentación y recursos de capacitación. (15)
Además, introduce soporte de idiomas para el desarrollo de la especificación Java Standard Edition (Java SE 7) con las características del lenguaje Java Development Kit (JDK 7). Incluye un diseñador de GridBagLayout 4 para mejorar el desarrollo de interfaz gráfica de usuario.

Lenguajes de programación: es un idioma artificial diseñado para expresar procesos que pueden ser realizados por las computadoras. Describe un conjunto de acciones consecutivas que deben ser ejecutadas y que permiten crear programas a través de operadores y reglas de sintaxis. Es un modo práctico para dar instrucciones a un equipo de cómputo, que permite al desarrollador comunicarse con los dispositivos de hardware y software existentes.

Java: es un lenguaje de programación orientado a objetos creado por *Sun Microsystems, Inc*. que posee una curva de aprendizaje muy rápida. Fue diseñado para crear software altamente fiable, para ello proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Sus características de memoria liberan a los programadores de una familia entera de errores pues se ha prescindido por completo de los punteros, y la recolección de basura elimina la necesidad de liberación explícita de memoria. Java está diseñado para soportar aplicaciones que serán ejecutadas en los más variados entornos de red, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. (16)

Java Platform, Standard Edition o Java SE: es una colección de Interfaces de Programación de Aplicaciones (API), útiles para muchos programas de la Plataforma de lenguaje Java. Conocido hasta la versión 5.0 como *Java Platform* 2, *Standard Edition* o J2SE, incluye todas las clases en el Java SE. Java SE 7 es la versión actual y principal de la plataforma. Presenta aumentos en el rendimiento, la estabilidad, seguridad y mejoras en el lenguaje de programación Java que facilitan a los desarrolladores la escritura y optimización del código Java. (17) Java SE es la plataforma de desarrollo Java que provee el paquete básico de utilidades necesarias para el desarrollo de aplicaciones en este lenguaje.

Java Runtime Environment o JRE: es un conjunto de utilidades que permite la ejecución de programas realizados en Java sobre todas las plataformas soportadas. El entorno en tiempo de ejecución de Java está conformado por una Máquina Virtual de Java o JVM, un conjunto de bibliotecas Java y otros componentes necesarios para que una aplicación escrita en lenguaje Java pueda ser ejecutada. El JRE actúa como un *"intermediario"* entre el sistema operativo y Java. La JVM es el programa que ejecuta el código Java previamente compilado (*bytecode*) mientras que las librerías de clases estándar son las que implementan el API de Java.

Un usuario sólo necesita el JRE para ejecutar las aplicaciones desarrolladas en lenguaje Java, mientras que para desarrollar nuevas aplicaciones en dicho lenguaje es necesario un entorno de desarrollo, denominado JDK, además del JRE (requerimiento mínimo imprescindible) que incluye un compilador para Java. (17)

La aplicación *FCM – Decision*, está desarrollada en lenguaje Java, por esta razón y por la capacidad que presenta Java de ejecutarse en cualquier máquina y sobre cualquier sistema operativo o arquitectura,

manteniendo las facilidades básicas del lenguaje, se escogió el Netbeans 7.2.1 como herramienta de desarrollo.

Conclusiones del capítulo:

Como resultado del estudio realizado en este capítulo, se concluye que los sistemas existentes para el diseño automático de Mapas Cognitivos Difusos, no cumplen con los requisitos necesarios para solucionar el problema planteado. Quedaron definidas las bases arquitectónicas para el desarrollo del módulo. Se seleccionó Java como lenguaje de programación, *Netbeans 7.2.1* como entorno de desarrollo, *Visual Paradigm* 6.4 como herramienta de modelado y *OPEN UP* como metodología para guiar el proceso de desarrollo de software. Para modelar y visualizar los resultados se eligió el lenguaje de modelado UML.

Definir los elementos anteriores permitió identificar la necesidad del desarrollo del módulo diseño automático de MCD para la herramienta de modelado y análisis *FCM – Decision* que permita mayor aplicabilidad y fiabilidad a través de la correcta visualización del modelo causal.

CAPÍTULO 2: ANÁLISIS Y DISEÑO DEL SISTEMA

En el presente capítulo se identifican y describen los requisitos a implementar en el módulo durante el proceso de desarrollo del software. Se modela la estructura y el comportamiento del sistema para facilitar su comprensión por parte de clientes y desarrolladores. Partiendo de la identificación y definición de los requisitos funcionales, se identificaron los actores, casos de uso y las relaciones existentes entre estos, proporcionando como resultado la realización del diagrama de casos de uso del sistema.

Para guiar la implementación del módulo se utilizan los diagramas de clases del diseño, y a través de estos se muestra la estructura estática del sistema. Para modelar los aspectos dinámicos se realizaron los diagramas de secuencia donde se representan las clases y sus relaciones, incluyendo los mensajes que se pueden enviar entre ellas. Se describen los estilos arquitectónicos y patrones de diseño seleccionados, describiendo cómo se adaptan y se representan en el módulo. Se especifica la estructura física de la solución que se propone mediante el diagrama de despliegue.

2.1- Modelo del Dominio

El modelo de dominio constituye la representación estática del entorno real de los objetos de un proyecto. En él se realiza el modelado de los aspectos más importantes del entorno en el que se desarrolla la aplicación así como los elementos de la misma. Colabora a una mejor comprensión, por partes de los usuarios, de las definiciones asociadas al software y con las que este trabaja. Su objetivo es centrarse en una parte del negocio relacionándola con el ámbito del proyecto. A continuación se presenta el Modelo de Dominio (ver figura 3) del módulo diseño automático de Mapas Cognitivos Difusos y se describen sus clases.

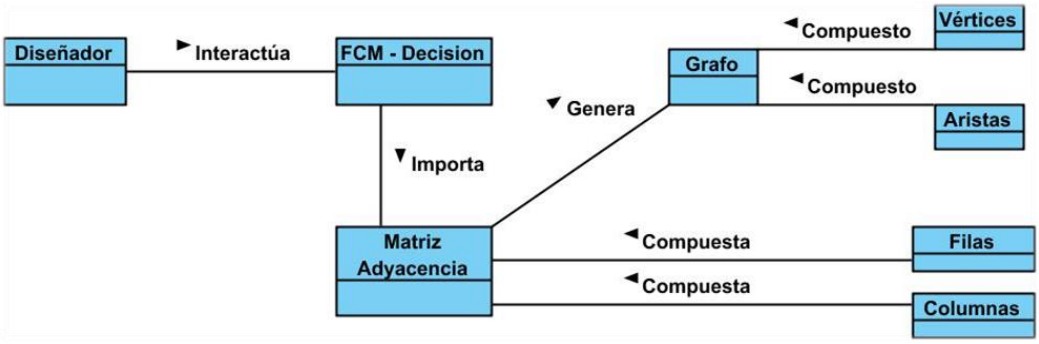


Figura 3: Modelo de Dominio

En el modelo de dominio anterior el concepto Diseñador representa al usuario, el cual interactúa con la aplicación *FCM – Decision* para importar y modificar una Matriz de Adyacencia a través de la cual se genera el grafo correspondiente. La Matriz de Adyacencia está compuesta por Filas y Columnas y el grafo por Vértices y Aristas.

Descripción de los conceptos del dominio:

Diseñador: interactúa con la aplicación *FCM – Decision* para importar y modificar una Matriz de Adyacencia a través de la cual se genera el grafo correspondiente.

Matriz Adyacencia: concepto contenedor de información del negocio compuesto por Filas y Columnas.

Filas y Columnas: componen una Matriz de Adyacencia y representan los nodos del grafo. Cada casilla de la matriz contiene valores numéricos que están en el intervalo [-1,1], existiendo un camino de un nodo o fila con una columna en caso de que estos valores sean distintos de cero.

Grafo: concepto contenedor de información del negocio compuesto por Vértices y Aristas, es la forma de representar los Mapas Cognitivos Difusos.

Vértices: es el punto donde se juntan las aristas. Además son conceptos causales y pueden modelar eventos, acciones, valores, metas o procesos.

Aristas: son líneas o segmentos que unen dos vértices e indican relación causal.

2.2- Especificación de requisitos

Durante la captura de requisitos se describen las condiciones o capacidades que debe cumplir el sistema y las propiedades o cualidades que debe presentar el mismo para darle solución al problema planteado. (18)

2.2.1- Requisitos funcionales

Los requisitos funcionales definen el comportamiento interno de un software, son condiciones que el sistema debe cumplir. Estos describen las funcionalidades que deben satisfacerse para cumplir con las especificaciones de software. (18) El módulo diseño automático de Mapas Cognitivos Difusos debe cumplir con los siguientes requisitos funcionales:

RF1: Importar Matriz de Adyacencia.

Descripción: el sistema permite importar una matriz de adyacencia. Para esto el usuario selecciona la opción importar matriz en la barra de menú o en el ícono correspondiente en la barra de herramientas.

Luego escoge el fichero donde se encuentra la misma y presiona sobre el botón abrir para completar la acción.

Entrada: fichero con extensión: txt. El mismo debe poseer un arreglo bidimensional de tamaño “n”, donde “n” es la máxima cantidad de nodos del grafo. Cada casilla de la matriz se carga con valores numéricos distintos de cero, cuando existe una arista entre la columna y el nodo o la fila y cero en caso de no existir relación. Las columnas de la matriz están separadas por un espacio simple (barra espaciadora) y las filas por un salto de línea (enter).

Salida: se muestra la Matriz de adyacencia compuesta por valores numéricos, exactamente igual a como está en el fichero importado.

RF2: Visualizar Matriz de Adyacencia.
Descripción: el sistema muestra la matriz de adyacencia importada en una pestaña.
Entrada: matriz de adyacencia compuesta por valores numéricos.
Salida: visualización de la Matriz de adyacencia compuesta por valores numéricos.

RF3: Modificar Matriz Adyacencia:
Descripción: el sistema modifica la matriz de adyacencia importada. Para esto el usuario introduce el valor a cambiar en la celda que seleccione, cambiándose el mismo al presionar la tecla enter o haciendo clic fuera de la celda modificada.
Entrada: matriz de adyacencia compuesta por valores numéricos.
Salida: matriz de adyacencia modificada.

RF4: Dibujar Mapa Cognitivo Difuso.
Descripción: el sistema a partir de la matriz de adyacencia genera el trazado del Mapa Cognitivo Difuso correspondiente y lo muestra en una pestaña.
Entrada: matriz de adyacencia importada por el usuario.
Salida: el Mapa Cognitivo Difuso compuesto por las relaciones entre los vértices y aristas.

2.2.2- Requisitos no funcionales

Los requisitos no funcionales constituyen propiedades o cualidades que el producto debe tener. A continuación se muestran los requisitos del módulo: (18)

Usabilidad:

- El sistema agrupa botones por grupos funcionales.
- El sistema presenta una ayuda para orientar a los usuarios en el uso de la interfaz.
- El sistema utiliza el idioma castellano para los mensajes de excepción y los textos en la interfaces.

Software:

- La estación de trabajo donde se ejecuta la aplicación puede tener cualquier sistema operativo pero debe estar instalada la máquina virtual de Java.

Restricciones de Diseño e Implementación:

- El sistema emplea como lenguaje de programación Java y la herramienta de desarrollo será el Netbeans 7.2.1.

Hardware:

- La computadora donde se ejecuta la aplicación debe cumplir con los siguientes requisitos: Procesador Intel Pentium 3, o AMD similar o superior, 256 MB RAM o superior y 20 GB de espacio libre en disco duro.

Fiabilidad:

- El sistema debe poseer la capacidad para capturar excepciones en casos de errores. Estos pueden ocurrir en la entrada de datos por parte del usuario cuando modifica una matriz de adyacencia, o errores que presente la aplicación cuando se manda a dibujar un mapa cognitivo difuso.

2.3- Diagrama de caso de usos del sistema

El diagrama de casos de uso del sistema es la representación del comportamiento del software desde el punto de vista del usuario. Permite visualizar las funcionalidades en casos de uso, el entorno del sistema en actores, y las relaciones entre ambos. Aunque la parte más visible de dicho modelo son los diagramas de casos de uso, suele ir acompañado de una especificación textual de cada uno de los casos de uso. A continuación se muestra el actor del sistema y se conciben, a través de la agrupación de los requisitos funcionales anteriormente descritos, los casos de uso del sistema.

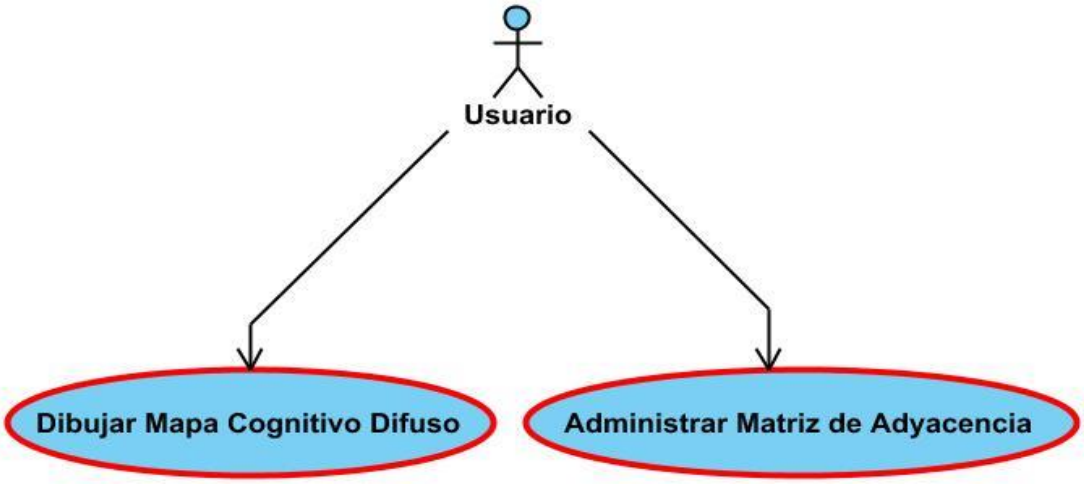


Figura 4: Diagrama de casos de uso del sistema

En el diagrama de casos de uso del sistema de la figura 4, el actor Usuario es el encargado de iniciar los casos de uso: Administrar Matriz de Adyacencia y Dibujar Mapa Cognitivo Difuso. De esta forma, se agruparon los cuatro requisitos funcionales en dos casos de uso, ambos de alta prioridad para la arquitectura del módulo.

Descripción del actor del sistema:

Actor	Descripción
Usuario	Actor del sistema que interactúa con la aplicación. Responsable de realizar acciones tales como: seleccionar la matriz de adyacencia a importar y modificarla.

Descripción del caso de uso: Dibujar Mapa Cognitivo Difuso:

Caso de Uso:	**Dibujar Mapa Cognitivo Difuso**
Actor:	**Usuario**
Resumen:	El caso de uso inicia cuando el actor “Usuario” selecciona la opción “Dibujar MCD” en el menú Opciones o en la barra de herramientas. Luego se muestra una ventana donde el usuario especifica los parámetros por los

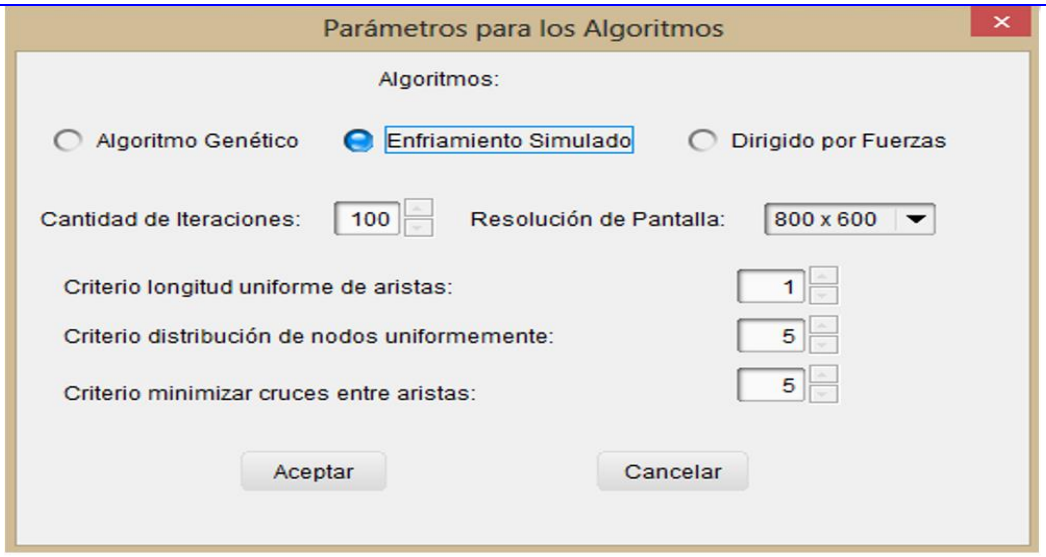
	cuales se regirá el trazado. Cuando el usuario selecciona el botón aceptar se muestra el trazado del MCD correspondiente a la matriz de adyacencia importada con anterioridad.
Precondiciones	El sistema debe encontrarse ejecutándose correctamente y tener una matriz de adyacencia importada.
Referencias	RF1, RF2
Prioridad	Crítico

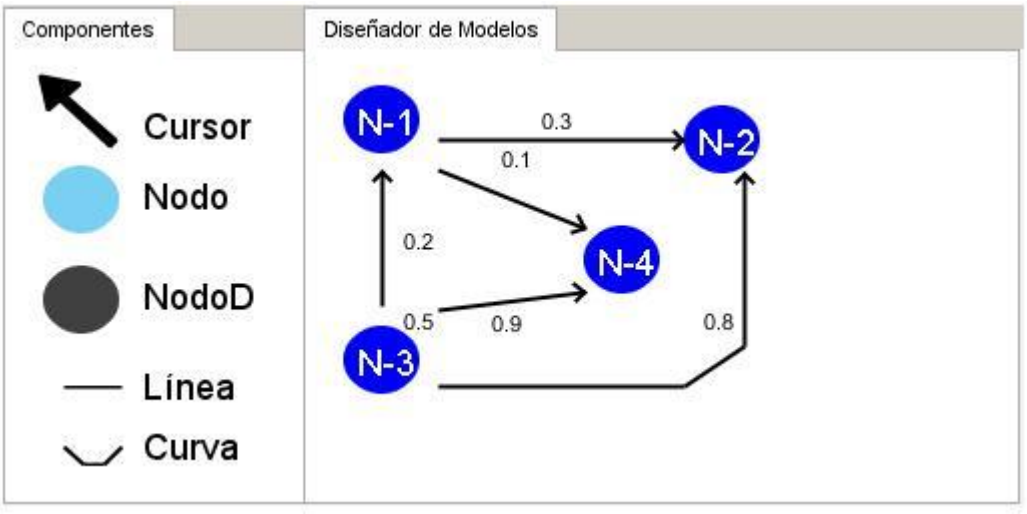
Flujo Normal de Eventos

Sección "Dibujar Mapa Cognitivo Difuso"

Acción del Actor	**Respuesta del Sistema**
1. El caso de uso inicia con la selección por parte del "Usuario", de la opción "Dibujar MCD" en el menú Opciones o directamente en el ícono correspondiente en la barra de herramientas.	2. Muestra una ventana para que el usuario entre los parámetros por los cuales se regirá el trazado.
3. Introduce los parámetros por los cuales se regirá el trazado del MCD y presiona el botón Aceptar.	4. Muestra el MCD en una pestaña, terminando así el caso de uso.

Prototipo de Interfaz





lujos Alternos	
Acción del Actor	**Respuesta del Sistema**
2.1. Selecciona el botón Cancelar para cerrar la ventana finalizando así el caso de uso.	

	3.1. Muestra un mensaje indicando el error que existe en los datos entrados.

Prototipo de Interfaz

Poscondiciones	El sistema muestra en una pestaña un MCD.

2.4- Diagrama de clases del diseño

Los diagramas de clases son diagramas estáticos que describen la estructura de un sistema. Muestran las diferentes clases que componen al mismo y cómo se relacionan unas con otras.

En el diseño los diagramas de clases son la manera de describir gráficamente clases, interfaces y colaboraciones, así como sus relaciones. Son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa. El siguiente diagrama de clases del diseño (ver figura 5) contiene las principales clases con sus respectivos métodos y atributos para el caso de uso Dibujar Mapa Cognitivo Difuso. El diagrama correspondiente al caso de uso Administrar Matriz de Adyacencia se muestra en el Anexo 1.

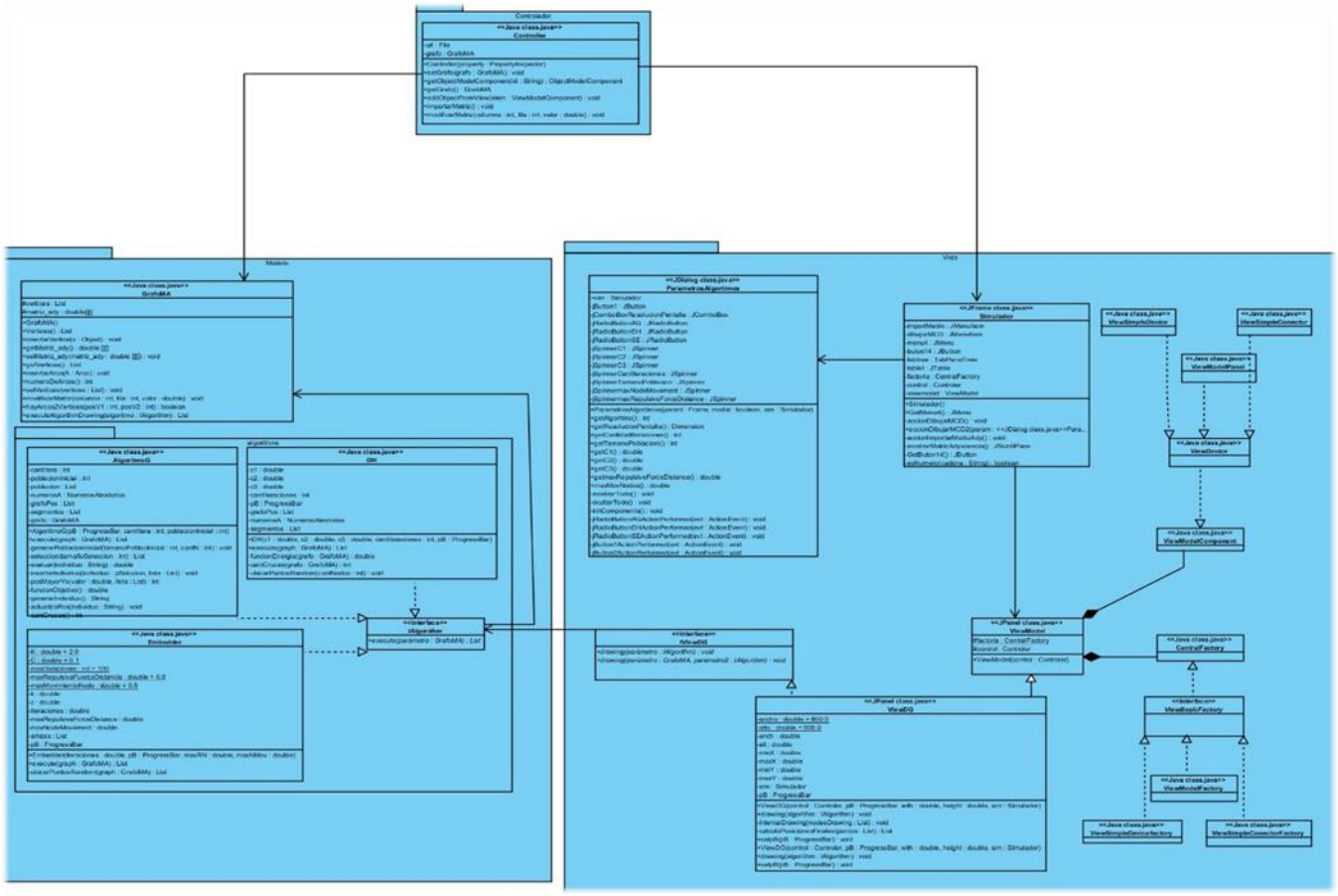


Figura 5: Diagrama de clases del diseño para el caso de uso Dibujar Mapa Cognitivo Difuso

Descripción de las clases:

A continuación se describen las principales clases del módulo.

Nombre de la clase: Simulador
Descripción: Interfaz principal de la aplicación.
Propósito: Crear los componentes de la interfaz principal de la aplicación. Permite al usuario interactuar con la aplicación a través de botones específicos para cada funcionalidad.

Nombre de la clase: ViewModel
Descripción: Panel que maneja los eventos del teclado y el mouse.
Propósito: Pintar y realizar las operaciones de copia y pega de los nodos del grafo sin utilización de matriz de adyacencia.

Nombre de la clase: ViewDG
Descripción: Clase que se utiliza para dibujar Vértices y Aristas.
Propósito: Pintar los vértices y aristas, a partir de la matriz de adyacencia importada por el usuario. Hereda de la clase ViewModel por lo que maneja también los eventos del mouse y el teclado y realiza las operaciones de copia y pega de los nodos del grafo.

Nombre de la clase: ParámetrosAlgoritmos
Descripción: Contiene la ventana donde se modifican los parámetros por los cuales se regirá el trazado según la convención escogida.
Propósito: Brindar al usuario la posibilidad de modificar los parámetros por los cuales se regirá el trazado según la convención escogida.

Nombre de la clase: GrafoMA
Descripción: Contiene toda la información sobre el grafo.
Propósito: Realizar todas las operaciones sobre el grafo (Modificar matriz de adyacencia, Insertar Vértice, Insertar Arista).

Nombre de la clase: *Controller*
Descripción: Controladora de todas las funcionalidades que se realizan en la aplicación.
Propósito: Controlar, recibiendo y enviando peticiones a las demás clases, de todas las funcionalidades de la aplicación, entre estas funcionales se encuentran: Importar Matriz de Adyacencia, Añadir un componente a la vista, entre otras.

Nombre de la clase: *IAlgorithm*
Descripción: Interfaz que es ejecutada por las distintas convenciones que existen para el trazado del grafo.
Propósito: Ejecutar la convención para el trazado del grafo selecciona por el usuario.

2.5- Diagrama de secuencia

La creación de los diagramas de secuencia de un sistema es una de las actividades más importantes en el desarrollo del mismo. Al construirlos se toman decisiones claves acerca de su funcionamiento futuro y permiten representar su comportamiento dinámico. Este comportamiento se muestra a continuación para el escenario Dibujar Mapa Cognitivo Difuso, los escenarios correspondientes al caso de uso Administrar Matriz de Adyacencia se pueden observar en el Anexo 2.

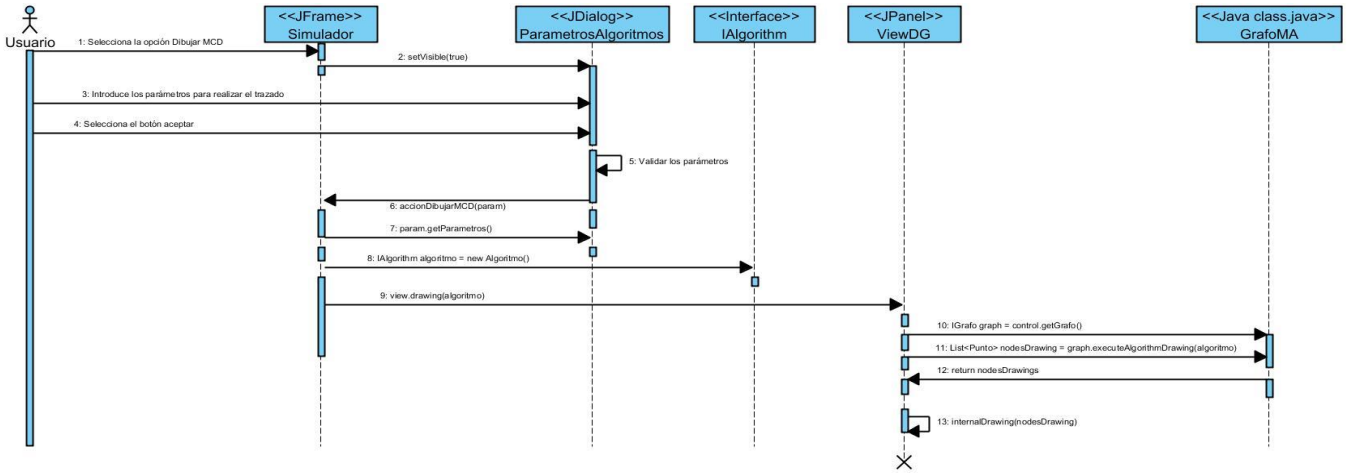


Figura 6: Diagrama de secuencia del escenario Dibujar Mapa Cognitivo Difuso

En el diagrama de secuencia el actor selecciona la opción Dibujar Mapa Cognitivo Difuso, mostrándose una ventana en la que introduce los parámetros por los cuales se regirá el trazado, escogerá la convención para realizar el mismo y presionará el botón Aceptar. El sistema validará los parámetros entrados y se ejecutará el método "accionDibujarMCD" el cual obtiene los parámetros entrados y a partir de estos se creará un algoritmo a través de la interfaz "*IAlgorithm"*. A continuación se llama al método "*drawing"* de la clase "*ViewDG"* el cual obtiene el grafo a dibujar y ejecuta el método "*executeAlgorithmDrawing"* el cual retorna la ubicación final donde se pintarán los vértices y las aristas para formar el grafo, cuando se ejecute el método "*internalDrawing*".

2.6- Patrones utilizados en la solución

Un patrón es un modelo o guía que captura la experiencia existente y probada para promover buenas prácticas. Para el desarrollo de un software existen varios tipos de patrones siendo los de arquitectura los que están encaminados a describir la estructura u organización del software.

Patrones Arquitectónicos:

Modelo Vista Controlador (MVC): es un patrón de arquitectura de software, que permite separar los datos de una aplicación, la interfaz de usuario y la lógica de negocio en tres componentes distintos. Esto proporciona múltiples vistas sobre un mismo modelo de datos. Cada vista tiene asociado un componente de controlador o puede existir un controlador para todas las vistas. Los controladores reciben entradas, casi siempre en forma de eventos que codifican los movimientos del ratón o la activación de botones. Los eventos se transforman en peticiones de servicios del modelo o de la vista. (19) El MVC es la razón de ser de las aplicaciones interactivas, las cuales necesitan de interfaces de usuarios muy flexibles. Los tres elementos esenciales de este patrón son los siguientes:

- **Modelo:** es el núcleo de la aplicación. Administra el comportamiento y los datos del dominio de la aplicación, responde a requisitos de información sobre su estado, usualmente formulados desde la vista, respondiendo a instrucciones de cambio para modificar el estado de estos datos, desde el controlador.
- **Vista:** es la encargada de mostrar al usuario el progreso del funcionamiento del modelo. Cuando el modelo cambia, la vista solicita el nuevo estado y se actualiza.

- **Controlador:** es una especie de capa intermediaria que conecta la vista con el modelo. De acuerdo con los mensajes recibidos de la vista (la acción del usuario), modifica a ésta, y se pone en contacto con el modelo como lo muestra la figura 7.

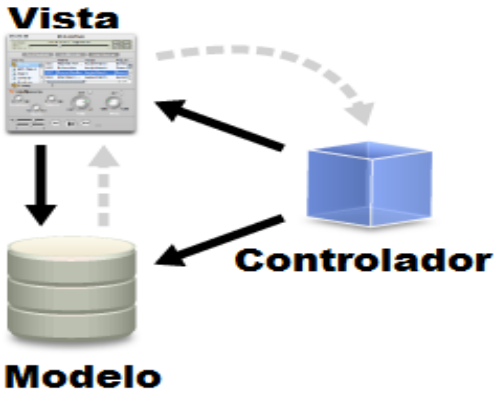


Figura 7: Representación del patrón Modelo Vista Controlador

El uso de este patrón proporciona las siguientes ventajas:

- Soporta múltiples vistas: Como la vista se encuentra separada del modelo y no existe dependencia directa del modelo con respecto a la vista, la interfaz de usuario puede mostrar múltiples vistas de los mismos datos simultáneamente.
- Adaptación al cambio: Los requisitos de interfaz de usuario tienden a cambiar con mayor rapidez que las reglas de negocios. Los usuarios pueden preferir distintas opciones de representación y como el modelo no depende de las vistas, agregar nuevas opciones de presentación, generalmente no afecta al modelo.

La experiencia en el desarrollo de aplicaciones complejas ha demostrado que la lógica de una interfaz de usuario cambia con más frecuencia que los almacenes de datos y la lógica del negocio. Si se realizara un diseño donde se mezclen los componentes de la interfaz y del negocio, puede traer como consecuencia que al modificar la interfaz, también se deben modificar los componentes del negocio, haciendo el trabajo más complejo y con un mayor riesgo de que ocurran errores. Esta es otra de las razones por la cual se escogió este patrón, pues su uso posibilita realizar el diseño de una aplicación donde se desacople la vista del modelo, con la finalidad de mejorar la reusabilidad. De esta forma las modificaciones en las vistas impactan en menor medida en la lógica de negocio o de datos.

Patrones de Diseño:

Los patrones de diseño proporcionan un esquema para refinar los subsistemas o componentes de un software y las relaciones entre ellos, por lo que están más próximos a lo que sería el código fuente final.

***Abstract Factory* (Factoría Abstracta):**

Proporciona una interfaz para crear familias de objetos relacionadas o dependientes, sin especificar su clase concreta. El uso de este patrón consiste en declarar una interfaz de operaciones *AbstractFactory* que crea una familia de productos que también son definidos en clases abstractas, toda la interacción del cliente será mediante las interfaces declaradas. La creación de nuevos productos se realiza a través de la interfaz sin necesidad de saber que *ConcreteFactory* está siendo utilizada ni cual *Product* se recibe.

El patrón *Abstract Factory*, se aplica en el módulo para la creación del modelo, los nodos y las aristas. En su estructura (ver figura 8), se representa la clase *ViewModel* como el modelo gráfico del Diseño de Mapas Cognitivos Difusos, el mismo contiene una instancia de la clase *CentralFactory*, que funciona como una fábrica central que contiene varias fábricas concretas, estas últimas se implementan en la interfaz *ViewBasicFactory*. En esta interfaz se crea el modelo y sus componentes a través de las clases *ViewModelFactory* y V*iewSimpleDevicefactory* respectivamente.

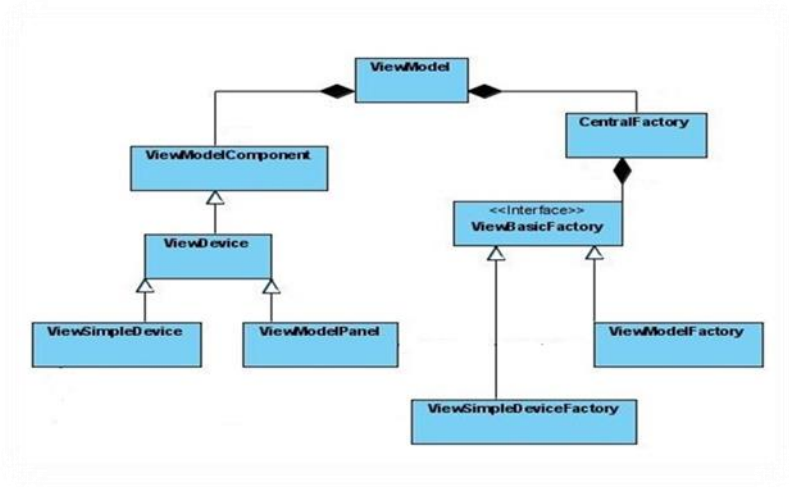


Figura 8: Representación del patrón de diseño *Abstract Factory*

El uso de este patrón posibilita crear el modelo y sus componentes sin especificar cuál de ellos se creará. Para crear uno de estos se obtiene un componente partiendo de varios datos, según los datos especificados será el componente que se creará. Por ejemplo cuando se crea un nodo, se especifican su

coordenada (x, y) que representa la posición en el plano que ocupará el mismo, y el modelo donde se insertará el mismo. El modelo también se especifica en el caso de una arista además de las coordenadas de origen y destino.

Experto:

En la aplicación, con la utilización de este patrón se conserva el encapsulamiento pues los objetos se valen de su propia información para hacer lo que se les pide. Esto soporta un bajo acoplamiento, lo que favorece al hecho de tener sistemas más robustos. El comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y más cohesivas que son más fáciles de comprender y de mantener.

El patrón se aplica cuando se desea modificar una matriz de adyacencia desde la clase Simulador. En este caso toda la responsabilidad para la implementación del método recae sobre la clase GrafoMA, esta es la experta en la información necesaria para crear y modificar la matriz, de esta manera el método es más sencillo de realizar y comprender. Para implementarlo la clase *Controller*, obtiene los datos para la modificación de la matriz de la clase Simulador y se los envía a la clase GrafoMA, experta en la información que se va a modificar (ver figura 9).

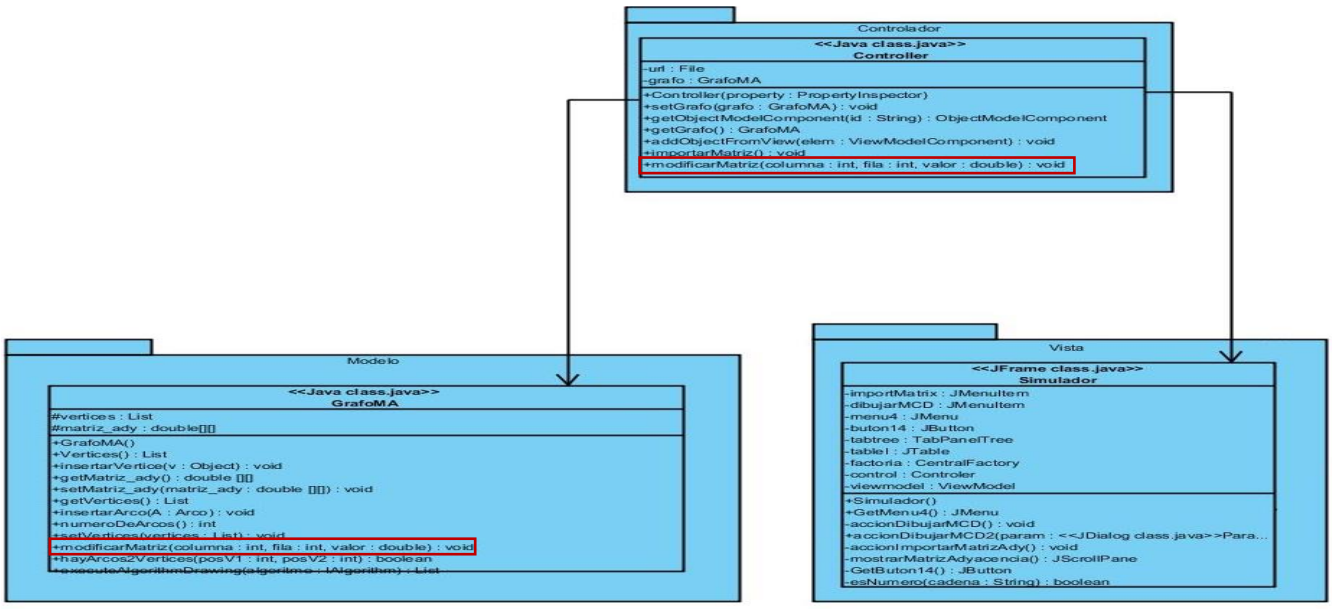


Figura 9: Representación del patrón Experto

Controlador:

El uso de este patrón aporta un mayor potencial de los componentes reutilizables**.** Garantiza que los procesos de dominio sean manejados por la capa de los objetos del dominio y no por la de la interfaz. El hecho de delegar a un controlador la responsabilidad de las operaciones de un sistema, soporta la reutilización de la lógica para manejar los procesos afines del negocio en aplicaciones futuras. Este patrón sugiere que la lógica de negocios debe estar separada de la capa de presentación, aumentando así la reutilización de código y a la vez tener un mayor control.

La responsabilidad para aplicar el patrón controlador en el módulo la tiene la clase *Controller,* esta funciona como intermediaria entre las interfaces y las demás clases con quien tiene relación. De esta forma es la que recibe los datos del usuario y los envía a las distintas clases según el método llamado.

2.7- Vista de Despliegue

El Modelo de Despliegue es un modelo de objetos que describe la distribución física del sistema en términos de cómo se distribuye la funcionalidad entre los nodos de cómputo. Un nodo es un elemento físico que existe en tiempo de ejecución y representa un recurso computacional, que generalmente tiene memoria y a menudo, capacidad de procesamiento. Los nodos se utilizan para modelar la topología del hardware sobre el que se ejecuta el sistema. Representan un procesador o un dispositivo sobre el que se pueden desplegar los componentes. La relación entre un nodo y el componente que despliega puede mostrarse con una relación de dependencia. La siguiente figura representa la vista de despliegue de la aplicación *FCM – Decision.*

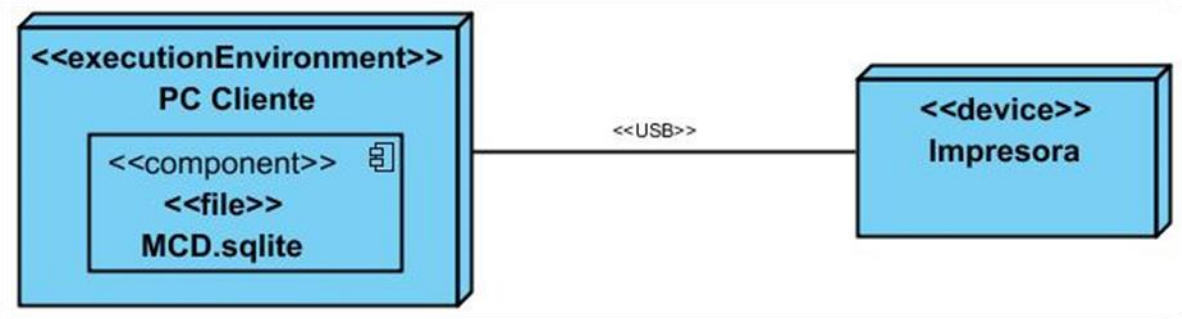


Figura 10: Vista de Despliegue

Para desplegar la aplicación *FCM – Decision* se necesita una estación de trabajo, la misma debe contener el archivo MCD.sqlite el cual contiene los modelos generados por la aplicación. Para imprimir dichos modelos se requiere de una impresora conectada a la estación de trabajo, de esta forma se tendrán en formato duro los modelos impresos para su posterior análisis.

Conclusiones del capítulo

En el presente capítulo se representaron, a través del modelo del dominio, las relaciones entre los principales conceptos del negocio. Se seleccionaron y describieron los requisitos funcionales y no funcionales, agrupando los funcionales en dos casos de uso críticos para los cuales se realizó el diagrama de casos de uso del sistema. Se realizaron los diagramas de clases del diseño, identificando y describiendo las clases necesarias para el correcto funcionamiento del sistema. Para mostrar al desarrollador la estructura dinámica del sistema se realizaron los diagramas de secuencia. Finalmente se identificaron y utilizaron patrones de arquitectura y diseño con el objetivo de proveer a los desarrolladores un catálogo de referencia ante problemas en la construcción del módulo.

CAPÍTULO 3: IMPLEMENTACIÓN Y PRUEBA

La implementación comienza a partir de los resultados obtenidos en el diseño implementando el sistema en términos de componentes, es decir, ficheros de código fuente, ficheros de código binario y ejecutables. En el desarrollo de este capítulo se describe la implementación del módulo Diseño Automático de Mapas Cognitivos Difusos, mostrando fragmentos de código de las principales clases. De esta forma se generan los artefactos pertenecientes a las fases de implementación y prueba de un software. De esta última fase se especifican los casos de pruebas realizados al módulo para validar su correcto funcionamiento.

3.1- Diagrama de componentes

Los diagramas de componentes modelan la vista estática de un sistema, mostrando la organización y las dependencias entre un conjunto de componentes. Para la creación de estos diagramas no siempre es necesario incluir todos los componentes del sistema, normalmente estos se realizan por partes donde cada una describe un apartado del sistema. En él se representan las librerías, archivos, ejecutables y documentos que formen parte del mismo.

Al iniciar la implementación de la herramienta, todos los diagramas generados en fases anteriores se materializan en un sistema integrando las partes necesarias para la obtención de un producto final. Estas relaciones de dependencia se modelan a través del diagrama de componentes, como se muestra en la figura 11.

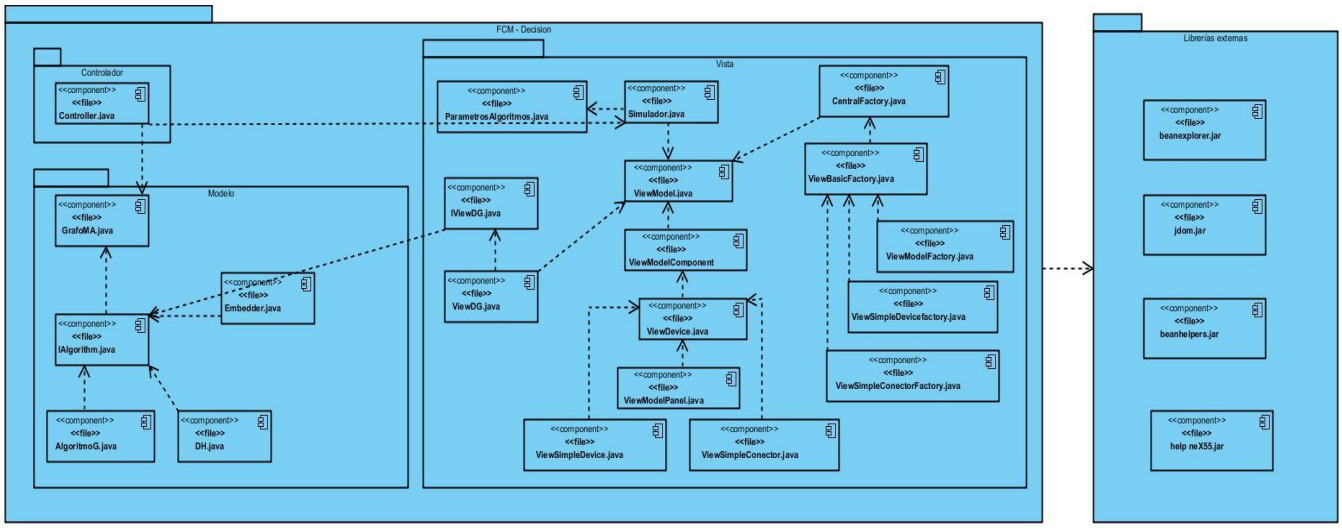


Figura 11: Diagrama de Componentes

En el diagrama anterior se muestran los componentes divididos en paquetes utilizando el patrón Modelo Vista Controlador. Solo se representaron las librerías utilizadas y los principales componentes del módulo con extensión ".java" pues si se representaran todos, elevarían el nivel de complejidad del diagrama, provocando una menor comprensión del mismo.

En el paquete Vista se localizan los componentes que pertenecen a las interfaces del módulo, mientras que el componente intermediario entre la vista y el modelo y encargado del control de las funcionalidades, se encuentra en el paquete Controlador. "GrafoMA.java", componente que representa el núcleo del módulo, así como los que permiten realizarle cambios a este, se exhiben en el paquete Modelo. Los componentes que se muestran con extensión ".jar" representan las librerías utilizadas, las cuales están contenidas en el paquete Librerías Externas.

3.2- Estándar de codificación

Para garantizar la uniformidad del código en el desarrollo del módulo, se definió un estándar de codificación, lo cual posibilita cometer menos errores durante la implementación y que la lectura y comprensión del mismo sea mucho más fácil para otros desarrolladores. A continuación se muestra el estándar utilizado:

Indentación:

Longitud de la línea:

- Las líneas tienen siempre no más de 80 caracteres.

Rompiendo líneas:

Cuando una expresión no entra en una línea, se rompe de acuerdo con estos principios:

- Romper después de una coma.
- Romper antes de un operador.

Ejemplo:

```
double peso = Double.parseDouble(JOptionPane.showInputDialog(null, "Peso del arco",
                                        element.getType(), JOptionPane.PLAIN_MESSAGE));
```

Declaraciones:

- Las variables y los métodos se declararon comenzando con minúscula, y la primera letra de las siguientes palabras en mayúscula.

- Para declarar una clase se comienza con letra mayúscula. Deben ser nombres cortos pero con significado.

Miscelánea: Se utilizó el estándar que brinda el Netbeans. Ejemplo:
if (condición) {
…….
} else {
…..
}

3.3- Código fuente de las principales clases del módulo desarrollado

El siguiente fragmento de código pertenece al método que permite al usuario importar una matriz de adyacencia.

```
public double[][] importarMatriz() throws Exception {
    double[][] matrizAdy = new double[0][0];
    grafo.setMatriz_ady(matrizAdy);
    grafo.setVertices(new ListaSE<ObjectModelComponent>());
    File direccion = null;
    JFileChooser nuevo = new JFileChooser();
    if (nuevo.showOpenDialog(nuevo) == 0) {
        direccion = nuevo.getSelectedFile();
    }
    if (direccion != null) {
        BufferedReader bu = new BufferedReader(new FileReader(direccion));
        ListaSE<String[]> lista = new ListaSE<String[]>();
        String a = bu.readLine();
        while (a != null) {
            String[] arreglo = a.split(" ");
            lista.Adicionar(arreglo);
            a = bu.readLine();
        }
        matrizAdy = new double[lista.Longitud()][lista.Obtener(0).length];
        int cantRFilas = lista.Longitud();
        int cantRColums = lista.Obtener(0).length;
        grafo.setCantRealFilas(cantRFilas);
        grafo.setCantRealColumnas(cantRColums);
        for (int i = 0; i < cantRFilas; i++) {
            for (int j = 0; j < cantRColums; j++) {
                String cadena = lista.Obtener(i)[j];
                for (int k = 0; k < cadena.length(); k++) {
                    if (Character.isLetter(cadena.charAt(k))) {
                        throw new soloNumerosException();
                    } else {
                        matrizAdy[i][j] = Double.parseDouble(cadena);
                    } } }  } }
grafo.setMatriz_ady(matrizAdy);
return matrizAdy;
}
```

Figura 12: Procedimiento para Importar Matriz de Adyacencia

Para realizar el trazado del Mapa Cognitivo Difuso utilizando la convención “Enfriamiento Simulado” se siguió el procedimiento que se muestra a continuación en pseudocódigo.

EnfriamientoSimulado (Grafo G)

1. Asignar temperatura inicial $t=t_0$.
2. Asignar una posición al azar a cada vértice de G
3. Repetir M veces:
   - 3.1. Repetir T veces:
     - 3.1.1. Para cada $v \in V$
       - 3.1.1.1. $p_{ant} = p_v$
       - 3.1.1.2. $p_v = p_v + \Delta_{rand}$
       - 3.1.1.3. Si $E(p_{ant}) < E(p_v)$
         - 3.1.1.3.1 Con probabilidad $1\text{-}e^{\frac{E(p_{ant})-E(p_v)}{t}}$, volver a $p_v = p_{ant}$
   - 3.2 Reducir temperatura t

Figura 13: Pseudocódigo del Procedimiento realizar el trazado del Mapa Cognitivo Difuso utilizando la convención “Enfriamiento Simulado”

A continuación se muestra, en dos columnas, la implementación del procedimiento para realizar el trazado del Mapa Cognitivo Difuso, utilizando la convención “Enfriamiento Simulado” a partir del pseudocódigo anterior.

```
public List<Punto> execute(GrafoMA graph) {
    int cantNodos = graph.getMatrizD().getCantVertices();
    ubicarPuntosRandom(cantNodos);
    double temperatura = Double.MAX_VALUE;
    double r = 10;
    for (int i = 0; i < cantIteraciones; i++) {
        int cantit = 30 * cantNodos;
        int porciento = (i * 100) / (cantIteraciones);
        pB.getjProgressBar1().setValue(porciento);
        pB.getjProgressBar1().setStringPainted(true);
        for (int k = 0; k < cantit; k++) {
            for (int j = 0; j < cantNodos; j++) {
            Punto pAnterior = new Punto(grafoPos.get(j).getX(), grafoPos.get(j).getY());
            double ang = Math.toRadians((int) (Math.random() * 360));
            double dx = r * Math.cos(ang);
            double dy = r * Math.sin(ang);
            double x = grafoPos.get(j).getX();
            double y = grafoPos.get(j).getY();
            x += dx;
            y += dy;
            double eAnterior = funcionEnergia(graph);
            grafoPos.get(j).setX(x);
            grafoPos.get(j).setY(y);
            double eActual = funcionEnergia(graph);
            if (eActual > eAnterior) {
                if (numerosA.valorProbabilidad() > ((Math.exp((eAnterior - eActual) / temperatura)))) {
                    grafoPos.add(j, pAnterior);
                    grafoPos.remove(j + 1);
                }
            }
        }
    }
    temperatura *= 0.75;
    r *= 0.75;
}
for (int j = 0; j < cantNodos; j++) {
    grafoPos.get(j).setX(grafoPos.get(j).getX() + 225);
    grafoPos.get(j).setY(grafoPos.get(j).getY() + 225);
}
return grafoPos;
}
```

Figura 14: Implementación del procedimiento para realizar el trazado del Mapa Cognitivo Difuso utilizando la convención “Enfriamiento Simulado”

La implementación del trazado del Mapa Cognitivo Difuso utilizando la convención “Algoritmo Genético” estuvo guiada por el procedimiento que se muestra a continuación en pseudocódigo:

```
BEGIN /* Algoritmo Genético Simple */
    Generar una población inicial.
    Computar la función de evaluación de cada individuo.
    WHILE NOT Terminado DO
    BEGIN /* Producir nueva generación */
        FOR Tamaño población/2 DO
        BEGIN /*Ciclo Reproductivo */
            Seleccionar dos individuos de la anterior generación,
            para el cruce (probabilidad de selección proporcional
            a la función de evaluación del individuo).
            Cruzar con cierta probabilidad los dos
            individuos obteniendo dos descendientes.
            Mutar los dos descendientes con cierta probabilidad.
            Computar la función de evaluación de los dos
            descendientes mutados.
            Insertar los dos descendientes mutados en la nueva generación.
        END
        IF  la población ha convergido THEN
            Terminado := TRUE
    END
END
```

Figura 15: Pseudocódigo del Procedimiento realizar el trazado del Mapa Cognitivo Difuso utilizando la convención “Algoritmo Genético”

La siguiente figura muestra el código fuente del procedimiento implementado, a partir del pseudocódigo anterior para realizar el trazado del Mapa Cognitivo Difuso.

```
public List<Punto> execute(GrafoMA graph) {
    grafo = graph;
    int cantN = graph.getMatrizD().getCantVertices();
    if (poblacionInicial % 2 != 0) {
        poblacionInicial += 1;
    }
    pB.getjProgressBar1().setStringPainted(true);
    generarPoblacionInicial(poblacionInicial, cantN);
    int cont = 0;
    while (cantItera > cont) {
        int porciento = (cont * 100) / cantItera;
        pB.getjProgressBar1().setValue(porciento);
        List<pSolucion> seleccionados = seleccion(poblacionInicial);
        poblacion = new LinkedList<pSolucion>();
        CruzamientoyMutacion cruceyMutacion = new CruzamientoyMutacion();

        for (int i = 0; i < seleccionados.size(); i += 2) {
            List<pSolucion> evaluaciones = new LinkedList<pSolucion>();

            String[] hijos = cruceyMutacion.cruceUnPunto(seleccionados.get(i).getIndividuo(), seleccionados.get(i + 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijos[0]), hijos[0]), evaluaciones);
            insertarIndividuo(new pSolucion(evaluar(hijos[1]), hijos[1]), evaluaciones);

            hijos = cruceyMutacion.cruceUniforme(seleccionados.get(i).getIndividuo(), seleccionados.get(i + 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijos[0]), hijos[0]), evaluaciones);
            insertarIndividuo(new pSolucion(evaluar(hijos[1]), hijos[1]), evaluaciones);

            hijos = cruceyMutacion.cruceUniformeVariedad(seleccionados.get(i).getIndividuo(), seleccionados.get(i + 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijos[0]), hijos[0]), evaluaciones);
            insertarIndividuo(new pSolucion(evaluar(hijos[1]), hijos[1]), evaluaciones);
            // tres cruces juntos
            hijos = cruceyMutacion.cruzar(seleccionados.get(i).getIndividuo(), seleccionados.get(i + 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijos[0]), hijos[0]), evaluaciones);
            insertarIndividuo(new pSolucion(evaluar(hijos[1]), hijos[1]), evaluaciones);
            //Mutacion con el mejor
            String hijo = cruceyMutacion.mutacionSimple(evaluaciones.get(evaluaciones.size() - 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijo), hijo), evaluaciones);

            hijo = cruceyMutacion.mutacionSimpleValor(evaluaciones.get(evaluaciones.size() - 2).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijo), hijo), evaluaciones);
            //dos mutaciones juntas
            hijo = cruceyMutacion.mutar(evaluaciones.get(evaluaciones.size() - 1).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijo), hijo), evaluaciones);

            hijo = cruceyMutacion.mutar(evaluaciones.get(evaluaciones.size() - 2).getIndividuo());
            insertarIndividuo(new pSolucion(evaluar(hijo), hijo), evaluaciones);

            insertarIndividuo(evaluaciones.get(evaluaciones.size() - 1), poblacion);
            insertarIndividuo(evaluaciones.get(evaluaciones.size() - 2), poblacion);
        }
        cont++;
    }
    actualizaPos(poblacion.get(poblacion.size() - 1).getIndividuo());
    return grafoPos;
}
```

Figura 16: Implementación del procedimiento para realizar el trazado del Mapa Cognitivo Difuso utilizando la convención "Algoritmos Genéticos"

3.4- Pruebas de software

Las pruebas de software son un conjunto de herramientas, técnicas y métodos que evalúan la excelencia y el desempeño de un software, involucra las operaciones del sistema bajo condiciones controladas y evaluando los resultados. Las técnicas para encontrar problemas en un programa son extensamente variadas y van desde el uso del ingenio por parte del personal de prueba hasta herramientas automatizadas que ayudan a aliviar el peso y el costo de tiempo de esta actividad. (20)

Una vez completada la implementación es necesario probar el software con el objetivo de descubrir y corregir la mayor cantidad de errores posibles antes de entregárselo al cliente. Las pruebas aplicadas al módulo Diseño Automático de Mapas Cognitivos Difusos fueron:

Nivel de Prueba: Pruebas de Unidad y de Integración.

Luego de concluir totalmente la implementación del sistema se procede a la revisión de la misma. Para ello se proponen pruebas unitarias en pos de analizar la lógica interna de la aplicación. Estas pruebas constituyen la primera fase de las pruebas dinámicas y se realizan sobre cada módulo de la aplicación de manera independiente. Su objetivo es comprobar que cada módulo, entendido como una unidad funcional de un programa independiente, esté correctamente codificado. En estas pruebas cada módulo será probado por separado y lo hará, generalmente, la persona que lo creó.

Para la realización de las pruebas unitarias el desarrollador diseñó los Casos de Prueba basados en casos de usos y apoyándose en el Estándar de Codificación definido. El desarrollador fue el rol especificado para la ejecución de las pruebas. Para el desempeño de la misma verificó la correspondencia del código de la aplicación con las métricas y el estándar de codificación definido.

Las pruebas de integración se realizan luego de las pruebas unitarias y enmarcan su atención en la integración de los componentes de la aplicación. El objetivo de estas pruebas es verificar que el sistema funcione correctamente una vez integrado.

Técnicas de Prueba: Pruebas de Funcionalidad.

Para probar las funcionalidades del sistema se diseñaron y aplicaron Casos de Pruebas, además de comprobar que la aplicación se corresponda con lo definido para el desarrollo de la misma, entiéndase Requisitos Funcionales y Descripción de Casos de Uso.

Las pruebas de funcionalidad aseguran el trabajo apropiado de los requisitos funcionales, incluyendo la navegación, entrada de datos, procesamiento y obtención de resultados. Estas se centraron en verificar el procesamiento, recuperación e implementación adecuada de las reglas del negocio, verificando la apropiada aceptación de datos.

Las pruebas de funcionalidad se realizaron para los dos casos de uso del módulo, apoyadas en el diseño de los casos de prueba, con el objetivo de que el módulo y la aplicación, una vez integrados, quede sin no conformidades. Se determinó además que las pruebas a realizar se regirán por el método de caja negra.

Métodos de Prueba: Caja Negra.

Se utilizó como método de prueba, el de Caja Negra. El probador ejecutó todos los casos de uso introduciendo datos válidos e inválidos. El objetivo de la entrada de datos válidos e inválidos, es verificar que se aplicaron correctamente cada una de las reglas de negocio establecidas, que la respuesta del sistema esté de acuerdo con lo que se espera en cada una de las entradas de datos y que en caso de entrada de datos inválidos, se muestren mensajes de precaución o de error con su descripción.

Para la aplicación del Método de Caja Negra se empleó la técnica Partición Equivalente por ser una de las más efectivas, pues permite examinar los valores válidos e inválidos de las entradas existentes en el sistema. La Partición de Equivalencia se dirige a la definición de casos de prueba que descubran clases de errores, reduciendo así el número de casos de prueba a desarrollar.

Diseño de Casos de Prueba basado en casos de uso:

Para diseñar los casos de prueba se tienen en cuenta las funcionalidades descritas en los casos de uso. Cada planilla de caso de prueba contiene la especificación de un caso de uso, dividido en secciones y escenarios, definiendo las funcionalidades descritas en él y describiendo cada variable que recoge el caso de uso en cuestión. Para detallar el caso de uso se utiliza una tabla, desglosándose cada funcionalidad en secciones y estas a su vez en escenarios, haciendo así más ventajosa la realización de las pruebas. En la siguiente tabla se presentan los escenarios probados para el caso de uso Dibujar Mapa Cognitivo Difuso.

Nombre del Escenario	Descripción de la funcionalidad
EC 1: Dibujar Mapa Cognitivo Difuso.	Dibuja un Mapa Cognitivo Difuso a partir de una matriz de adyacencia.

Tabla 2: Escenario de prueba para el caso de uso Dibujar Mapa Cognitivo Difuso

Las variables asociadas al caso de uso Dibujar Mapa Cognitivo Difuso se muestran en la siguiente tabla.

No	Nombre del Variable	Clasificación	Valor Nulo	Descripción
Variable 1	Menú	Selector	No	Debe seleccionarse la opción Dibujar Mapa cognitivo Difuso del menú Opciones.

Tabla 3: Descripción de las variables para el caso de uso Dibujar Mapa Cognitivo Difuso

La descripción anterior posibilitó que se realizara la matriz de la tabla 4, donde se evaluó y probó la validez de los datos introducidos en el sistema, específicamente para el caso de uso Dibujar Mapa Cognitivo Difuso. Utilizando un juego de datos válidos e inválidos se registraron, con el empleo de la técnica de partición de equivalencia, los resultados de las pruebas.

Caso de Uso	Variable Menú	Respuesta del Sistema	Resultado de la Prueba	Flujo Central
Dibujar Mapa Cognitivo Difuso	Datos Válidos	El sistema realiza el trazado del Mapa Cognitivo Difuso.	Satisfactorio	El sistema realiza el trazado del MCD correspondiente a la Matriz de Adyacencia importada.
	Datos Inválidos	El sistema devuelve un mensaje debido a que ha ocurrido un error con la Matriz de Adyacencia que se utilizará para el trazado.	Satisfactorio	

Tabla 4: Matriz de datos del caso de uso Dibujar Mapa Cognitivo Difuso

Se efectuaron dos iteraciones de prueba, donde se aplicaron los casos de prueba diseñados, la primera iteración arrojó 21 No Conformidades (NC). A continuación se presentan los resultados obtenidos en la primera iteración de pruebas.

Pruebas/Criterios	C-1	C-2	C-3	C-4	C-5	C-6	C-7	C-8
Módulo	5	2	1	2	3	2	1	2
Aplicación con el Módulo Integrado	-	2	-	-	-	1	-	-

Tabla 5: Resultados de la Primera Iteración de las pruebas

Leyenda de la Tabla:

C-1. Excepciones
C-2. Funcionalidad
C-3. Error de Interfaz
C-4. Formato
C-5. Ortografía
C-6. Validación
C-7. Redacción
C-8. Diseño CP

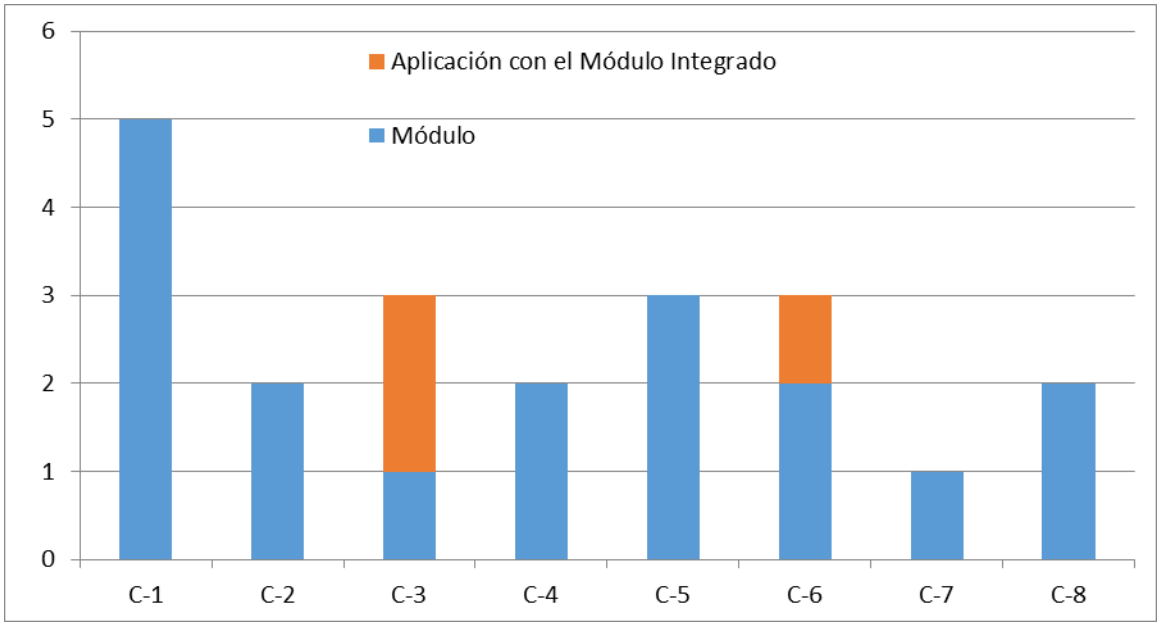


Figura 17: Gráfica de las NC en la primera iteración de las pruebas

Las NC detectadas en la primera iteración de pruebas fueron resueltas en su totalidad, dando paso a una segunda iteración que arrojó los siguientes resultados:

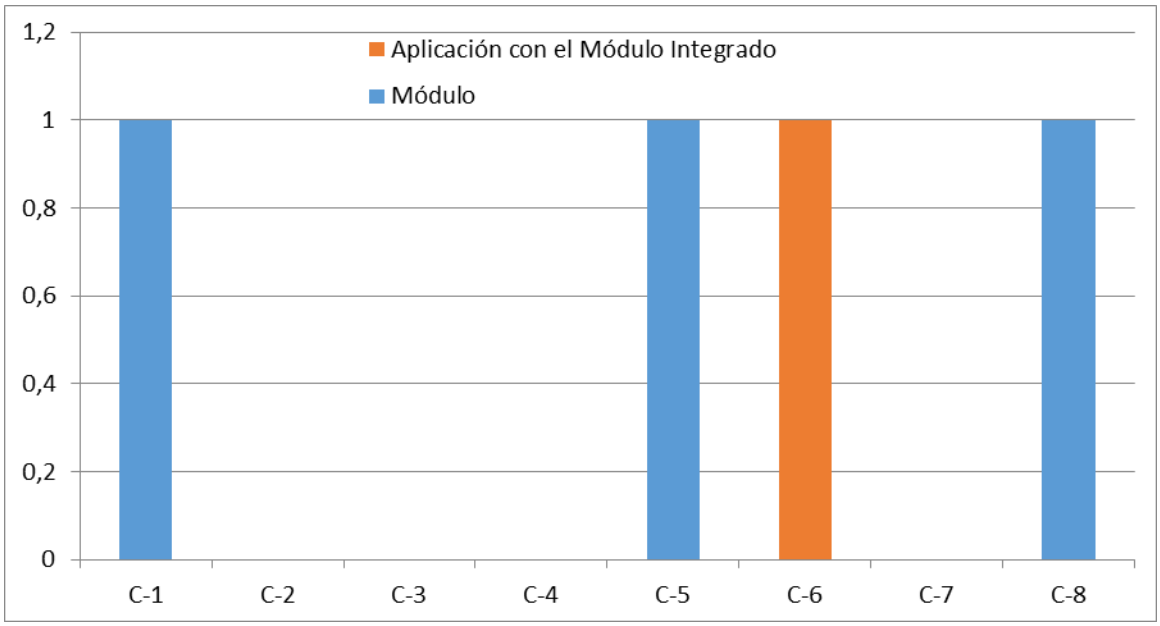


Figura 18: Gráfica de las NC en la segunda iteración de las pruebas

Luego de la segunda iteración se detectaron cuatro NC, las cuales fueron resueltas, culminando el presente trabajo con resultados satisfactorios.

3.5- Interfaces de la aplicación

El módulo, después de integrado a la aplicación *FCM – Decision,* consta de una interfaz principal. En la parte superior se encuentra ubicada la barra de herramienta, esta contiene funcionalidades básicas de la aplicación para copiar y pegar un dispositivo, importar una matriz de adyacencia, dibujar mapa cognitivo difuso y crear una nueva área de trabajo. En el lado izquierdo se encuentran los componentes que pueden ser insertados en el área de trabajo y en el derecho se muestran los nodos del MCD trazado. En la región central se localiza el área de trabajo, lugar donde se observarán las matrices de adyacencia importadas. A continuación se muestra en una vista de la aplicación, una pestaña con una matriz de adyacencia importada.

Figura 19: Vista de una Matriz de Adyacencia importada por el usuario

En el área de trabajo, se observarán además los Mapas Cognitivos Difusos después de ser trazados. La siguiente vista pertenece al trazado del Mapa Cognitivo Difuso correspondiente a la matriz anterior, en esta se muestran conceptos definidos para determinar el Trastorno del Espectro Autista.

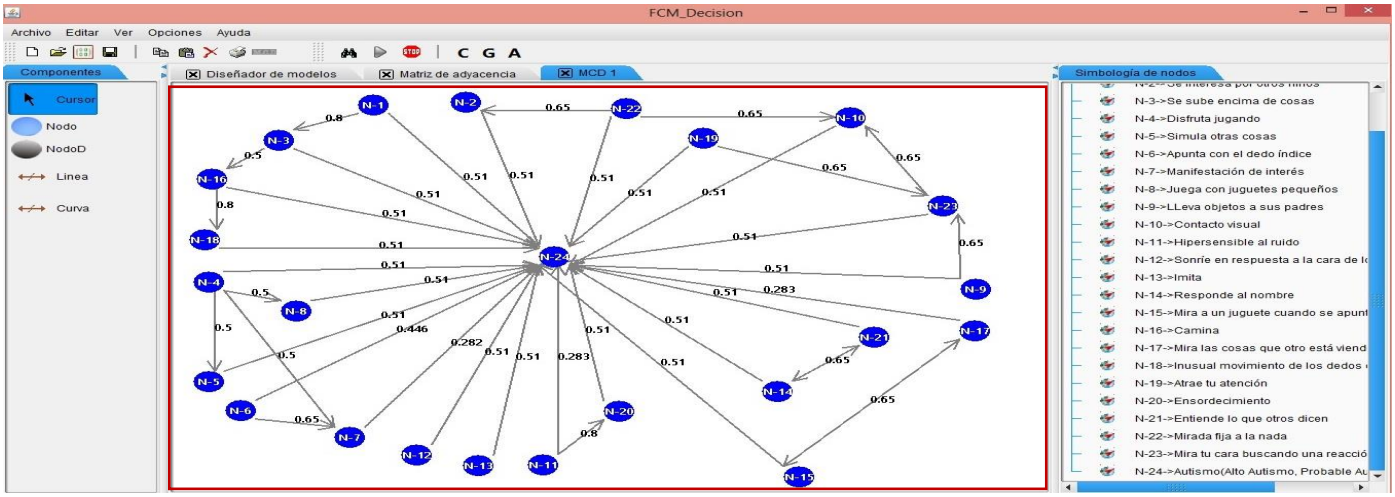


Figura 20: Vista del trazado del Mapa Cognitivo Difuso con conceptos definidos para determinar el Trastorno del Espectro Autista

Conclusiones del capítulo

En el presente capítulo se elaboró el diagrama de componentes representando la organización y dependencias entre las clases del diseño estructuradas en paquetes de componentes. Se definió un estándar de codificación con el objetivo de garantizar la uniformidad del código en el desarrollo del módulo. Se evaluó el funcionamiento del módulo antes y luego de ser integrado con la aplicación *FCM – Decision*, validando el cumplimiento de los requisitos funcionales establecidos en la fase inicial del proceso de desarrollo del módulo y constatando el correcto funcionamiento del mismo. Para determinar que el software posee una calidad aceptable se mostraron los resultados obtenidos durante la realización de las pruebas y las principales interfaces del mismo.

CONCLUSIONES

Teniendo en cuenta los objetivos trazados en la investigación y en aras de dar cumplimiento al problema propuesto, se arribó a las siguientes conclusiones:

- Se realizó un estudio del marco conceptual referencial sobre el modelado de Mapas Cognitivos Difusos y la teoría de grafos.
- Se seleccionaron las herramientas y tecnologías a utilizar para el desarrollo del módulo.
- Se realizó el análisis y diseño del módulo de Diseño Automático de Mapas Cognitivos Difusos obteniéndose los artefactos necesarios para el desarrollo del mismo.
- Se realizó la implementación del módulo cumpliendo con los requisitos funcionales identificados en las fases de análisis y diseño.
- El diseño y ejecución de las pruebas permitió comprobar el correcto funcionamiento del módulo, luego de resolver las No Conformidades detectadas.
- Se integró exitosamente el módulo Diseño Automático de Mapas Cognitivos Difusos a la herramienta *FCM – Decision*.

Con el desarrollo de esta investigación se alcanzó satisfactoriamente el objetivo propuesto, pues se desarrolló el módulo Diseño Automático de Mapas Cognitivos Difusos para la herramienta de modelado y análisis *FCM – Decision*.

RECOMENDACIONES

Al concluir este trabajo se recomienda:

- Desarrollar otras convenciones en las que se pueda realizar el trazado del Mapa Cognitivo Difuso.
- Realizar los cambios necesarios para realizar el trazado de los Mapas Cognitivos Difusos en 3D lo cual puede facilitar, que el usuario visualice el Mapa Cognitivo Difuso desde cualquier ángulo.

REFERENCIAS BIBLIOGRÁFICAS

1. ***White, E y Mazlack, L. J.*** *Discerning suicide notes causality using fuzzy cognitive maps. Fuzzy Systems : IEEE International Conference on, 2011.*

2. ***PING, C. W.*** *A Methodology for Constructing Causal Knowledge Model from Fuzzy Cognitive Map to. s.l. : Department of Computer Science, Chonnam National University. PHD., 2009.*

3. ***Kosko, B.*** *"Fuzzy cognitive maps." International Journal of Man-Machine Studies. 1986.*

4. ***Salmeron, J. L.*** *Supporting decision makers with Fuzzy Cognitive Maps. Research-Technology Management. 2009.*

5. ***Zhi-Qiang, L. I. U.*** *Causation, bayesian networks, and cognitive maps. s.l. : ACTA AUTOMATICA SINICA, 2001.*

6. ***BuenasTareas.*** *Definicion De Causalidad Teoria De Piaget. BuenasTareas. [En línea] 2011. http://www.buenastareas.com/ensayos/Definicion-De-Causalidad-Teoria-De-Piaget/1424974.html.*

7. ***Aguilar, Jose y Contreras, Jose.*** *Los Mapas Cognitivos Difusos Dinámicos en Tareas de Control de Sistemas. Venezuela : s.n., 2008.*

8. ***Quispe-Otazu, Rodolfo.*** *Blog de Rodolfo Quispe-Otazu . [En línea] 2007. Disponible en: http://www.rodolfoquispe.org/blog/que-es-un-algoritmo.php.*

9. ***Hernández, Luis Alberto García.*** *Técnicas de visualización para crawlers. Jwire, un caso práctico. Salamanca : s.n., 2010.*

10. ***Morfeo Formación.*** *[En línea] 2009. [Citado el: 01 de 03 de 2013.] http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6#OpenUP.*

11. Gestión de Proyectos. *[En línea] 27 de 09 de 2008. [Citado el: 01 de 03 de 2013.] http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html.*

12. Rodríguez Martini, Jorge Luis y Bradshaw Benzant, Maikel. *Análisis, diseño e implementación del Portal WAP del SIIPOL Movil. La Habana : s.n., 2010.*

13. UML, Visual Paradigm for. Sitio de Descargas de Software. Visual Paradigm for UML (ME). *[En línea] 5 de 03 de 2007. [Citado el: 16 de 11 de 2012.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.*

14. González Aguilera, Julio C. *El cuidado multidisciplinario. El cuidado multidisciplinario. [En línea] 2020. http://www.encolombia.com/medicina/enfermeria/enfermeria6203-editorial.htm .*

15. Netbeans. Netbeans. NetBeans IDE 7.2.1 Release Information. *[En línea] 2012. [Citado el: 16 de 11 de 2012.] http://netbeans.org/community/releases/72/index.html.*

16. Iec.csic. *[En línea] 1999. [Citado el: 16 de 11 de 2012.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.*

17. Oracle. *[En línea] 2012. [Citado el: 20 de 11 de 2012.] http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html.*

18. Roger S. Pressmqn, Ph.D. *Software Engineering: A Practitioner´s Approach. New York : Higher Education, 2005.*

19. Reynoso, C. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft. *[En línea] 2004. [Citado el: 20 de 11 de 20012.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#10.*

20. Rolando, Daniel, Geonías y Valdivia, Eduardo Espinosa. *Estándares de Calidad para las pruebas de Software, Pruebas de Software. [En línea] 2004. https://forja.mondragon.edu.*

21. García-Retamero R, Hoffrage U. *How causal knowledge simplifies decisionmaking. s.l. : Minds Mach, 2006.*

22. Oré, Ing. Alexander. *Calidad y Software. 2010.*

23. López Pérez, Carmelo. *Modelo de Madurez de la Capacidad del Software. Región de Murcia : s.n., 2004.*

24. Consulting, Milestone. Milestone consulting. Curso práctico de Modelado de Negocios con BPMN 2.0 y UML. *[En línea] 3 de 2 de 2012. [Citado el: 16 de 11 de 2012.] http://www.milestone.com.mx/CursoModeladoNegociosBPMN.htm.*

BIBLIOGRAFÍA

1. ***Aguilar, Jose y Contreras, Jose.*** *Los Mapas Cognitivos Difusos Dinámicos en Tareas de Control de Sistemas. Venezuela : s.n., 2008.*

2. ***BuenasTareas.*** *Definicion De Causalidad Teoria De Piaget. BuenasTareas. [En línea] 2011. http://www.buenastareas.com/ensayos/Definicion-De-Causalidad-Teoria-De-Piaget/1424974.html.*

3. ***Consulting, Milestone. Milestone consulting. Curso práctico de Modelado de Negocios con BPMN 2.0 y UML.*** *[En línea] 3 de 2 de 2012. [Citado el: 16 de 11 de 2012.] http://www.milestone.com.mx/CursoModeladoNegociosBPMN.htm.*

4. ***Di Batista, Giuseppe, y otros.*** *Graph Drawing. Algorithms for the Visualization Graphs. New Jersey : Prentice - Hall, Inc., 1999.*

5. ***Drawing Graphs Nicely Using Simulated Annealing. Davidson, Ron y Harel, David.*** *4, The Weizmann Institute of Science. : ACM Transactions on Graphics,, 1996, Vol. 15.*

6. ***García-Retamero R, Hoffrage U.*** *How causal knowledge simplifies decisionmaking. s.l. : Minds Mach, 2006.*

7. ***Gestión de Proyectos****. [En línea] 27 de 09 de 2008. [Citado el: 01 de 03 de 2013.] http://kasyles.blogspot.com/2008/09/openup-como-alternativa-metodolgica.html.*

8. ***González Aguilera, Julio C****. El cuidado multidisciplinario. El cuidado multidisciplinario. [En línea] 2020. http://www.encolombia.com/medicina/enfermeria/enfermeria6203-editorial.htm.*

9. ***Groves, Lindsay J., y otros.*** *Genetic Algorithms for Drawings Directed Graphs. Deparment of Computing Sciense, Victoria University, Wellington, New Zeland. : s.n., 2012.*

10. ***Hernández, Luis Alberto García.*** *Técnicas de visualización para crawlers. Jwire, un caso práctico. Salamanca : s.n., 2010.*

11. ***Iec.csic.*** *[En línea] 1999. [Citado el: 16 de 11 de 2012.] http://www.iec.csic.es/criptonomicon/java/quesjava.html.*

12. ***Kosko, B.*** *"Fuzzy cognitive maps." International Journal of Man-Machine Studies. 1986.*

13. ***López Pérez, Carmelo.*** *Modelo de Madurez de la Capacidad del Software. Región de Murcia : s.n., 2004.*

14. ***Modelado y análisis de los Factores Críticos de Éxito de los proyectos de software mediante Mapas Cognitivos Difusos. Leyva Vázquez, Maikel Y., Rosado Rosello, Reynaldo y Febles Estrada, Ailyn.*** *2, s.l. : Ciencias de la Computación, 2012, Vol. 43.*

15. ***Morfeo Formación.*** *[En línea] 2009. [Citado el: 01 de 03 de 2013.] http://formacion.morfeo-project.org/wiki/index.php/PT3:GPSL:UF6#OpenUP.*

16. ***Netbeans. Netbeans. NetBeans IDE 7.2.1 Release Information.*** *[En línea] 2012. [Citado el: 16 de 11 de 2012.] http://netbeans.org/community/releases/72/index.html.*

17. ***Oracle.*** *[En línea] 2012. [Citado el: 20 de 11 de 2012.] http://www.oracle.com/technetwork/java/javase/overview/index-jsp-136246.html.*

18. ***Oré, Ing. Alexander.*** *Calidad y Software. 2010.*

19. ***PING, C. W.*** *A Methodology for Constructing Causal Knowledge Model from Fuzzy Cognitive Map to. s.l. : Department of Computer Science, Chonnam National University. PHD., 2009.*

20. ***Quispe-Otazu, Rodolfo.*** *Blog de Rodolfo Quispe-Otazu . [En línea] 2007. Disponible en: http://www.rodolfoquispe.org/blog/que-es-un-algoritmo.php.*

21. ***Reynoso, C. Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.*** *[En línea] 2004. [Citado el: 20 de 11 de 20012.] http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/style.asp#10.*

22. ***Rodríguez Martini, Jorge Luis y Bradshaw Benzant, Maikel.*** *Análisis, diseño e implementación del Portal WAP del SIIPOL Movil. La Habana : s.n., 2010.*

23. ***Roger S. Pressman, Ph.D.*** *Software Engineering: A Practitioner´s Approach. New York : Higher Education, 2005.*

24. ***Rolando, Daniel, Geonías y Valdivia, Eduardo Espinosa. Estándares de Calidad para las pruebas de Software, Pruebas de Software.*** *[En línea] 2004. https://forja.mondragon.edu.*

25. ***Salmeron, J. L.*** *Supporting decision makers with Fuzzy Cognitive Maps. Research-Technology Management. 2009.*

26. ***Técnicas para la representación del conocimiento causal: un estudio de caso en Informática Médica. MSc. Maikel Leyva-Vázquez, MSc. Karina Pérez-Teruel, Dra. C. Ailyn Febles-Estrada,Dr.C. Jorge Gullín-González.*** *Universidad de las Ciencias Informáticas (UCI), La Habana, Cuba. : s.n., 2013.*

27. ***Trazado de grafos mediante métodos dirigidos por fuerzas: Revisión del estado del arte y presentación de algoritmos para grafos donde los vértices son regiones geográficas. Aiello, Andrés y Ignasio Silveira, Rodrigo.*** *Universidad de Buenos Aires : s.n., 2004.*

28. ***UML, Visual Paradigm for. Sitio de Descargas de Software. Visual Paradigm for UML (ME).*** *[En línea] 5 de 03 de 2007. [Citado el: 16 de 11 de 2012.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_%28M%C3%8D%29_14720_p/.*

29. ***Una Solución Flexible y Eficiente para el trazado de grafos basada en el Escalador de Colinas Estocástico. Rosete Suárez, Alejandro.*** *La Habana : s.n., 2000.*

30. ***White, E y Mazlack, L. J.*** *Discerning suicide notes causality using fuzzy cognitive maps. Fuzzy Systems : IEEE International Conference on, 2011.*

31. ***Zhi-Qiang, L. I. U.*** *Causation, bayesian networks, and cognitive maps. s.l. : ACTA AUTOMATICA SINICA, 2001.*

ANEXOS

Anexo 1: Diagrama de clases del diseño del caso de uso: Administrar Matriz de Adyacencia

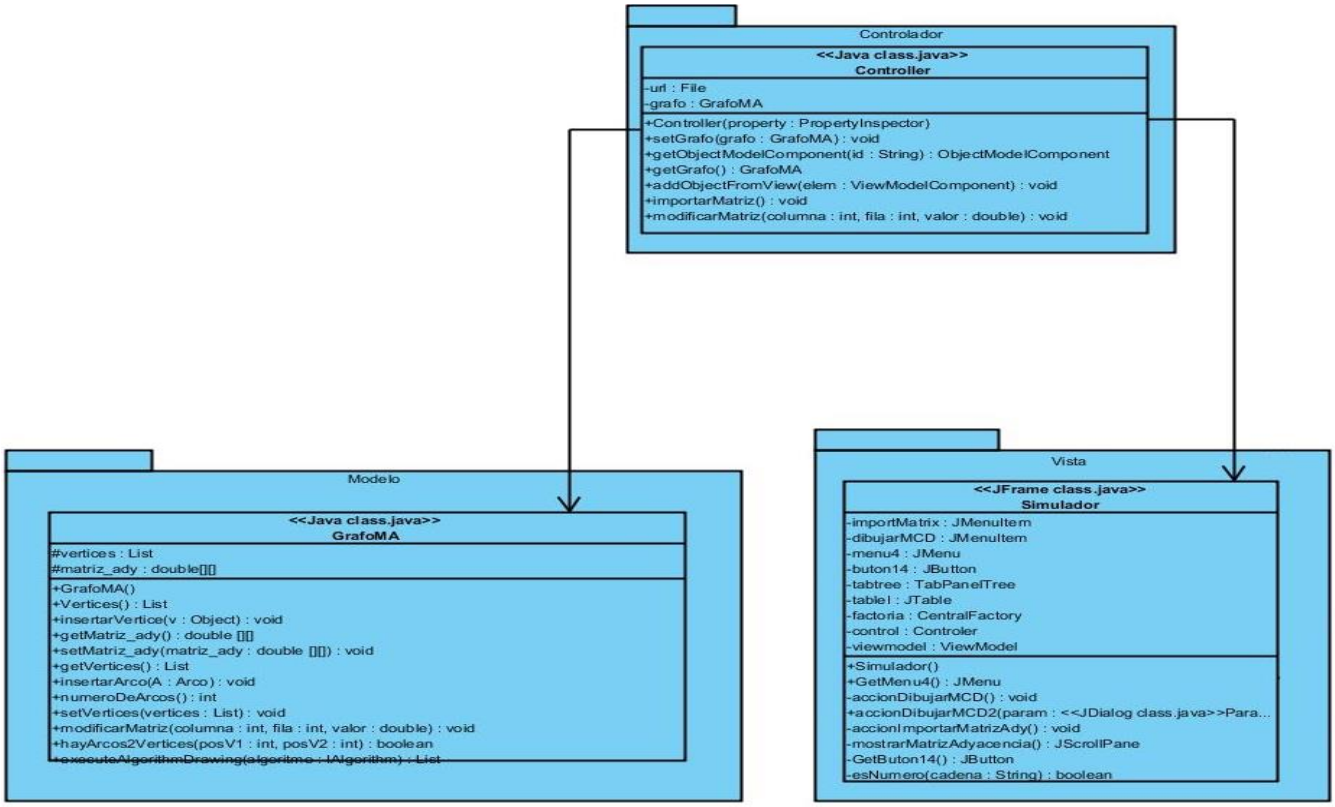


Figura 21: Diagrama de clases del diseño del caso de uso Administrar Matriz de Adyacencia

Anexo 2: Diagramas de secuencia del caso de uso: Administrar Matriz de Adyacencia

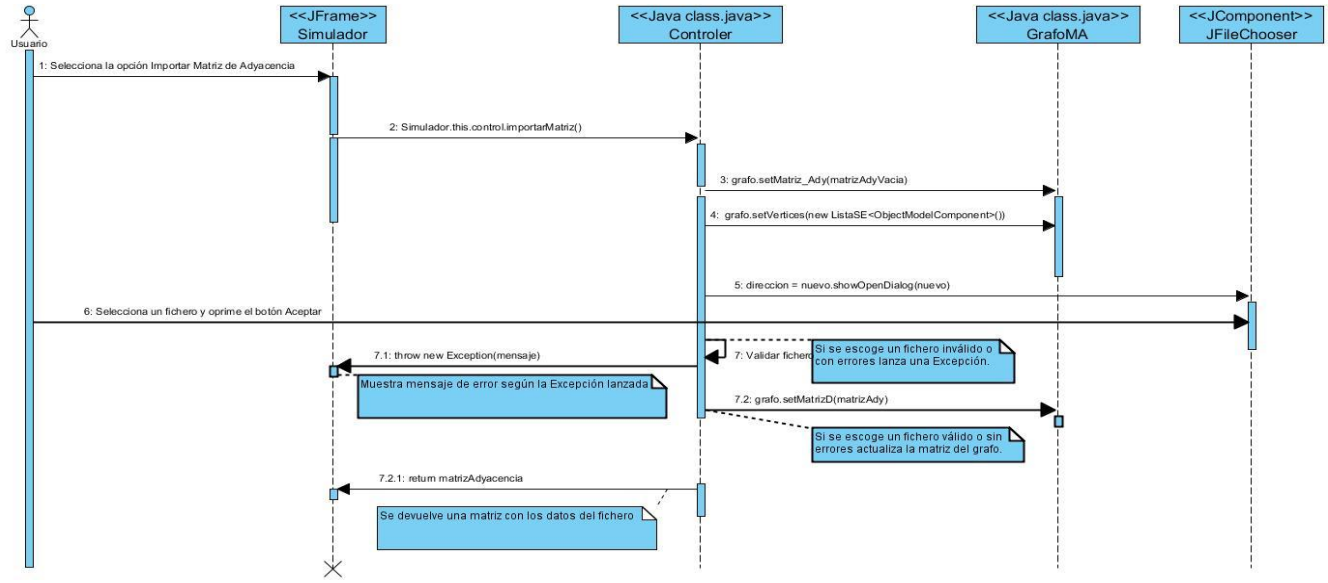


Figura 22: Escenario Importar Matriz de Adyacencia

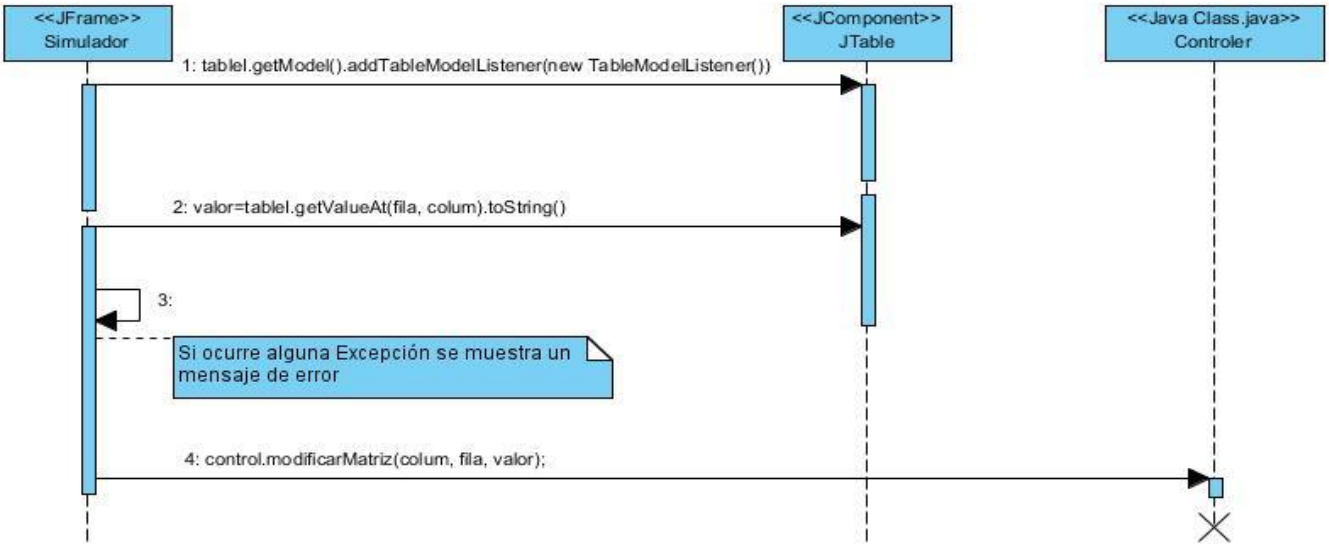


Figura 23: Escenario Modificar Matriz de Adyacencia

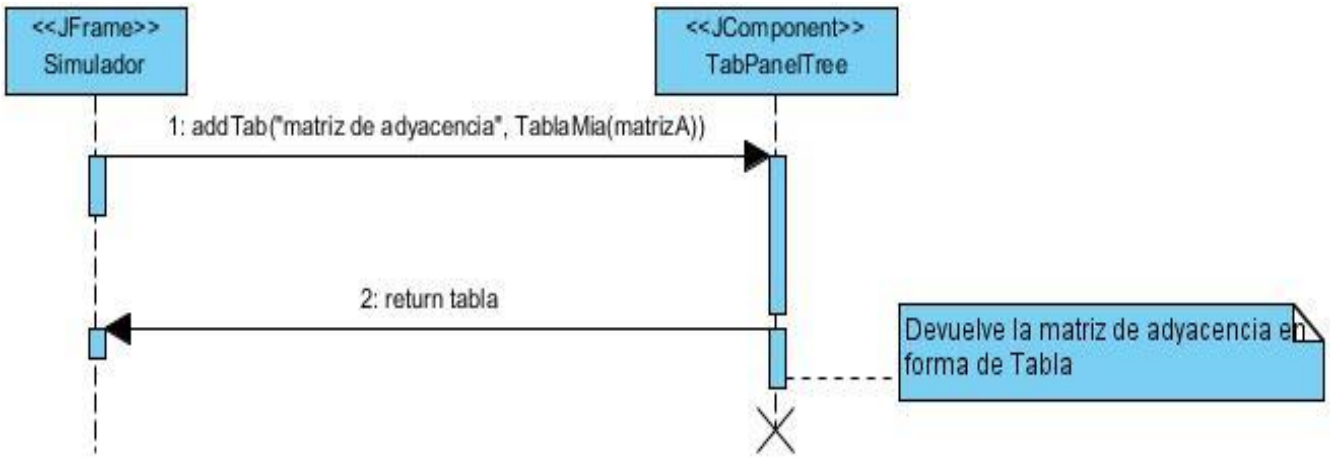


Figura 24: Escenario Visualizar Matriz de Adyacencia

GLOSARIO DE TÉRMINOS

Algoritmo: es una secuencia finita de instrucciones que se utilizan para ejecutar una tarea o resolver un problema determinado.

API: (*Application Programming Interface* - Interfaz de Programación de Aplicaciones) es un conjunto de convenciones internacionales que definen cómo debe invocarse una determinada función de un programa desde una aplicación.

Causalidad: es el fenómeno mediante el cual se relacionan causas con efectos. Es la conexión que existe entre las razones o las causas de ciertos fenómenos o procesos y los resultados o efectos de los mismos. La noción de causalidad implica así una permanente relación entre un evento anterior y su continuación, además de formarse así un círculo infinito de conexión entre sucesos y eventos que se generan unos a otros.

Inferencia causal:

Inferencia: el proceso de usar los datos para extraer conclusiones más amplias sobre conceptos e hipótesis que son el objeto de la investigación.

Causal: el proceso de conseguir conclusiones sobre la causalidad de acuerdo con los datos observados.

Inteligencia artificial: es la ciencia que estudia y analiza el comportamiento humano. De esta manera, las aplicaciones de la IA se sitúan principalmente en la simulación de actividades intelectuales del hombre. Es decir, imitar por medio de máquinas, normalmente electrónicas, tantas actividades mentales como sea posible, y quizás llegar a mejorar las capacidades humanas en estos aspectos.

Mapas Cognitivos: son redes capaces de adquirir, aprender, codificar y decodificar conocimiento/información, con respecto a eventos causales.

Mapas Cognitivos Difusos: fueron introducidos por Bart Kosko, quien le da significado a esta nueva representación como un grafo capaz de codificar conocimiento empleando lógica difusa. Son una herramienta de representación causal, están compuestos por conceptos y relaciones causales, los cuales usan la teoría de lógica difusa para describir su estructura. Mejoran los mapas cognitivos al describir la fortaleza de la relación mediante el empleo de valores borrosos en el intervalo [-1,1]. (7)

Matriz de adyacencia: sea G un grafo dirigido con un conjunto de nodos o vértices V = {1, 2,…n} y un conjunto de arcos o aristas. La matriz de adyacencia de G es la matriz M: n x n en la cual el elemento en la fila i y columna j es el peso de la arista que va desde el vértice i al vértice j.

Modelos causales: constituyen instrumentos prácticos que son empleados frecuentemente para comprender los sistemas complejos. A partir de los modelos causales se pueden establecer las causas de algunos eventos y predecir sus efectos. (21)

Razonamiento causal: es un proceso de lógica mediante el cual, partiendo de uno o más juicios, se deriva la validez, la posibilidad o la falsedad de otro juicio distinto. El estudio de los argumentos corresponde a la lógica, de modo que a ella también le corresponde indirectamente el estudio del razonamiento. Por lo general, los juicios en que se basa un razonamiento expresan conocimientos ya adquiridos o, por lo menos, postulados como hipótesis.