Universidad de las Ciencias Informáticas

Facultad 6

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Título: "Interfaz gráfica de usuario de flujo de trabajo para el cálculo de los descriptores moleculares basados en teoría de la información"

Autor: Alejandro Riverón Sagastume

Tutores: Mcs. Longendri Aguilera Mendoza

Ing. Osvel Chávez Hernández

La Habana, junio de 2014

"Año 55 de la Revolución"

Frase

“El futuro de nuestra patria tiene que ser necesariamente un futuro de hombres de ciencia, tiene que ser un futuro de hombres de pensamiento, porque precisamente es lo que más estamos sembrando; lo que más estamos sembrando son oportunidades a la inteligencia (...)”

Fidel Castro Ruz

Declaración de autoría

Declaración de autoría

Declaro ser el autor del presente trabajo de diploma y reconozco a la Universidad de las Ciencias Informáticas los derechos patrimoniales del mismo, con carácter exclusivo. Autorizo a dicho centro para que haga el uso que estime pertinente con este trabajo.

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

Alejandro Riverón Sagastume

_____

Firma del autor

Mcs.Longendri Aguilera Mendoza

_____

Firma del tutor

Ing. Osvel Chavez Hernández

_____

Firma del tutor

Datos del contacto

Mcs. Longendri Aguilera Mendoza:
Licenciado en Ciencias de la Computación.
Máster en Bioinformática.
Profesor Asistente.
e-mail: loge@uci.cu

Ing. Osvel Chávez Hernández:
Ingeniero en Ciencias Informáticas.
Recién graduado en adiestramiento.
e-mail: ochavez@uci.cu

Agradecimientos

En los agradecimientos casi siempre se cometen injusticias, pues la memoria es a menudo traicionera. Pero aun sabiendo que no existirá una forma para agradecerles, hoy sin embargo, puedo afirmar sin lugar a dudas, que este trabajo no hubiese sido posible sin la ayuda de:

Mi familia en general, especialmente a mis padres y abuelos, les agradezco hoy, por todo el tiempo que pasaron pensando en mí, porque lo que soy se lo debo a ellos.

Mi novia Jisel por su amor, paciencia, respeto y estar estos cinco años junto a mí apoyándome en todo momento. También a su familia por acogerme como uno más del núcleo familiar.

Los grandes amigos que hice a lo largo de estos cinco años y compartieron momentos de estrés, felicidad y tristeza principalmente a Héctor.

Mis amigos más cercanos por apoyarme en la vocacional y aquí en la UCI, por estar siempre ahí cuando los necesitaba, por compartir momentos felices juntos: Ernesto, Coco y Ledián.

Mis tutores por hacer lo mejor que pudieron y guiarme por el camino correcto a pesar de todo su cúmulo de trabajo.

Todos Profesores con que he podido interactuar a lo largo de todo este tiempo y que me han ayudado a formarme tanto docentemente como en lo referente a todos los valores humano necesarios para la vida.

Resumen

Varios estudios se han centrado en cómo obtener y convertir por una vía teórica la información codificada en la estructura molecular en uno o más números, llamados descriptores moleculares. El campo de descriptores moleculares es fuertemente interdisciplinario e involucra una gran cantidad de teorías diferentes dentro de las que se encuentra la teoría de la información. En nuestros días los descriptores moleculares basados en teoría de la información analizando a la molécula como canal de información no han sido utilizados para la solución de un problema real. Con esta finalidad, a principios del año 2013, la Universidad Central de las Villas Martha Abreu en colaboración con la Universidad de las Ciencias Informáticas, desarrollaron un API[1] para el cálculo de estos descriptores, esta API no cuenta con una interfaz gráfica de usuario donde los investigadores puedan interactuar, por lo que surge la necesidad de desarrollar una interfaz gráfica de usuario que permita la utilización y puesta en práctica de los descriptores moleculares (implementados en el API) por parte de los investigadores. Durante la realización de la investigación se hizo un estudio de las principales metodologías de desarrollo de software, herramientas y tecnologías a utilizar. Se realizó el proceso de análisis y diseño de la aplicación y posteriormente la implementación y prueba del mismo. Como resultado se obtuvo una aplicación informática basada en la Plataforma Netbeans construida a partir de varios módulos que permite el diseño gráfico del proceso de cálculo de los descriptores moleculares en forma de flujo de trabajo.

Palabras claves: descriptores moleculares basados en teoría de la información, flujo de trabajo, interfaz gráfica de usuario (GUI).

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

Abstract

Abstract

Some studies have focused on how to obtain and transform, theoretically, the information coded in the molecular structure of one or more numbers, the so called molecular descriptors. The field of molecular descriptors is strongly interdisciplinary and involve different theories like the information theory. Nowadays, molecular descriptors, considering the molecule as a channel of information, haven't been used in the solution of any real problem. Due to this situation, at the beginning of 2013, the Central University ¨Martha Abreu¨ de las Villas, in collaboration with the University of Informatics Sciences, developed an API to calculate these descriptors. This API does not have an user graphic interface where researchers could interact; therefore it was necessary to develop an user graphic interface that allow the researchers the setup and utilization of molecular descriptors, implemented in API. A study about the most important methodologies of software development, programing languages, tools and technologies was carried out. The process of analysis and design of the application, and later on its implementation and testing, was also carried out. As a result, an informatics application was obtained based on the Netbeans platform built from various modules that allow the graphic design for calculating the molecular descriptors as workflow.

Keywords: graphical user interface (GUI), molecular descriptors based on information theory, workflow.

Índice General

Índice General

Índice General

Índice de Figuras

Índice de figuras

Índice de tablas

Índice de tablas

Introducción

Como consecuencia del creciente desarrollo tecnológico alcanzado en la actualidad, muchas instituciones y empresas se han dado la tarea de automatizar sus procesos, de forma tal de que estos sean ejecutados con mayor rapidez y con menor cantidad de recursos humanos, económicos y de cómputos. Esto ha permitido un aumento considerable en la calidad de las actividades que se realizan en estas entidades.

Una de las áreas científicas beneficiadas con este auge tecnológico es la quimioinformática[2], al punto que se pueden realizar complejos cálculos matemáticos que hace varios años parecían inadmisibles. Permitiendo así mejorar nuestra compresión sobre el comportamiento de las moléculas en los sistemas biológicos (2).

En las últimas décadas, varios estudios se han centrado en cómo obtener y convertir por una vía teórica la información codificada en la estructura molecular en uno o más números, llamados descriptores moleculares, para establecer relaciones cuantitativas entre estructuras y propiedades, las actividades biológicas y otras propiedades experimentales; lo que se traduce a la teoría Quantitative Structure–Activity/Structure-property Relationships, (QSAR/QSPR por sus siglas en inglés) (2).

El campo de descriptores moleculares es fuertemente interdisciplinario e involucra una gran cantidad de teorías diferentes. Para la definición de los descriptores moleculares, se requieren conocimientos de álgebra, teoría de grafos, química computacional, teorías de la reactividad química-orgánica-física y de teoría de la información generalmente (3).

La teoría de información ha tenido aplicaciones en distintas ramas de la ciencia, lo cual se justifica en el paradigma de que principios comunes rigen el universo como una fuente de información en general. En los últimos años, la aplicación de la teoría de información a la química ha recibido creciente interés, tanto con el objetivo de proporcionar mejor interpretación a conceptos químicos tradicionales como para caracterizar fenómenos y especies químicas, mediante la utilización de descriptores moleculares. La

[2] *Es la mezcla de aquellos recursos informáticos para transformar datos en información e información en conocimiento, con el propósito hacer más rápidas mejores decisiones en el ámbito de la identificación de los fármacos cabeza de serie (o de molécula inicial) y la optimización de fármacos. (*F.K. Brown, 1998*).*

amplia utilización de las ideas de la teoría de información en distintas ramas de la ciencia está ligada al hecho de que la misma es fundamentalmente una teoría matemática. Sus conceptos principales solo se determinan a través de las probabilidades de acontecimientos, a los cuales se les puede atribuir diverso contenido físico (1).

Aunque el número de descriptores moleculares ha aumentado considerablemente, la búsqueda de nuevos descriptores moleculares sigue siendo de interés científico en aras de mejorar la diversidad de estos, y recoger información estructural no codificada adecuadamente por el conjunto de descriptores moleculares existentes.

Aún en nuestros días los descriptores moleculares basados en teoría de la información analizando a la molécula como canal de información y no como fuente de información, no han sido utilizados en la solución de un problema real, por lo que a principios del año 2013 la Universidad Central de las Villas Martha Abreu (UCLV) en conjunto con la Universidad de las Ciencias Informáticas (UCI) desarrollaron un API para el cálculo de nuevos índices basados en la teoría de la información, la cual tiene definida una serie de tareas, mediante las cuales se establece un orden lógico de ejecución para la realización del cálculo. Esta API puede ser utilizada por otras aplicaciones pero no por los usuarios finales, debido a que no cuenta con una interfaz gráfica donde los usuarios puedan interactuar.

Por la situación antes expuesta se plantea como **problema de la investigación:** El API implementado por el grupo de Bioinformática de la UCI para el cálculo de descriptores moleculares basados en teoría de la información no permite la utilización y puesta en práctica de dichos descriptores por los investigadores en el campo de la química computacional sin conocimientos previos de desarrollo de software.

Se define como **objeto de estudio** de la investigación: Las interfaces gráficas de usuarios. El **campo de acción** ha sido enmarcado en: Las interfaces gráficas de usuarios para el cálculo de los descriptores moleculares basados en teoría de la información.

El **objetivo general** de la investigación es: Desarrollar una interfaz gráfica de usuario que permita la utilización y puesta en práctica de los descriptores moleculares (implementados en el API) basados en teoría de la información por parte de los investigadores.

Para dar cumplimiento al objetivo planteado se proponen las siguientes **tareas de investigación:**

- Revisión bibliográfica sobre las interfaces gráficas de usuarios para identificar los elementos a tener en cuenta en el desarrollo de la investigación.
- Estudio de la interfaz gráfica de usuario de herramientas que servirán como base para el desarrollo de la aplicación informática que facilitará el cálculo de los descriptores moleculares basados en teoría de la información.
- Identificación de los requisitos de la herramienta informática que realizará el cálculo de los descriptores moleculares basados en teoría de la información para su posterior implementación.
- Realización de los artefactos y diagramas correspondientes para implementar la herramienta informática que facilitará el cálculo de los descriptores moleculares basados en teoría de la información.
- Implementación de las funcionalidades de la herramienta informática que realizará el cálculo de los descriptores moleculares basados en teoría de la información para su posterior utilización.
- Diseño y ejecución de los casos de prueba correspondientes a las funcionalidades de la herramienta informática que facilitará el cálculo de los descriptores moleculares basados en teoría de la información para su correcta validación.

Preguntas científicas:

- ¿Cuáles son los fundamentos teóricos asociados con la realización de interfaces gráficas de usuarios?
- ¿Qué metodología, herramientas y tecnologías utilizar para el desarrollo e implementación de la interfaz gráfica de usuario para el cálculo de los descriptores moleculares basados en teoría de la información?
- ¿Cómo estructurar el proceso de desarrollo de la interfaz gráfica para que permita a los investigadores utilizar los descriptores moleculares implementados en el API?
- ¿La interfaz gráfica de usuario desarrollada permite diseñar gráficamente el proceso de cálculo de los descriptores moleculares basados en teoría de la información?

Introducción

Para el cumplimiento del objetivo general planteado se utilizarán los siguientes **métodos de investigación**:

Métodos teóricos:

Análisis y síntesis: Para la elaboración del marco teórico referencial, con la finalidad de seleccionar los elementos que sirven de base en la interfaz gráfica de usuario, así como en el diagnóstico inicial y final del problema y la formulación de las conclusiones.

Histórico y lógico: Permitió el estudio del desarrollo histórico sobre la interfaz gráfica de usuario y para los referentes teóricos y metodológicos.

Modelación: Para realizar los diagramas correspondientes a los modelos de análisis, diseño e implementación para el cálculo de los descriptores moleculares basados en teoría de la información.
Métodos empíricos:

Análisis documental: Es utilizado para realizar una revisión detallada de la literatura y documentos relacionados con las interfaces gráficas de usuarios y los descriptores moleculares basados en teoría de la información.

Tormenta de ideas: Se utiliza para determinar que funcionalidades debe poseer la aplicación informática, obteniendo las opiniones de los especialistas del Centro de Matemática Computacional y ver las expectativas que ellos persiguen con la aplicación.

El presente trabajo está compuesto por introducción, tres capítulos, conclusiones, recomendaciones y referencia bibliográfica. Los capítulos están estructurados de la siguiente manera:

Capítulo 1. Fundamentos Teóricos: En este capítulo se presentan los elementos teóricos que sirven de base a la investigación para darle solución al problema planteado, analizando los principales conceptos relacionados con los descriptores moleculares basados en teoría de la información y con las interfaces gráficas de usuario. Se presentan las herramientas y metodología a utilizar para el desarrollo del sistema propuesto.

Capítulo 2. Análisis y diseño de la aplicación El presente capítulo muestra el modelo de dominio, así

como la especificación de los requisitos funcionales y no funcionales de la aplicación, se presenta el diagrama de casos de usos del sistema para identificar la relación entre los actores y casos de usos. Además se brinda una descripción de la arquitectura del sistema, los patrones de diseños más utilizados y la aplicación de los mismos para la confección de los diagramas del modelo del diseño.

Capítulo 3. Implementación y pruebas: En este capítulo, se expondrá el diagrama de componentes, donde se describen los elementos físicos del sistema y sus relaciones. Además se explicarán elementos necesarios para la extensibilidad e implementación de los componentes que formaran parte del proceso de cálculo de los descriptores moleculares basados en teoría de la información. Por último, se describen detalladamente las pruebas que se le realizan al sistema, para corroborar el correcto funcionamiento de la aplicación cumpliendo con los requisitos implementados.

Capítulo 1: Fundamentos teóricos

En este capítulo se abordan conceptos fundamentales asociados a los descriptores moleculares basados en la teoría de la información, así como elementos a tener en cuenta para realizar los cálculos. Además se describen las metodologías y las herramientas que posibilitan el desarrollo del flujo de trabajo para tales descriptores.

1.1 Conceptos asociados al dominio del problema

A continuación se abordan algunos conceptos esenciales que permitirán un mayor entendimiento del dominio del problema planteado, lo cual facilitará la elaboración de la propuesta de solución a dicho problema.

1.1.1 Descriptores moleculares

En la actualidad, la definición más reconocida de un descriptor molecular (DM), es la propuesta por Todeschini y Consonni quienes plantean que, *"es el resultado final de un procedimiento lógico y matemático que transforma información química codificada dentro de una representación simbólica de una molécula en un número de utilidad o el resultado de algún experimento estandarizado".* Aquí el término "utilidad" tiene un doble sentido y puede significar que el número obtenido nos brinde información que nos permita interpretar propiedades moleculares y/o ese número permite obtener modelos matemáticos capaces de predecir alguna propiedad de interés a nuevas moléculas (3).

Los descriptores moleculares son utilizados por la teoría QSAR/QSPR en busca de cuantificar las relaciones, a través del desarrollo de modelos y la combinación de métodos de la estadística matemática con la quimioinformática. Tales modelos son importantes a la hora de predecir el valor de la propiedad de una sustancia si ésta es desconocida o resulta difícil de adquirir, sea por su inestabilidad, toxicidad o costo económico. Así también la teoría ha sido ampliamente utilizada para el diseño y optimización de compuestos tipo-droga, y hasta para inferir resultados sobre mecanismos de reacción de compuestos orgánicos (1).

1.1.3 Teoría de Información

La teoría de información trata todos los problemas y tareas en cuya definición entra el concepto de información. Sobre la base de este enfoque esta teoría se extiende al estudio de los procesos ligados al manejo de información. Con tal planteamiento, la teoría de información comprende los problemas de otras ciencias, en particular de la cibernética, biología, economía, psicología, lingüística y pedagogía (1).

La amplia utilización de las ideas de la teoría de información en distintas ramas de la ciencia está ligada al hecho de que la misma es fundamentalmente una teoría matemática. Sus conceptos principales (entropía, cantidad de información y poder de transmisión) solo se determinan a través de las probabilidades de acontecimientos, a los cuales se les puede atribuir diverso contenido físico (1).

La teoría de información constituye una herramienta matemática valiosa cuya utilidad sobrepasa su empleo en las comunicaciones, teniendo aplicación en diversas áreas de la ciencia. De forma general, el interés principal en la aplicación de la teoría de información en otras disciplinas, se ha enfocado en el análisis de patrones estadísticos de fuentes de información, en términos genéricos, siendo justificado este paradigma por el hecho de que principios comunes rigen el universo como una fuente de información (1).

1.1.4 Índices de información

Los índices de información (IF), como caso particular, son una familia de descriptores moleculares calculados como el contenido de información de átomos o moléculas. Esta amplia familia de descriptores moleculares, incluye índices muy sencillos como los que se derivan de la composición elemental de un átomo o molécula (índices de composición), hasta los más complejos, como por ejemplo; los índices de simetría de vecindad, los cuales no son ajenos a las limitaciones de los descriptores moleculares (1).

1.2 Características del API encargada de realizar los cálculos de los descriptores moleculares

En el API desarrollada se propone una nueva familia de descriptores moleculares basados en la aplicación de la teoría de información a la codificación de la estructura química, caracterizada por “fragmentos moleculares” generados en función de la aplicación de criterios, a los cuales denominamos sucesos. Estos sucesos se clasifican en tres grupos principales: grafo teóricos (sub-grafos conexos, caminos terminales,

incidencia de vértices en caminos, sub-grafos cuánticos, caminos de longitud k y sub-grafos de Sach), huellas (MACCs, estado-e, subestructuras) y magnitudes fisicoquímicas (hidrofobicidad y refractividad atómica) (1).

Los sucesos, como fuentes discretas de información, sirven de base para la aplicación de la teoría de codificación de una fuente empleando las ecuaciones de entropía de Shannon, negentropía y entropía de Shannon estandarizada y, por primera vez en química-matemática, la teoría de codificación de canales de transmisión fundamentada en los conceptos de la información mutua, entropía condicional y de unión, para un sistema de comunicación de dos fuentes (dupla). Luego se extiende este modelo de un sistema binario a dimensiones superiores, o sea, sistemas de tres y cuatro fuentes (terna y cuaterna). Además, se definen estrategias que generalizan la definición de invariantes globales y locales como combinación lineal de las contribuciones atómicas. En este sentido, se introducen las normas (distancias), medias, invariantes estadísticas y algoritmos clásicos, de las cuales los algoritmos clásicos demuestran el mejor desempeño, seguidos por las medias, normas e invariantes estadísticas (1).

Los índices de información implementados en el API desarrollado, se han comprobados en anteriores trabajos que son capaces de captar información no codificada por los índices de información de otras aplicaciones (1).

A continuación se hará una breve descripción de los pasos a seguir para el cálculo de los índices de información correspondientes:

Paso 1: Construir las matrices de frecuencias de relaciones para la fuente de información generada según el modelo de “fragmentos” químicos seleccionado.

Paso 2: Obtener las matrices de probabilidad cuyos elementos P_{ijkl}, P_{ijk} o P_{ij} se derivan de las matrices de relaciones cuaternas, ternas y dupla respectivamente.

Paso 3: Calcular la información mutua, entropía condicional y entropía de unión.

Paso 4: Cálculo de las medidas entrópicas atómicas (LOVIs[3]).

Paso 5: Aplicación de las invariantes al vector de LOVIs.

[3] Invariante local de vértices

La secuencia de pasos descrita anteriormente puede ser orientada a flujos de trabajo. Permitiendo que el usuario diseñe y organice gráficamente los elementos a tener en cuenta para la realización del proceso de cálculo de los descriptores moleculares basados en teoría de la información.

1.3 Interfaz gráfica de usuario (GUI)

La interfaz gráfica de usuario, conocida también como GUI (del inglés graphical user interface) es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en proporcionar un entorno visual sencillo para permitir la comunicación con el sistema operativo de una máquina o computador (4).

Habitualmente las acciones se realizan mediante manipulación directa, para facilitar la interacción del usuario con la computadora. Surge como evolución de las interfaces de línea de comandos que se usaban para operar los primeros sistemas operativos y es pieza fundamental en un entorno gráfico. Como ejemplos de interfaz gráfica de usuario, cabe citar los entornos de escritorio Windows, el X-Window de GNU/Linux o el de Mac OS X, Aqua (4).

En el contexto del proceso de interacción persona-ordenador, la interfaz gráfica de usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático.

1.3.1 Clasificación de las interfaces de usuarios

Las Interfaces de Usuario se distinguen básicamente en dos tipos (5):

- Una interfaz de hardware, a nivel de los dispositivos utilizados para ingresar, procesar y entregar los datos: teclado, ratón y pantalla visualizadora.

- Una interfaz de software, destinada a entregar información acerca de los procesos y herramientas de control, a través de lo que el usuario observa habitualmente en la pantalla (5).

De esta clasificación general se puede ir desprendiendo algunas, así por ejemplo según su evolución tenemos:

Interfaces de línea de mandatos (command-line user interfaces, CUIs)

Es la característica del DOS, el sistema operativo de disco en los primeros PC, y es el estilo más antiguo de interacción hombre-máquina. El usuario escribe órdenes utilizando un lenguaje formal con un vocabulario y una sintaxis propia (los mandatos en el caso del DOS). Se usa un teclado, típicamente, y las órdenes están encaminadas a realizar una acción (5).

Inconveniente: carga de memoria del usuario (debe memorizar los mandatos; incluso la ayuda es difícil de leer); nombres no siempre adecuados a las funciones, significado de los mandatos mal comprendido a veces (varios mandatos con el mismo o parecido significado, como DEL y ERASE); inflexible en los nombres (DEL y no DELETE) (5).

Ventajas: potente, flexible y controlado por el usuario, aunque esto es una ventaja para usuarios experimentados. La sintaxis es estricta, y los errores pueden ser graves.

Interfaces de menús

Un menú es una lista de opciones que se muestran en la pantalla o en una ventana de la pantalla para que los usuarios elijan la opción que deseen. Los menús permiten dos cosas: navegar dentro de un sistema, presentando rutas que llevan de un sitio a otro, y seleccionar elementos de una lista, que representan propiedades o acciones que los usuarios desean realizar sobre algún objeto (5).

Existen distintos tipos de menús:

- Menú de pantalla completa.
- Menú de barra y menú desplegable.
- Paletas o barras de herramientas.
- Menús Contextuales (Pop-Up).

En resumen, este tipo de interfaz de usuario, siempre y cuando se encuentren bien estructuradas, son bien recomendadas para usuarios noveles o principiantes. Son fáciles de aprender y de recordar. Estas pueden variar desde menús muy simples hasta menús muy avanzados.

Precauciones: No ocupar demasiado espacio de la pantalla, no colocar demasiados elementos en el menú, así como agrupar estos de manera lógica, permitir la personalización por parte del usuario, hacer uso de una terminología adecuada y consistente dentro de la aplicación, así como con otros programas (Exit, Quit, Escape, Clore, Return, Back). Estas interfaces son comúnmente utilizadas en conjunción con otros tipos de interfaces (5).

1.3.2 Características humanas del diseño de interfaz

Al diseñar interfaces de usuario deben tenerse en cuenta las habilidades cognitivas y de percepción de las personas, y adaptar el programa a ellas. Así, una de las cosas más importantes que una interfaz puede hacer es reducir la dependencia de las personas de su propia memoria, no forzándoles a recordar cosas innecesariamente (6).

Las personas tienen habilidades muy distintas a las de un ordenador, y ésta debe utilizar las suyas para soslayar las de aquellas, como por ejemplo la escasa capacidad de la memoria de corto alcance (6).

- Velocidad de Aprendizaje: La cual está referida a la velocidad con que el usuario aprende a utilizar el sistema. Es decir, pretende que el usuario minimice este tiempo de aprendizaje.
- Velocidad de Respuesta: Es el tiempo necesario en que se realiza una operación en el sistema.
- Tasa de Errores: Esta define el porcentaje de errores que comete el usuario.
- Retención: Es la capacidad de retención de información que presenta un usuario sobre un sistema en un periodo de tiempo.
- Satisfacción: Está referida a conocer en qué medida un usuario está satisfecho con un sistema dado.

Adecuación:

- Características Físicas: Cada persona tiene diferentes características físicas. Hay algunas personas que no les gustan los teclados mientras que a otras sí. Es por eso que hay teclados ergonómicos. Lo mismo sucede con el mouse.
- Ambiente: El lugar donde va a ser usado el sistema. Cada interfaz tiene que adecuarse al lugar.
- Visibilidad: Tomar en cuenta la cantidad de iluminación del lugar. ¿Se refleja el brillo en la pantalla?

- Personalidad: De acuerdo a la edad, nivel socio-económico, etc.
- Cultura: Los japoneses no tienen las mismas pantallas, ventanas, etc. Este factor es importante si el mercado para el sistema es a nivel internacional.

Motivación:

- Sistemas Vitales: Son de vida o muerte; muchas personas dependen de ellos. Ejemplo: un sistema para reactores nucleares. Este sistema trabaja en tiempo real, y es de suma importancia la seguridad y efectividad del mismo.
- Sistemas Comerciales e Industriales: Sirven para aumentar la productividad y vender más.
- Sistemas de Oficina, Hogar y Juegos: Factor importante: el mercado a quien está dirigido; tienen que ser muy amigables y satisfacer al cliente.
- Sistemas de Investigación: Realizan tareas muy específicas y tratan de imitar el medio en el que se desenvuelve el usuario.

1.3.3 Pasos para el desarrollo de un una interfaz

En el proceso de diseño de una interfaz de usuario se pueden distinguir cuatro fases o pasos fundamentales:

1. Reunir y analizar la información del usuario.
2. Diseñar la interfaz de usuario.
3. Construir la interfaz de usuario.
4. Validar la interfaz de usuario.

Reunir y analizar la información del usuario:

Esta fase se basa en concretar a través de técnicas de requerimientos, qué tipo de usuarios van a utilizar el programa, qué tareas van a realizar los usuarios y cómo las van a realizar, qué exigen los usuarios del programa, en qué entorno se desenvuelven los usuarios (físico, social, cultural) (5).

Diseñar la interfaz de usuario.

Es importante dedicar tiempo y recursos a esta fase, antes de entrar en la codificación. En esta fase se definen los objetivos de usabilidad del programa, las tareas del usuario, los objetos y acciones de la

interfaz, los iconos, vistas y representaciones visuales de los objetos, los menús de los objetos y ventanas. Todos los elementos visuales se pueden hacer primero a mano y luego refinar con las herramientas adecuadas (5).

Construir la interfaz de usuario.

Es interesante realizar un prototipo previo, una primera versión del programa que se realice rápidamente y permita visualizar el producto para poderlo probar antes de codificarlo definitivamente (5).

Validar la interfaz de usuario.

Se deben realizar pruebas funcionales al producto, y ver en qué medida se cumplieron las expectativas del cliente.

Existen 11 pasos en el proceso de diseño "centrado en las tareas", similar al anterior pero que desglosa algunas actividades implícitas en otras, así (5):

1.- Entender quién usará el sistema para hacer qué.
2.- Elegir tareas representativas para el diseño.
3.- Plagiar o copiar.
4.- Bosquejar un diseño.
5.- Pensar acerca del diseño.
6.- Crear un prototipo.
7.- Evaluarla con los usuarios.
8.- Repetir.
9.- Construirla.
10.- Rastrearla.
11.- Cambiarla

1.4 Análisis de la interfaz gráfica de usuario de algunas herramientas

A partir de lo expuesto en el epígrafe 1.2, se analizarán a continuación algunas herramientas que servirán de base para el desarrollo de nuestra aplicación, las cuales permiten diseñar de forma gráfica la secuencia

de pasos pertinentes al proceso que realiza cada una.

1.4.1 Ketlle (Pentaho Data Integration).

Es un proyecto belga que incluye un conjunto de herramientas para realizar ETL[4]. Uno de sus objetivos es que el proyecto ETL sea fácil de generar, mantener y desplegar.
Se compone de 4 herramientas:

Spoon: permite diseñar de forma gráfica la transformación ETL.

Pan: ejecuta las transformaciones diseñadas con Spoon.

Chef: permite, mediante una interfaz gráfica, diseñar la carga de datos incluyendo un control de estado de los trabajos.

Kitchen: permite ejecutar los trabajos diseñados con Chef.

La interfaz de usuario del spoon es muy sencilla, disponiendo de dos perspectivas. Una de visualización (view), donde vemos los componentes que forman las transformación, y otra de diseño (Design), donde vemos los pasos disponibles. Según estemos trabajando con transformaciones o con trabajos, los pasos disponibles irán cambiando. En la figura: 1, se pueden ver la perspectiva Diseño. A la izquierda tenemos los diferentes pasos que iremos arrastrando a la sección de la derecha donde se encuentra el área de diseño. Los pasos tanto de transformaciones como de trabajos se irán enlazando con los saltos correspondientes (7).

[4] Extracción, transformación y carga de datos.

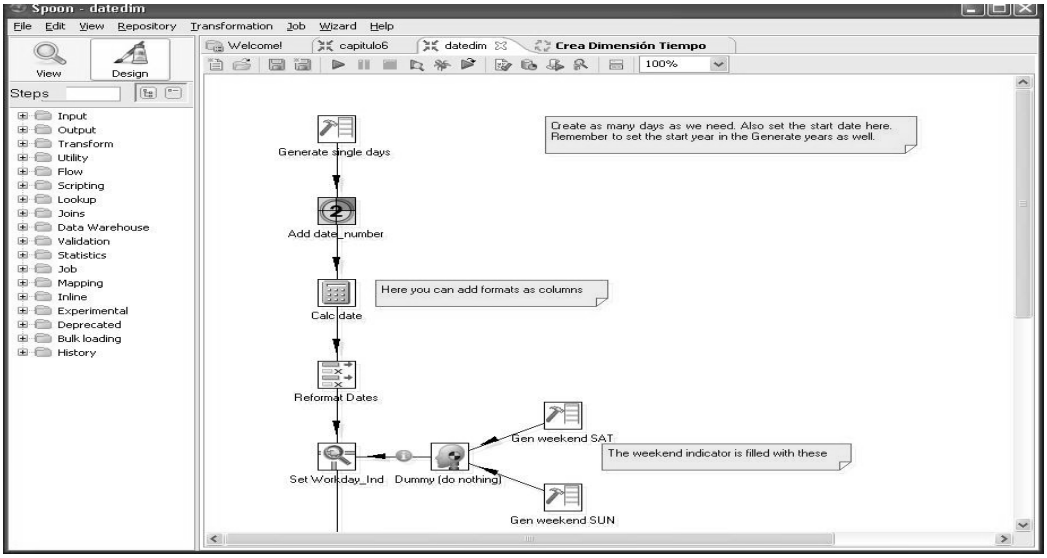

Figura: 1 Interfaz gráfica del Ketlle

En la perspectiva Vista, se visualizan en forma de árbol los componentes utilizados en el diseño de la transformación o trabajo y haciendo doble click en cada uno de los componentes en la parte de diseño podemos acceder a la configuración de sus propiedades.

1.4.2 Weka

Es una herramienta que se usa en la minería de datos la cual presenta un conjunto librerías que pueden ser llamadas desde su misma interfaz o incluso desde propias clases java.

Al ejecutar la aplicación nos aparece el selector de interfaz de WEKA que da la opción de seleccionar entre cuatro posibles interfaces de usuario para acceder a las funcionalidades del programa, éstas son Simple CLI, Explorer, Experimenter y Knowledge Flow.

Knowledge Flow

Es una interfaz que soporta esencialmente las mismas funciones que el Explorer pero con una interfaz que permite arrastrar y soltar. Una ventaja es que ofrece soporte para el aprendizaje incremental.

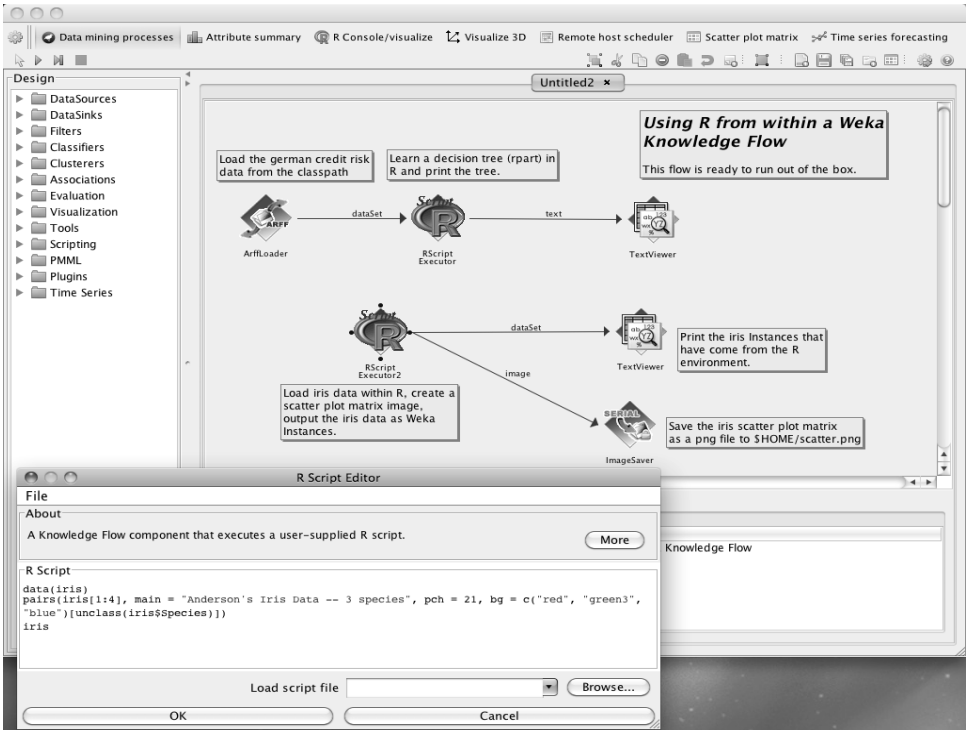

Figura: 2 Interfaz gráfica Del Knowledge Flow del Weka

El KnowledgeFlow presenta una interfaz inspirada en el flujo de datos para WEKA. El usuario puede seleccionar componentes WEKA desde una barra de herramientas, colocarlos en un área diseño y conectarlos entre sí para formar un flujo de conocimiento para la transformación y análisis de datos. En la actualidad, todos los clasificadores de WEKA, filtros, clústeres, cargadores y protectores están disponibles en el KnowledgeFlow junto con algunas herramientas extras (8).

El KnowledgeFlow ofrece las siguientes características:

• Diseño en estilo de flujo.

• Los datos de proceso en lotes o por incrementos.

•Procesa múltiples hilos en paralelo (cada flujo se ejecuta por separado en su propio hilo).

• Filtros de juntas de la cadena.

• Visualiza el rendimiento de los clasificadores incrementales durante el procesamiento (desplazamiento parcelas de precisión de la clasificación, el error RMS, predicciones, etc.)

• Instalación de plugin para permitir la fácil adición de nuevos componentes para el flujo de conocimientos.

Resultado del análisis

La interfaz de estas herramientas es de gran utilidad para los usuarios. Presentan un área donde se pueden visualizar los componentes que interactúan en sus respectivos procesos, evitando que el usuario haga uso de su memoria, no forzándoles a recordar eventos innecesarios. Además favorecen la comprensión del proceso al mostrarlo como un dibujo.

1.5 Metodología, Herramientas y Tecnologías

La solución propuesta requiere la selección de una metodología de desarrollo, que permita elaborar un marco de trabajo para la estructuración, planeación y control del proceso de desarrollo. Se deben seleccionar además las tecnologías y herramientas a utilizar para su construcción, teniendo en cuenta las características que debe tener el sistema y las restricciones de uso de las herramientas.

1.5.1 Metodologías de desarrollo

Desarrollar un buen software depende de un sin número de actividades y etapas, donde el impacto de elegir la mejor metodología para un equipo, en un determinado proyecto es trascendental para el éxito del producto. El papel preponderante de las metodologías es sin duda esencial en un proyecto y en el paso inicial, que debe encajar en el equipo, guiar y organizar actividades que conlleven a las metas trazadas.

Existen las metodologías tradicionales y metodologías ágiles, las primeras están pensadas para el uso exhaustivo de documentación durante todo el ciclo del proyecto mientras que las segundas ponen vital importancia en la capacidad de respuesta a los cambios, la confianza en las habilidades del equipo y al mantener una buena relación con el cliente (9).

Rational Unified Process (RUP) es uno de los métodos que más se usan de las metodologías tradicionales, provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Fue desarrollado por Rational Software, y está integrado con toda la suite Rational de herramientas. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. Es guiado por casos de uso y centrado en la arquitectura, y utiliza UML como lenguaje de notación (9).

Fases que la componen

Las cuatro fases del ciclo de vida son:

Concepción: Durante la fase de inicio se define el modelo del negocio y el alcance del proyecto. Se identifican todos los actores y Casos de Uso, y se diseñan los Casos de Uso más esenciales (aproximadamente el 20% del modelo completo). Se desarrolla, un plan de negocio para determinar que recursos deben ser asignados al proyecto.

Elaboración: El propósito de la fase de elaboración es analizar el dominio del problema, establecer los cimientos de la arquitectura, desarrollar el plan del proyecto y eliminar los mayores riesgos.

Construcción: La finalidad principal de esta fase es alcanzar la capacidad operacional del producto de forma incremental a través de las sucesivas iteraciones. Durante esta fase todos los componentes, características y requisitos deben ser implementados, integrados y probados en su totalidad, obteniendo una versión aceptable del producto.

Transición : La finalidad de la fase de transición es poner el producto en manos de los usuarios finales, para lo que se requiere desarrollar nuevas versiones actualizadas del producto, completar la documentación, entrenar al usuario en el manejo del producto, y en general tareas relacionadas con el ajuste, configuración, instalación y facilidad de uso del producto.

Ventajas

1. Evaluación en cada fase que permite cambios de objetivos
2. Funciona bien en proyectos de innovación.
3. Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
4. Seguimiento detallado en cada una de las fases.

Desventajas

1. La evaluación de riesgos es compleja
2. Excesiva flexibilidad para algunos proyectos
3. Estamos poniendo a nuestro cliente en una situación que puede ser muy incómoda para él.
4. Nuestro cliente deberá ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto con él.

Programing Extreme(XP) es un proceso ágil desarrollado por Kent Beck. La programación extrema se diferencia de las metodologías tradicionales principalmente en que pone más énfasis en la adaptabilidad que en la previsibilidad.
Los defensores de XP consideran que los cambios de requisitos sobre la marcha son un aspecto natural, inevitable e incluso deseable del desarrollo de proyectos (9).

Las características fundamentales del método son:

- Desarrollo iterativo e incremental: pequeñas mejoras, unas tras otras.
- Pruebas unitarias continuas, frecuentemente repetidas y automatizadas, incluyendo pruebas de regresión. Se aconseja escribir el código de la prueba antes de la codificación.
- Programación por parejas: se recomienda que las tareas de desarrollo se lleven a cabo por dos personas en un mismo puesto. Se supone que la mayor calidad del código escrito de esta manera el código es revisado y discutido mientras se escribe es más importante que la posible pérdida de productividad inmediata.
- Frecuente interacción del equipo de programación con el cliente o usuario. Se recomienda que un representante del cliente trabaje junto al equipo de desarrollo.
- Corrección de todos los errores antes de añadir nueva funcionalidad. Hacer entregas frecuentes.

- Refactorización del código, es decir, reescribir ciertas partes del código para aumentar su legibilidad y mantenibilidad pero sin modificar su comportamiento. Las pruebas han de garantizar que en la refactorización no se ha introducido ningún fallo.
- Propiedad del código compartida: en vez de dividir la responsabilidad en el desarrollo de cada módulo en grupos de trabajo distintos, este método promueve el que todo el personal pueda corregir y extender cualquier parte del proyecto. Las frecuentes pruebas de regresión garantizan que los posibles errores serán detectados

Ventajas

1. Apropiado para entornos volátiles.
2. Estar preparados para el cambio, significa reducir su coste.
3. Planificación más transparente para nuestros clientes, conocen las fechas de entrega de funcionalidades. Vital para su negocio
4. Permitirá definir en cada iteración cuales son los objetivos de la siguiente

5. Permite tener realimentación de los usuarios muy útil.

Desventajas

1. Delimitar el alcance del proyecto con nuestro cliente.

Para mitigar esta desventaja se plantea definir un alcance a alto nivel basado en la experiencia.

OpenUP es un proceso ágil con presencia de un desarrollo iterativo del software que es mínimo, completo y extensible. El proceso es mínimo porque solamente el contenido fundamental es incluido; es completo puesto que puede ser manifestado como todo el proceso para construir un sistema; extensible a causa de que puede ser utilizado como fundamento sobre el cual el contenido de proceso se pueda agregar o adaptar según lo necesitado. Como metodología de desarrollo es conducida por el principio de colaboración para alinear intereses y para compartir su comprensión.

OpenUP es un proceso de desarrollo unificado que está basado en Rational Unified Process (RUP) preservando la esencia de este y manteniendo las mismas características: desarrollo iterativo e incremental, dirigido por casos de uso y centrado en la arquitectura (10).

Además OpenUP utiliza el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés) para el modelado de sistemas de software el cual es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. OpenUP representa un número de modelos de desarrollo basados en componentes que han sido propuestos en la industria. Haciendo uso de UML define los componentes que se utilizarán para construir el sistema y las interfaces que conectarán los componentes.

1.4.1 Lenguaje de Modelado: UML 2.1

El lenguaje unificado de modelado (UML) es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema. UML ofrece un estándar para describir un plano del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocio, funciones del sistema y aspectos concretos como expresiones de lenguajes de programación y esquemas de bases de datos. Su principal objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo (11).

1.4.2 Herramienta de modelado Visual Paradigm 8.0

Existen varias herramientas que utilizan UML como lenguaje de modelado para generar los artefactos correspondientes a las diferentes etapas de desarrollo. Una de ellas es Visual Paradigm, la cual permite la construcción de aplicaciones de calidad a través de un proceso de desarrollo bien documentado.

Visual Paradigmas es una herramienta de modelado, fácil de usar, con soporte multiplataforma y que proporciona excelentes facilidades de interoperabilidad con otras aplicaciones. Está desarrollada para cubrir todo el ciclo del desarrollo de software, permitiendo la captura de requisitos, análisis, diseño e implementación. Tiene la capacidad de crear el esquema de clases a partir de una base de datos y crear la definición de base de datos a partir del esquema de las clases (12).

Permite invertir código fuente de programas, archivos ejecutables y binarios en modelos UML al instante, creando de manera simple toda la documentación. Está diseñada para usuarios interesados en sistemas de software con el uso del acercamiento orientado a objeto, además apoya los estándares más recientes de las notaciones de Java y de UML. Incorpora el soporte para trabajo en equipo, que permite que varios desarrolladores trabajen a la vez en el mismo diagrama y vean en tiempo real los cambios hechos por sus compañeros.

1.4.3 Lenguaje de programación.

Un lenguaje de programación es un lenguaje diseñado para describir el conjunto de acciones consecutivas que un equipo debe ejecutar. Por lo tanto, un lenguaje de programación es un modo práctico para que los seres humanos puedan dar instrucciones a un equipo (14).

En nuestros días existen una gran variedad de lenguajes de programación que se hacen unos más dependientes que otros en dependencia de lo que se quiera realizar. Teniendo en cuenta que el API está realizada en el lenguaje de programación Java, se analiza a continuación las principales características de este lenguaje.

Java es un lenguaje de programación, que fue diseñado por la compañía Sun Microsystems Inc, con el propósito de lograr su funcionamiento en redes computacionales heterogéneas (redes de computadora formadas por más de un tipo de computadora ya sean PC, MAC's, estaciones de trabajo, etc.) y ser

independiente de la plataforma en que se ejecute. Por tanto, un programa de Java puede ejecutarse en cualquier máquina o plataforma (17).

Entre las características más sobresalientes de este lenguaje, se puede citar que:

- Es un lenguaje bastante simple, su diseño se basa en el paradigma de programación orientada a objetos.
- Resulta bastante familiar para los programadores que estén acostumbrados a lenguajes como C++ o C#.
- Se encarga automáticamente del manejo de la memoria.
- Fue diseñado con ciertas restricciones, sobre lo que se puede hacer y no sobre los recursos críticos de la computadora, aspecto que evita la posibilidad de codificar virus sobre él.
- Un programa escrito sobre Java es portable, ya que puede ser utilizado por cualquier computadora que tenga implementado su interprete.
- Puede ejecutar diferentes líneas de código al mismo tiempo.
- Es un lenguaje interpretado, por tanto, puede ejecutarse sobre una máquina virtual.
- Es multiplataforma.

Este programa presenta algunas ventajas como:

- No es necesario volver a escribir el código cuando se desea ejecutar en otra máquina, basta con tener en ella la máquina virtual de Java instalada.
- Como es orientado a objetos, garantiza una mayor reutilización de código y demás ventajas que aporta este paradigma de programación.
- Mediante el uso de Java, se pueden realizar cálculos matemáticos, procesadores de palabras, bases de datos, aplicaciones gráficas, animaciones, sonido y hojas de cálculo.
- Permite desarrollar páginas web dinámicas, interactivas, a las que se les pueden incorporar animaciones y toda clase de elementos de multimedia, sin necesidad de invertir en la compra de paquetes de multimedia, que tienen un alto costo.
- El JDK es una herramienta libre de licencias (sin costo).
- Cada 6 meses Sun Microsystems Inc, pone en el mercado una nueva versión del JDK.
- Es independiente de la plataforma de desarrollo.

- Tiene las bibliotecas de clases graficas: AWT y SWING, que permiten realizar un diseño más refinado y funcional de las interfaces comunes.

1.4.4 Entornos de desarrollo integrado(del inglés IDE)

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, o sea, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Los IDEs pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes (13).Dentro de los IDEs libre de licencias más usados para programar, se encuentran el Netbeans y el Eclipse.

Netbeans es un proyecto de código abierto, fundado por la empresa Sun MicroSystems. Netbeans IDE es un entorno de desarrollo, es decir, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. Es un producto libre y gratuito, sin restricciones de uso (18).

Este entorno de desarrollo ofrece varias ventajas a los programadores tanto de Java como de otros lenguajes, ya que es muy fácil de usar, posee un editor de texto que permite el completamiento de código, aspecto importante tanto para principiantes como para expertos en el lenguaje, es muy factible para el desarrollo de aplicaciones desktop, pues a diferencia del Eclipse, no es necesario instalarle plug-ins como el Visual Editor, por lo que ofrece muchas ventajas para el diseño visual (librería Swing y awt) (18).

Eclipse es originalmente creado por la IBM en noviembre de 2001, como sucesor de la familia de herramientas VisualAge. Pertenece a una comunidad de código abierto, sin ánimos de lucro, que se centra en el desarrollo de una plataforma formada por herramientas para crear y gestionar software en todo el ciclo de vida.

Es un entorno de desarrollo integrado, de código abierto, multiplataforma, que tiene como objetivo desarrollar "Aplicaciones de Cliente Enriquecido", lo que es opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma típicamente ha sido usada para desarrollar entornos de desarrollo integrados, como el IDE de Java, llamado Java Development Toolkit (JDT) y el compilador (ECJ), que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse) (19).

1.4.5 La Plataforma Netbeans

La plataforma Netbeans hace fuerte hincapié en la construcción del software de forma modular, módulo sobre módulo y así sacarle un mejor provecho ya que brinda implementados los mecanismos de descubrimiento de nuevos módulos (y de actualizaciones de los existentes) desde sitios remotos, resolución de dependencias, activación/desactivación de módulos en caliente, comunicación entre ellos, permitiendo así preocuparse por la lógica y rápidamente desplegar aplicaciones, pudiendo ir extendiendo su funcionalidad a medida que pasa el tiempo.

Se pueden crear aplicaciones conformadas por varios módulos diferentes, cada uno responsable de llevar a cabo determinadas funcionalidades, según el rol de la persona que vaya a utilizarlo. Solo se cargan en la aplicación los módulos para cumplir su tarea, permitiendo tener un abanico de aplicaciones sin tener que tirar una línea de código innecesaria (20).

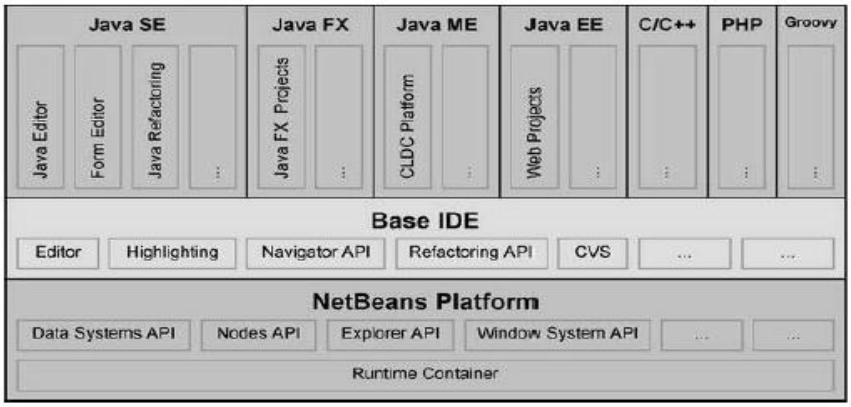

Figura: 3 Arquitectura de la Plataforma Netbeans

La plataforma ofrece servicios comunes a las aplicaciones de escritorio, permitiéndole al desarrollador enfocarse en la lógica específica de su aplicación. Entre las características de la plataforma están:

- Administración de las interfaces de usuario (ej. menús y barras de herramientas)
- Administración de las configuraciones del usuario Administración del almacenamiento (guardando y cargando cualquier tipo de dato)

- La plataforma Netbeans permite que las aplicaciones sean desarrolladas a partir de un conjunto de componentes llamados módulos.

Módulo es un archivo Java que contiene clases de java escritas para interactuar con las APIs de Netbeans y un archivo especial (manifest file) que lo identifica como módulo. Las aplicaciones construidas a partir de módulos pueden ser extendidas agregándole nuevos módulos. Debido a que los módulos pueden ser desarrollados independientemente, las aplicaciones basadas en la plataforma Netbeans pueden ser extendidas fácilmente por otros desarrolladores de software (20).

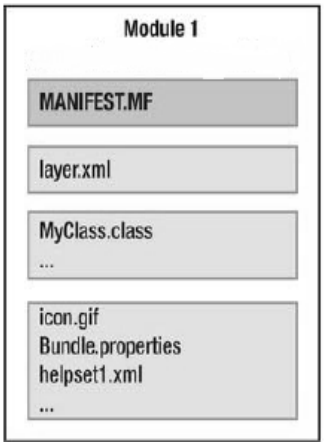

Figura: 4 Estructura de un módulo

Manifest file (manifest.mf): Cada módulo se ejecuta en la plataforma Netbeans tiene un archivo de manifiesto. Este archivo es una descripción textual del módulo y su entorno. Cuando se carga un módulo, el archivo de manifiesto es el primer archivo que lee el sistema de módulo. Un módulo de Netbeans se reconoce si el archivo de manifiesto contiene la OpenIDE-Módulo. Este es el único atributo obligatorio.

Layer file (*layer.xml*): Este es el archivo de configuración central, en el que prácticamente se define todo lo que un módulo añade a la Plataforma. Así, el archivo de capa es la interfaz entre el módulo y la plataforma Netbeans, mediante declaración que describe la integración de un módulo en el Plataforma.

MyClass.class: Estas son las clases que se van a crear e implementar en dependencia de lo que se quiera realizar.

Un módulo puede ser usado en cualquier proyecto desarrollado sobre Netbeans Plataform (siempre que se cumplan las restricciones de dependencias del mismo), incluso sobre Netbeans IDE.

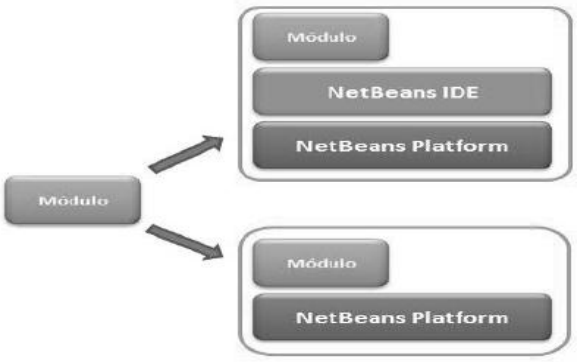

Figura: 5 Módulo para el IDE Netbeans o para otra aplicación desarrollada en Netbeans

Netbeans IDE es buen ejemplo de una aplicación de cliente enriquecido modular. La funcionalidad y características de una IDE, como su apoyo lenguaje Java o el editor de código, se crean en forma de módulos en la parte superior de la plataforma Netbeans, como se muestra en la Figura: 5. Esto ofrece una gran ventaja porque la aplicación se puede ampliar mediante módulos adicionales y adaptado a las necesidades específicas de los usuarios, lo que permite módulos específicos que no están acostumbrados a ser desactivado o desinstalado.

1.5 Conclusiones

El estudio de los principales conceptos relacionados con las interfaces gráficas de usuario permitió sentar las bases para el desarrollo de la interfaz gráfica de usuario que facilitará el proceso de cálculo de los descriptores moleculares basados en teoría de la información. Además se analizó la interfaz gráfica de algunas herramientas de diseño y ejecución de flujo de trabajo las cuales ofrecerán una idea para el desarrollo de la interfaz gráfica del presente trabajo, teniendo en cuenta la secuencia de pasos que se realizan en el proceso de cálculo de los descriptores moleculares.

Se decide realizar una aplicación basada en la plataforma del Netbeans aprovechando la posibilidad de la construcción de aplicaciones informáticas mediante una arquitectura basada en componentes y las funcionalidades que brinda de extensibilidad y adaptabilidad para la instalación de módulos y comunicación entre ellos.

Capítulo 1: Fundamentos teóricos

El proceso de desarrollo del software se guiará por la metodología OpenUP por ser una metodología adaptable al contexto y a las necesidades, que permite tener bien claro y accesible el proceso de desarrollo que se quiere alcanzar. Además se hará uso del lenguaje de modelado a UML y el Visual Paradigm 8.0 para construir los diagramas correspondientes con la misma.

Para el desarrollo de la aplicación se utilizará el lenguaje de programación java con su JDK en su versión 1.8 teniendo en cuenta que el API está realizada en ese mismo lenguaje y como herramienta de desarrollo el IDE Netbeans en su versión 7.4 partiendo de que este cuenta con más facilidades para el diseño de una interfaz visual atractiva a los usuarios, mejor elaborada y más funcional, con el propósito de lograr que interactúen con la aplicación de manera ágil y sencilla.

Capítulo 2: Análisis y diseño de la aplicación

Introducción

El presente capítulo muestra el modelo de dominio, así como la especificación de los requisitos funcionales y no funcionales de la aplicación, se presenta el diagrama de casos de usos del sistema para identificar la relación entre los actores y casos de usos. Además se brinda una descripción de la arquitectura del sistema, los patrones de diseños más utilizados y la aplicación de los mismos para la confección de los diagramas del modelo del diseño.

2.1 Modelo de dominio

El modelo de dominio es un artefacto de la fase de elaboración, se representa en UML con un diagrama de clases en el que se pueden mostrar conceptos u objetos del dominio del problema, asociaciones entre las clases conceptuales o atributos de las clases conceptuales. No contiene conceptos propios de un sistema de software sino de la propia realidad física (32).

Los modelos de dominio pueden utilizarse para capturar y expresar el entendimiento ganado en un área bajo análisis como paso previo al diseño de un sistema, ya sea de software o de otro tipo. Similares a los mapas mentales utilizados en el aprendizaje, el modelo de dominio es utilizado por el analista como un medio para comprender el sector industrial o de negocios al cual el sistema va a servir (32).

Capítulo 2: Análisis y diseño de la aplicación.

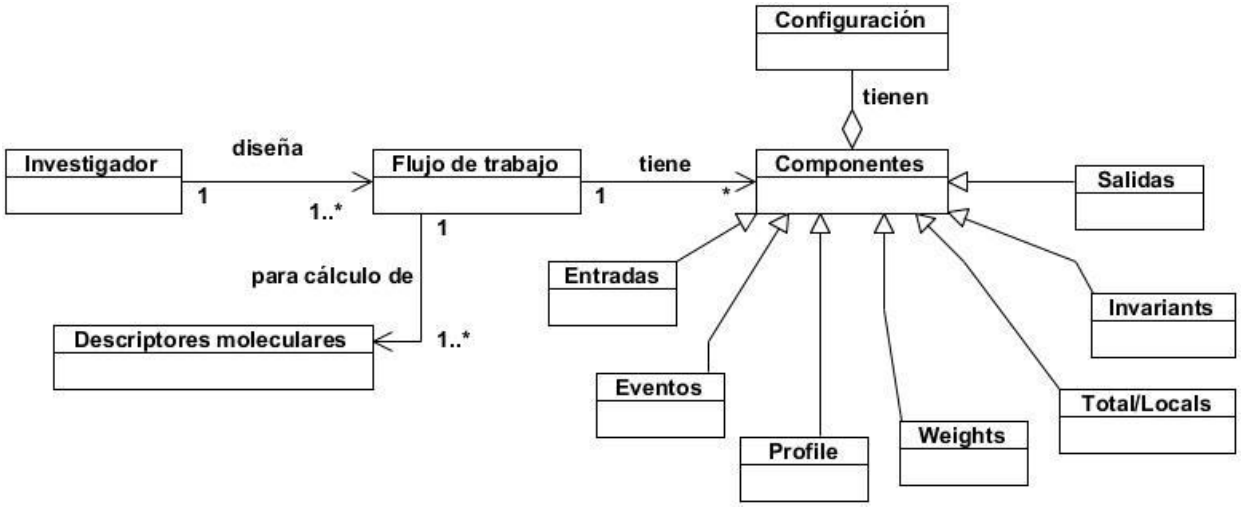

Figura: 6 Diagrama de clase del dominio de la aplicación

En este modelo se evidencia los elementos asociados con la aplicación. El investigador diseña uno o varios **flujo de trabajo** para los cálculos de los **descriptores moleculares**, este flujo de trabajo va a contener varios **componentes** que pueden tener o no **configuración** y estos componentes pueden ser de varios tipos.

Tabla 1 Descripción de las clases del modelo de dominio

Investigador	Persona encargada de realizar los cálculos de los descriptores moleculares.
Flujo de trabajo	Diseño gráfico de la secuencia de pasos para el cálculo de los descriptores moleculares.
Descriptores Moleculares	Información química que contiene una molécula dada por un número.
Componentes	Componentes necesarios para realizar el flujo de trabajo.
Configuración	Configuración de los componentes para el cálculo de los descriptores moleculares.

Entrada	Entrada de las moléculas para el cálculo de los descriptores moleculares
Eventos	Fragmentos moleculares generados a partir de la configuración definida por el investigador
Profile	Aplicación de la teoría de codificación de una fuente empleando ecuaciones
Weights	Algoritmos aplicados en el proceso de cálculo
Total/Locals	Combinación lineal de las contribuciones atómicas
Invariants	Algoritmos aplicados en el proceso de cálculo
Salidas	Resultado del cálculo de los descriptores moleculares

2.2 Requisitos del sistema

Los requisitos de un sistema son los servicios proporcionados por el sistema y sus restricciones operativas. Estos requisitos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema como el control de un dispositivo, hacer un pedido o encontrar información. El proceso de descubrir, analizar, documentar y verificar estos servicios y restricciones se denomina ingeniería de requisitos. Los requisitos del sistema del software se clasifican en funcionales y no funcionales (21).

1.2.1 *Requisitos Funcionales*

Este tipo de requisitos son declaraciones de los servicios que debe proporcionar el sistema, de la manera en que este debe reaccionar a entradas particulares y de cómo se debe de comportar en situaciones particulares (21). En algunos casos, los requisitos funcionales de los sistemas también pueden declarar explícitamente lo que el sistema no debe hacer. A continuación se describen los requisitos funcionales identificados para el módulo a implementar:

RF_1 Adicionar componente al área de trabajo.

RF_2 Eliminar componente del área de trabajo.

RF_3 Modificar las dimensiones del componente en el área de trabajo.

RF_4 Adicionar relación entre componentes en el área de trabajo.

RF_5 Eliminar relación entre componentes en el área de trabajo.

RF_6 Registrar configuración del componente en el área de trabajo.

RF_7 Modificar configuración del componente en el área de trabajo.

RF_8 Eliminar configuración del componente en área de trabajo.

RF_9 Adicionar nueva área de trabajo.

RF_10 Ejecutar flujo de trabajo diseñado.

RF_11 Parar la ejecución del flujo de trabajo diseñado

RF_12 Exportar el flujo de trabajo diseñado como proyecto en un directorio local.

RF_13 Importar el flujo de trabajo diseñado como proyecto desde un directorio local.

1.2.2 *Requisitos no funcionales*

Estos requisitos como su nombre lo sugiere, son aquellos requisitos que no se refieren directamente a las funciones específicas que proporciona el sistema, sino a las propiedades emergentes de éste como la fiabilidad, el tiempo de respuesta y la capacidad de almacenamiento. De manera general estos tipos de requisitos se encuentran estrechamente vinculados a los funcionales, atendiendo a que una vez que se conoce lo que el sistema debe hacer, es preciso determinar las cualidades que debe cumplir y cuán rápido debe ser (21). Los requisitos no funcionales que debe cumplir el sistema para garantizar su correcto funcionamiento son los siguientes:

RnF1. Usabilidad

Capítulo 2: Análisis y diseño de la aplicación.

La aplicación informática debe garantizar un acceso fácil y rápido, contando con un menú que satisfaga las necesidades de los usuarios. La herramienta podrá ser usada sólo por usuarios que posean conocimientos en el dominio de la especialidad.

RnF1. Interfaz

1. La interfaz debe tener colores y tipo de letra similar a los del Netbeans.

2. Los mensajes (Error o Información) deberán estar en idioma inglés.

RnF2. Requisitos de licencia

3. Las herramientas y las tecnologías en que estará basada la aplicación informática deben cumplir con las licencias de software libre.

4. Al finalizar el desarrollo de la Aplicación Informática y se haya identificado el proyecto que pueda utilizar la solución, la dirección del grupo definirá la licencia por la que se regirá el mismo.

RnF3. Requisitos de software

5. Se requiere para el funcionamiento de la aplicación disponer de Windows o Linux como sistema operativo y Máquina Virtual de Java versión 1.7 o superior.

RnF4. Requisitos de hardware

6. Para la ejecución de la aplicación se necesitará:

- 2 GB de memoria RAM como mínimo.
- 20 GB de capacidad en el disco duro.
- Procesador de más de dos núcleos.

Capítulo 2: Análisis y diseño de la aplicación.

2.3 Casos de uso del sistema (CUS)

Un caso de uso es una técnica para la captura de requisitos potenciales de un nuevo sistema o una actualización de software. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para conseguir un objetivo específico. Normalmente, en los casos de usos se evita el empleo de jergas técnicas, prefiriendo en su lugar un lenguaje más cercano al usuario final. En ocasiones, se utiliza a usuarios sin experiencia junto a los analistas para el desarrollo de casos de uso (19). A continuación se mencionan los casos de uso en que fueron agrupados los requisitos funcionales identificados para el desarrollo de la aplicación:

CU1_Gestionar diseño del componente.

RF_1 ,**RF_2**, **RF_3**, **RF_4**, **RF_5**

CU2_Gestionar configuración del componente.

RF_6, RF_7, **RF_8**

CU3_Administrar área de trabajo.

RF_9

CU3_Gestinar flujo de trabajo diseñado.

RF_10, **RF_11**, **RF_12**, **RF_13**

2.4 Diagrama de casos de uso del sistema

Los diagramas de casos de uso son el punto de partida en todo proyecto de desarrollo de software basado en UML y constituyen una representación gráfica de los procesos y su interacción con los actores, se utilizan para ilustrar los requisitos del sistema al mostrar cómo reacciona una respuesta a eventos que se producen en el mismo (22).

En la siguiente figura se muestra el diagrama de casos de uso del sistema propuesto para el desarrollo del sistema. En el que se incluyen todos los casos de uso necesarios para satisfacer los requisitos funcionales especificados.

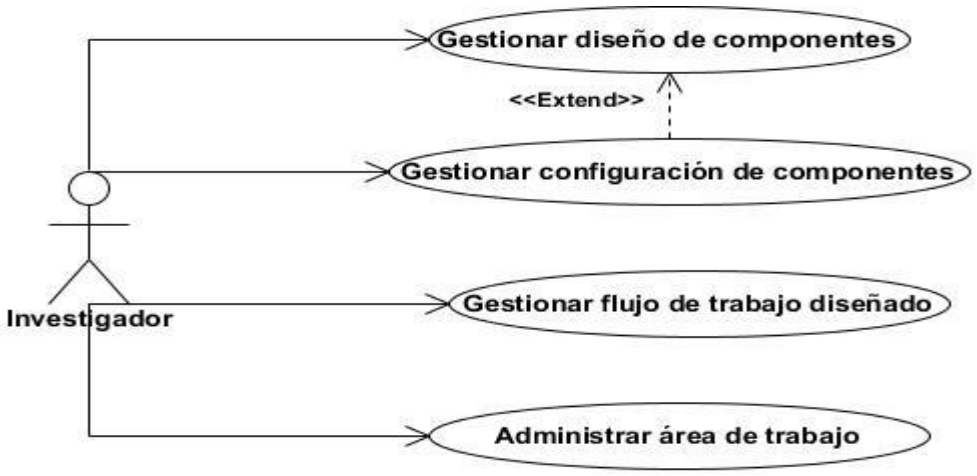

Figura: 7Diagrama de casos de uso del sistema.

2.5 Descripción de los casos de uso del sistema

Una vez que se identifiquen los casos de usos del sistema se realiza una detallada descripción de estos para satisfacer los requisitos funcionales. A continuación se muestra la descripción de un caso de uso crítico para el funcionamiento de la aplicación.

Tabla 2 Descripción del caso de uso Gestionar diseño de componentes del área de trabajo

Nombre	Gestionar diseño de componentes en el área de trabajo.
Objetivo	El Investigador realiza el diseño del flujo de trabajo utilizando los componentes definidos, puede adicionarlos, eliminarlos, modificar sus dimensiones, relacionarlos o quitar la relación entre los mismos.
Actores	**Investigador**: Adiciona y elimina componentes en el área de trabajo, modifica sus dimensiones y establece o elimina la relación entre ellos. Crea nueva área de trabajo.
Resumen	El investigador adiciona componentes al área de trabajo. Los componentes previamente adicionados se pueden eliminar del área de trabajo, modificar sus dimensiones (aumentarlo

	o disminuirlo), relacionar o quitar su relación con otros componentes.
Complejidad	Alta
Prioridad	Alta
Precondiciones	-
Postcondiciones	- Se adicionó un componente al área de trabajo. - Se eliminó un componente del área de trabajo. - Se modificaron las dimensiones de un componente en el área del trabajo. - Se adicionó una relación entre componentes en el área de trabajo. - Se eliminó una relación entre componentes en el área de trabajo. - Se creó una nueva área de trabajo.
Requisitos funcionales asociados	RF_1, RF_2, RF_3, RF_4, RF_5, RF_6

Flujo de eventos		
Flujo básico <Gestionar diseño de componente>		
	Actor	**Sistema**
1.	Realiza una de las siguientes acciones:	

	- Adicionar un componente al área de trabajo. - Eliminar un componente al área de trabajo. - Modificar las dimensiones del componente. - Adicionar relación entre componentes. - Eliminar relación entre componentes.	
2.		Permite realizar la acción indicada previamente: **7.** Adicionar un componente al área de trabajo, Ver Sección 1: Adicionar componente. **8.** Eliminar un componente al área de trabajo, Ver Sección 2: Eliminar componente. **9.** Modificar las dimensiones del componente, Ver Sección 3: Modificar dimensiones. **10.** Adicionar relación entre componentes, Ver Sección 4: Adicionar relación. **11.** Eliminar relación entre componentes. Ver Sección 5: Eliminar relación. **12.** Crear nueva área de trabajo. Ver Sección 6: Crear nueva área de

		trabajo.
3.		Termina el caso de uso.
Sección 1: "Adicionar componente"		
Flujo básico < Adicionar componente >		
	Actor	**Sistema**
1	Selecciona el tipo de componente (Entradas, Eventos, Profile, Weights, Total/Locals, Invariants, Salidas) que desea adicionar al área de trabajo.	
2	Indica el lugar del área del trabajo donde quiere colocar el componente seleccionado.	
3		Coloca el componente en la posición indicada en el área de trabajo.
4		Ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos.
Sección 2: "Eliminar componente"		
Flujo básico < Eliminar componente >		
	Actor	**Sistema**

1.	Selecciona el componente que desea eliminar del área de trabajo.	
2.	Indica eliminar el componente seleccionado.	
3.		Verifica si el componente seleccionado tiene relación con otros componentes del área de trabajo.
4.		Ejecuta la acción 2 del Flujo básico < Eliminar relación >, de la Sección 5: “Eliminar relación”.
5.		Verifica si el componente tiene configuración definida.
6.		Ejecuta la Sección 4: “Eliminar configuración del componente”, del Caso de Uso “Gestionar configuración de componentes en el área de trabajo”.
7.		Elimina el componente seleccionado del área de trabajo.
8.		Ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos.
Flujos alternos		

Nº Evento <3.a Componente sin relación>		
	Actor	**Sistema**
1.		Ejecuta la acción 5 del Flujo básico < Eliminar componente >, de la Sección 2: "Eliminar componente".
Nº Evento <5.a Componente sin configuración>		
	Actor	**Sistema**
1.		Ejecuta la acción 7 del Flujo básico < Eliminar componente >, de la Sección 2: "Eliminar componente".
Sección 3: "Modificar dimensiones"		
Flujo básico < Modificar dimensiones >		
	Actor	**Sistema**
1.	Selecciona el componente que desea modificar sus dimensiones en el área de trabajo.	
2.	Indica modificar las dimensiones del componente seleccionado.	

3.		Visualiza el componente en el área de trabajo con las nuevas dimensiones indicadas.
4.		Ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos.
Sección 4: "Adicionar relación"		
Flujo básico < Adicionar relación >		
	Actor	**Sistema**
1.	Selecciona los dos componentes a relacionar.	
2.	Indica relacionarlos y el sentido de la relación.	
3.		Verifica que sean sentidos de relaciones y tipos de componentes que se puedan relacionar según el flujo válido para el cálculo de descriptores moleculares. Relaciones válidas según tipo de componentes: **13.** Eventos con Entradas y Profile. **14.** Profile con Weights y Total/Locals. **15.** Total/Locals con Invariants y Salidas.

		Sentido válidos de las relaciones: **16.** Desde Entradas a Eventos. **17.** Desde Eventos a Profile. **18.** Desde Weights a Profile. **19.** Desde Profile Total/Locals. **20.** Desde Invariants a Total/Locals. **21.** Desde Total/Locals a Salidas.
4.		Adiciona la relación entre los componentes seleccionados.
5.		Ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos.
Flujos alternos		
Nº Evento <3.a Relación inválida >		
	Actor	**Sistema**
1.		No permite realizar la relación entre los componentes seleccionados.
2.		Ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos.

Sección 5: “Eliminar relación”		
Flujo básico < Eliminar relación >		
	Actor	**Sistema**
1.	Indica la relación entre componentes que desea eliminar del área de trabajo.	
2.		Elimina la relación entre los componentes.
3.		Ejecuta unas de las acciones en dependencia de donde se ejecutó esta sección: **22.** Si se ejecutó desde el Flujo básico <Gestionar diseño de componente>, del Flujo de eventos, ejecuta la acción 3 del Flujo básico <Gestionar diseño de componente>, del Flujo de eventos. **23.** Si se ejecutó desde el Flujo básico < Eliminar componente >, de la Sección 2: “Eliminar componente”, ejecuta la acción 5 del Flujo básico < Eliminar componente >, de la Sección 2: “Eliminar componente”.

Relaciones	**CU incluidos**	
	CU extendidos	Gestionar configuración de componentes.

2.6 Arquitectura utilizada

La arquitectura del software es la estructura del sistema, la cual cuenta de los elementos del software, las propiedades de estos elementos y las relaciones entre ellos. El principal objetivo de la arquitectura del software es aportar elementos que contribuyan a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permita la comunicación entre los equipos que participen en un proyecto. Para conseguirlo, la arquitectura del software construye abstracciones, materializándolas en forma de diagramas comentados (24).

El Sistema se desarrollará completamente sobre la Plataforma Netbeans, esta plataforma se basa principalmente en la construcción del software mediante la arquitectura basada en componentes, donde cada uno de los componentes tiene definido su propio grupo de funciones; además la plataforma está conformada por otros componentes ya definidos que son imprescindibles para el correcto funcionamiento. Este estilo proporcionará una alta flexibilidad y escalabilidad al sistema. A continuación se presenta una vista de la arquitectura del sistema.

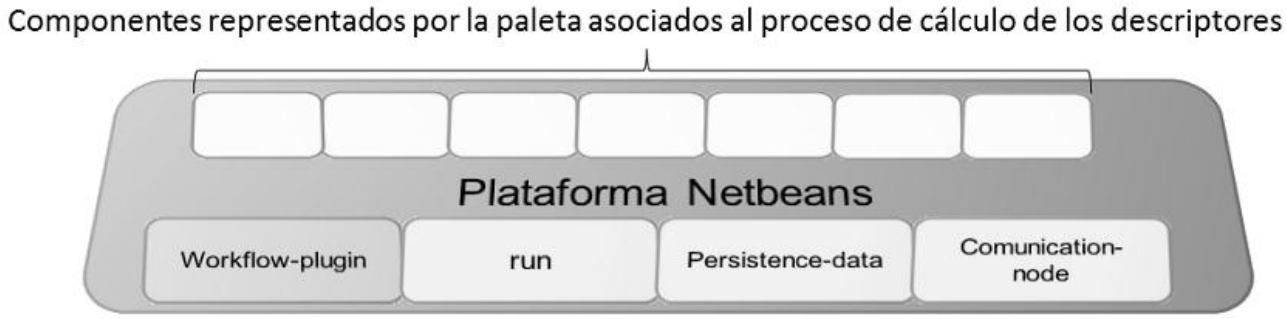

Figura: 8 Vista dela arquitectura del sistema.

- **workflow-plugin:** Este será el módulo principal de la aplicación que representará la paleta y el

área donde se diseñará el flujo de trabajo, además tendrá definida una serie de interfaces las cuales serán implementadas por otros módulos para establecer la comunicación entre ellos.

- **persistence-data:** Este será el módulo que tendrá implementado las funcionalidades de exportar e importar el flujo de trabajo diseñado
- **run:** Este será el módulo que contendrá el API encargada de realizar el proceso de cálculo de los descriptores moleculares y las funcionalidades de la ejecución y parar la ejecución del flujo de trabajo diseñado.
- **comunication-node:** Este será el módulo donde se definirá las relaciones que se podrán establecer entre los distintos tipos de componentes en el área de trabajo.

- **Módulos restantes**: Estos serán los módulos que intervendrán directamente en el diseño del flujo de trabajo los cuales estarán representados en la paleta de componentes. Estos componentes son de tipo (entrada, salida, event, profile, invariants, weights, total/local). Tratar estos módulos como componentes por separado permite la flexibilidad de crear nuevos tipos de componentes, además de hacer muy fácil el cambio sobre cualquiera de ellos.

2.6.1 Arquitectura basada en componentes

La definición de la arquitectura de componentes cubre aspectos únicamente lógicos y es totalmente independiente de la tecnología con la cual se implementarán los componentes. La independencia de la tecnología nos permite abstraernos de los tecnicismos de éstas así como elegir la más apta dependiendo del sistema que se esté desarrollando (25).

Esta arquitectura enfatiza la separación de asuntos por lo que se refiere a la funcionalidad de amplio rango disponible a través de un sistema de software dado. Es un acercamiento basado en la reutilización para definir, implementar, y componer componentes débilmente acoplados en sistemas. Esta práctica persigue un amplio grado de beneficios tanto en el corto como el largo plazo, para el software en sí mismo y para las organizaciones que patrocinan tal software.

2.7 Estructura del modelo de diseño

Para el desarrollo del Modelo de Diseño se hace necesario organizar al mismo en: subsistemas y

paquetes con sus interfaces y las dependencias entre éstos. Esta representación es muy significativa para la arquitectura en general, debido a que los subsistemas y sus interfaces constituyen la estructura fundamental del producto de software. A continuación se muestra en la figura: 8 las relaciones y dependencias entre los distintos subsistemas y paquetes.

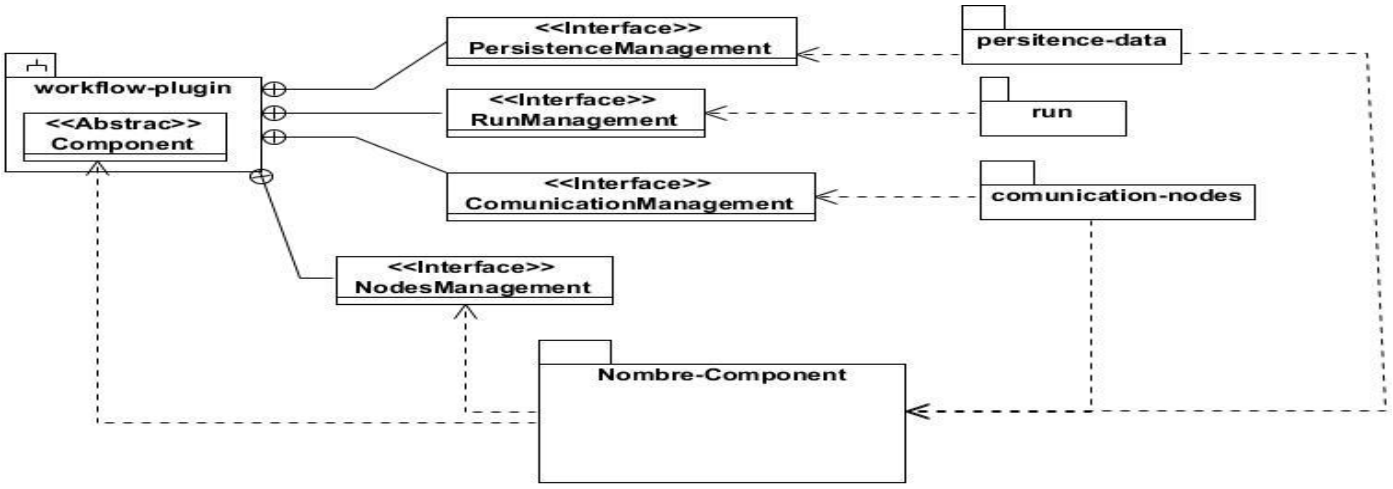

Figura: 9 Diagrama de paquete de la Aplicación

Teniendo en cuenta la importancia que representan las clases interfaces para la estructura del software se muestra a continuación la descripción de estas en forma de tablas.

Tabla 3. Descripción de la clase interface *NodesManagement.*

Nombre: NodesManagement	
Tipo de clase: Interface	
Responsabilidades: Interface que se define para añadir componentes a la paleta.	
Nombre:	**getNode()**
Descripción:	Se define para personalizar el nodo en cuanto a la sección de la paleta que va a pertenecer, el estereotipo que va a presentar y el nombre por el cual el usuario lo identifica.
Nombre:	**createComponent(TransferHandler.TransferSupport support, List<Component>components,ComponentPanel area)**
Descripción:	Se define para crear el componente en el área de trabajo y especifica el comportamiento que va a tener este en dicha área.

Tabla 4. Descripción de la clase interface *PersistenceManagement.*

Nombre: PersistenceManagement	
Tipo de clase: Interface	
Responsabilidades: Interface que se define para cargar y salvar el diseño del área de trabajo.	
Nombre:	**save(ObjectOutputStream oos, List<Component> components, List<JConnector> connectors)**
Descripción:	Se define para salva los componentes que estén en el área de trabajo en un archivo a un directorio local.
Nombre:	**loadComponents(ObjectInputStream ois)**
Descripción:	Se define para cargar los componentes en un archivo desde un directorio local para mostrarlo en el área de trabajo.
Nombre:	**loadConnectors(ObjectInputStream ois)**
Descripción:	Se define para cargar los conectores en un archivo desde un directorio local para mostrarlo en el área de trabajo.

Tabla 5. Descripción de la clase interface *RunManagement.*

Nombre: RunManagement	
Tipo de clase: Interface.	
Responsabilidades: Interface que se define para ejecutar y parar la ejecución del flujo de trabajo diseñado en el área de trabajo.	
Nombre:	**Run(List<Component>components)**
Descripción:	Se define para ejecutar el flujo de trabajo diseñado en el área de trabajo.
Nombre:	**Stop()**
Descripción:	Se define para parar la ejecución del flujo de trabajo ejecutándose.

Tabla 6. Descripción de la clase interface *ComunicationManagement*.

Nombre: ComunicationManagement

Tipo de clase: Interface.	
Responsabilidades: Interface que se define para validar las relaciones entre los componentes	
Nombre:	**CanConnect(Component source, Component destination)**
Descripción:	Se define para validar las relaciones entre los componentes en el área de trabajo.

Tabla 7 Descripción de la clase abstracta *Component*.

Nombre: Component	
Tipo de clase: Abstracta	
Responsabilidades: Extiende de la clase *JComponent de java* dando la posibilidad de personalizar los componentes que van a estar representados en la paleta, otorgándoles comportamientos específicos para ser utilizados en el diseño del flujo de trabajo asociado al proceso de cálculo de los descriptores moleculares. Dentro de los comportamientos que se redefinen se encuentran: *MouseAdapter:* Se implementan los nuevos comportamientos *que* presentaran los eventos producidos con el mouse. *KeyAdapter:* Se implementan los nuevos comportamientos *que* presentaran los eventos producidos con el teclado.	
Nombre:	**resize()**
Descripción:	Pinta el componente con sus nuevas dimensiones.
Nombre:	**checkPopup(MouseEvent e)**
Descripción:	Muestra un menú con las opciones de conectar, eliminar conexión, abrir y eliminar las cuales estarán visibles o no en dependencia del componente donde se produzca el evento.
Nombre:	**fillPopupMenu(JComponent c)**
Descripción:	Construye el menú que tendrán los componentes en el área de trabajo asociados al proceso de cálculo de los descriptores moleculares.
Nombre:	**setLocation(int x, int y)**
Descripción:	Traslada el nombre que representa al componente hacia el lugar que se mueva el componente en el área de trabajo.
Nombre:	**execute(HashMap map)**
Descripción:	Este es un método abstracto que deberá ser redefinido por todos los componentes que se creen, para de esta forma guardar la configuración que se realice sobre el componente en

	el área de trabajo.

2.8 Patrones de diseño

Los patrones de diseño son soluciones reutilizables de problemas recurrentes que aparecen durante el proceso de diseño de software orientado a objetos. Estos surgen por la necesidad de transmitir la experiencia a los demás desarrolladores. Dentro de ellos se encuentran los patrones GRASP (General Responsability Asignment Software Patterns, en español Patrones Generales de Software para Asignar Responsabilidades) (26).

2.8.1 Patrones GRASP utilizados

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Constituyen el fundamento de cómo se diseñará el sistema. Los más importantes son: Experto, Creador, Alta Cohesión, Bajo Acoplamiento y Controlador.

Controlador: Asigna la responsabilidad de recibir o manejar un mensaje de evento del sistema a una clase. En la aplicación se hace uso de este patrón usando la clase controladora DescriptorController.java que tiene la responsabilidad de controlar todo el flujo de eventos asociado la gestión del flujo de trabajo.

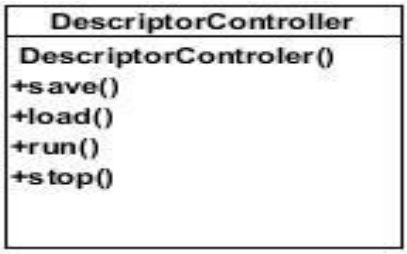

Figura: 10 Clase *DescriptorController del paquete controller del módulo workflow-plugin.*

Experto: Propone asignar una responsabilidad al experto en información, la clase que tiene la información necesaria para la realización de la asignación. Se hace uso del mismo en el módulo principal workflow-

plugin en el paquete component en la clase JConnector.java indicando que la responsabilidad de la creación de una línea de conexión representada por la clase ConnectLine.java debe caer sobre la misma ya que posee la información necesaria para crearlo.

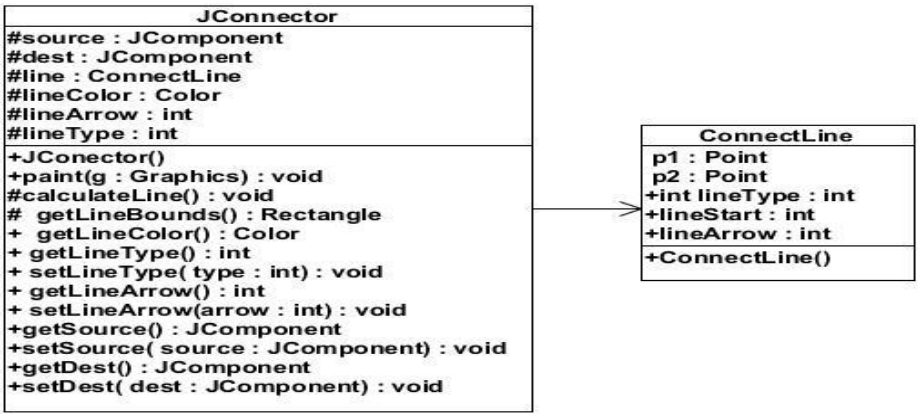

Figura: 11 Evidencia del patrón creador.

Creador: Identifica quién debe ser el responsable de crear o instanciar nuevos objetos de clases. En la aplicación se hace uso de este patrón en la clase FlowComponentNode.java que es la responsable de crear instancias de objetos de la clase FlowComponent.java.

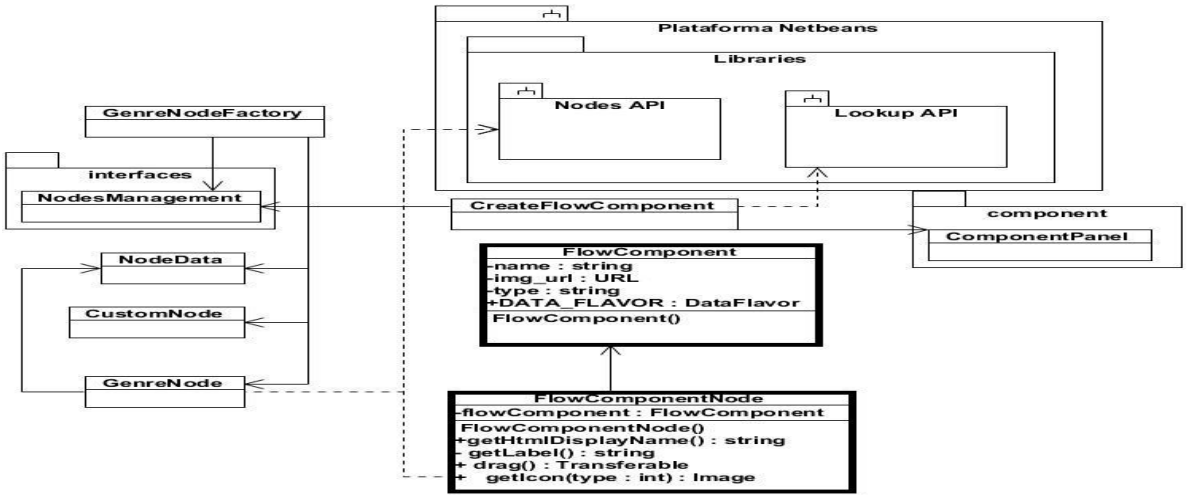

Figura: 12 Diagrama de clase de diseño del paquete node del módulo workflow-plugins

Bajo acoplamiento: Propone tener las clases con la menor dependencia que se pueda entre ellas. De tal forma que en caso de producirse una modificación en alguna, se tenga la mínima repercusión posible en el resto de clases, potenciando así la reutilización. Se evidencia en la clase FlowComponentNode.java debido a que la dependencia de ellas con otras clases es baja, esta clase solo recurre a la entidad FlowComponent.java.

Alta cohesión: Propone asignar responsabilidades a las clases evitando que estas realicen un trabajo excesivo y que las tareas no sean afines. En la solución se observa el uso de este patrón en la clase DescriptorController.java ya que no realizan un trabajo excesivo y sus responsabilidades son afines ya que es la encargada de realizar las acciones asociadas a la gestión del flujo de trabajo.

2.9 Conclusiones

En este capítulo la confección del diagrama de clases del dominio permitió visualizar la relación existente entre las clases u objetos significativos en el dominio del problema. Con el levantamiento de requisitos se obtuvieron 13 requisitos funcionales agrupados en 4 casos de uso los que fueron descritos textualmente para obtener una visión más detalla del flujo de actividades a implementar en la aplicación. El diseño de los diagramas de casos de uso del sistema brindó la posibilidad de mostrar gráficamente los procesos y su interacción con los actores. La utilización de la arquitectura basada en componentes, permitió separar la implementación de la aplicación en varios módulos, posibilitando que los cambios realizados en uno de ellas no tengan una gran repercusión en el resto. Una vez concluido el diseño de la solución se obtuvo como resultado una serie de diagramas del diseño los cuales permitieron comprender detalladamente las características del sistema.

Capítulo 3: Implementación y prueba

En este capítulo, se expondrá el diagrama de componentes, donde se describen los elementos físicos del sistema y sus relaciones. Además se explicarán elementos necesarios para la extensibilidad e implementación de los componentes que formaran parte del proceso de cálculo de los descriptores moleculares basados en teoría de la información. Por último, se describen detalladamente las pruebas que se le realizan al sistema, para corroborar el correcto funcionamiento de la aplicación cumpliendo con los requisitos implementados.

3.1 Modelo de implementación

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

3.2 Diagrama de componentes

Un diagrama de componentes muestra las organizaciones y dependencias lógicas entre componentes de software: código fuente, binarios o ejecutables. Desde el punto de vista del diagrama de componentes se tienen en consideración los requisitos relacionados con la facilidad de desarrollo, la gestión del software, la reutilización, y las limitaciones imputadas por los lenguajes de programación y las herramientas utilizadas en el desarrollo (27).

Un componente es una parte física y reemplazable de un sistema conformado con un conjunto de interfaces y proporciona la realización de dicho conjunto. Se usan para modelar los elementos físicos que pueden hallarse en un nodo, por lo que empaquetan elementos como clases, colaboraciones e interfaces (27). En la figura: 9 se muestra el diagrama de componentes del sistema de forma general.

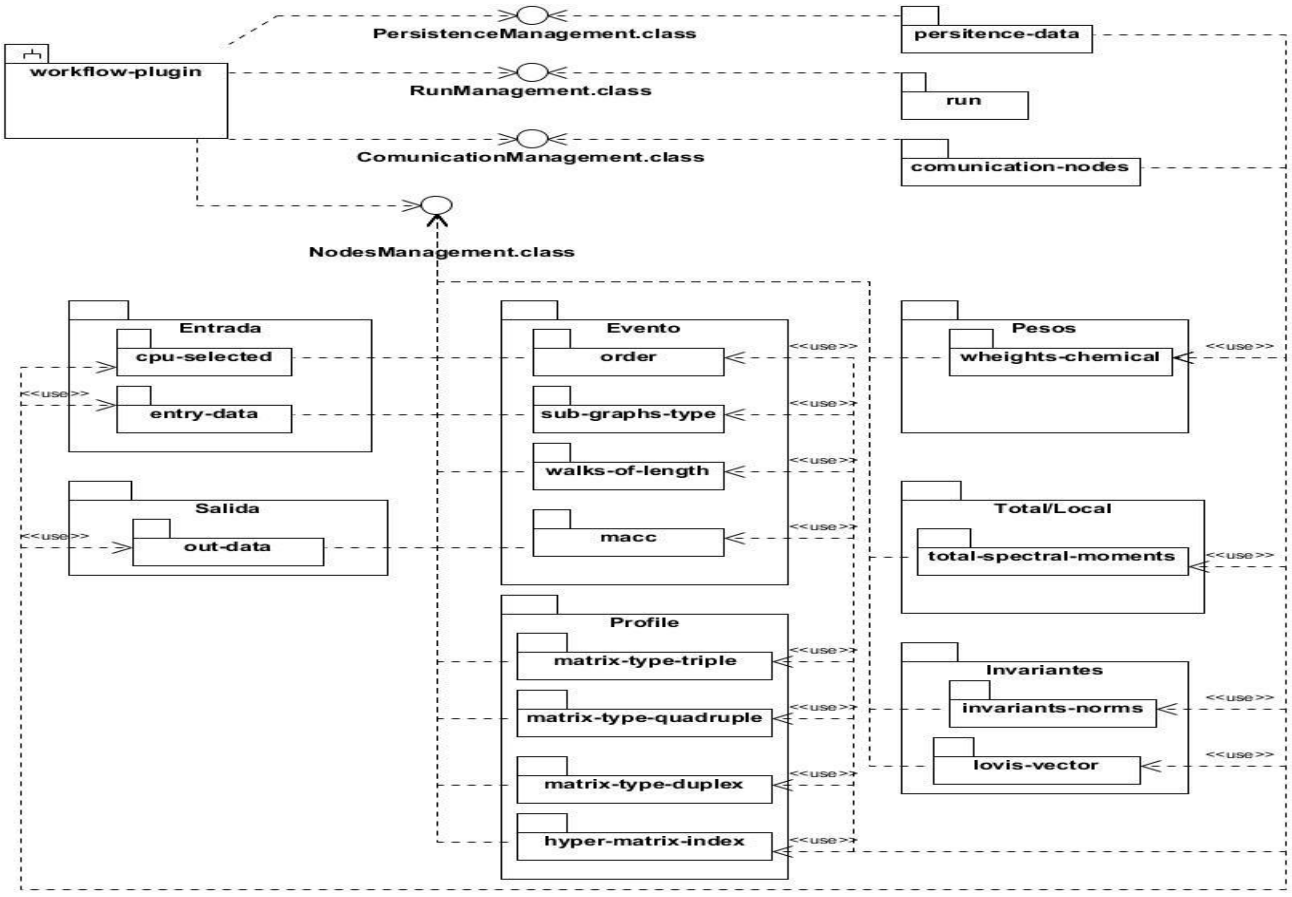

Figura: 13 Diagrama de componentes del sistema.

3.3 Código fuente

El Código Fuente es un conjunto de líneas que conforman un bloque de texto, escrito según las reglas sintácticas de algún lenguaje de programación destinado a ser legible por humanos. Es un programa en su forma original, tal y como fue escrito por el programador, no es ejecutable directamente por el computador, debe convertirse en lenguaje de maquina mediante compiladores, ensambladores o intérpretes (28).

3.3.1 Estándar de codificación

Un estándar de codificación completo comprende todos los aspectos de la generación de código. Si bien

los programadores deben implementar un estándar de forma prudente, éste debe tender siempre a lo práctico. Un código fuente completo debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código de una sola vez. La legibilidad del código fuente repercute directamente en lo bien que un programador comprende un sistema de software. La mantenibilidad del código es la facilidad con que el sistema de software puede modificarse para añadirle nuevas características, modificar las ya existentes, depurar errores, o mejorar el rendimiento (29). Usar técnicas de codificación sólidas y realizar buenas prácticas de programación con vistas a generar un código de alta calidad es de gran importancia para la calidad del software y para obtener un buen rendimiento.

Estilo de codificación utilizado

- Los nombres de variables y métodos estarán escritos con el estándar “lowerCamelCase”.
- Las palabras reservadas del lenguaje así como las construcciones que se asimilan a funciones (array(), list(), etc.) se escribirán todas en minúsculas.
- Las variables usadas en cualquier aplicación deben tener nombres que las identifiquen, es decir, nombres como "a", "b" o "c" no se deberían usar sino que deberían tener un nombre coherente con lo que contienen.

3.4 Extensión de la aplicación

El diseño arquitectónico de la aplicación permite que puedan ser agregados nuevos componentes a la paleta. Para la adición de estos componentes, con el objetivo de extender la aplicación informática es necesario tener en cuenta algunos aspectos importantes que servirán de guía y apoyo para un futuro enriquecimiento de esta. En base a obtener una estandarización en el diseño de estos componentes se muestra a continuación un diagrama de clases.

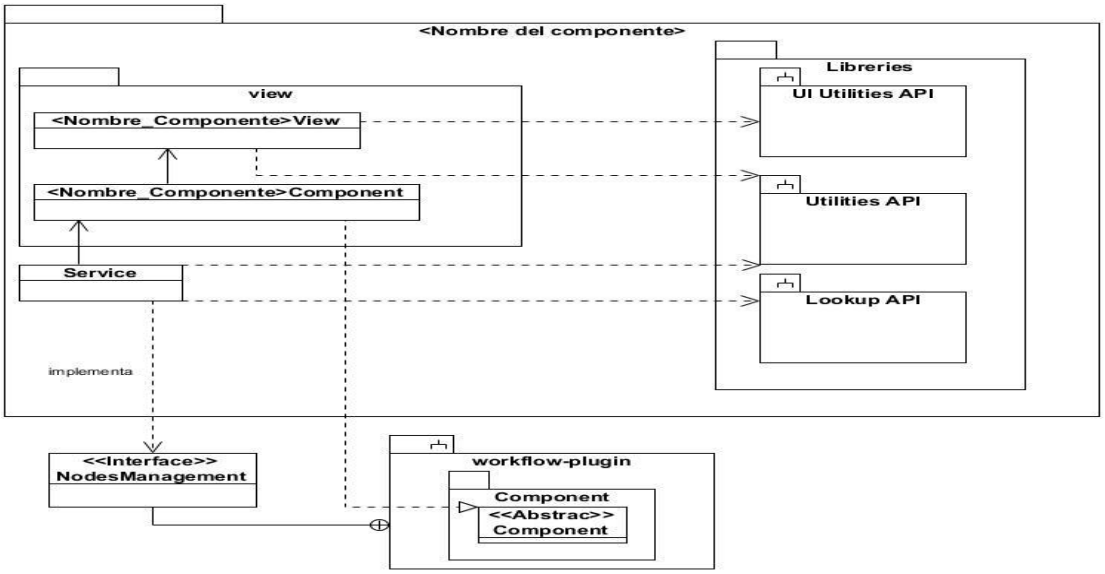

Figura: 14 Diseño de clases para la implementación de los componentes de la paleta.

1. Una vez creado un módulo, debe añadírsele como dependencia el módulo principal workflow-plugins para extender de la clase *Component* (ver tabla 7) y de esta forma heredar funcionalidades que definen comportamientos comunes en todos los componentes una vez adicionado a la plataforma.

2. Se debe implementar la interfaz *NodesManagement* del módulo principal workflow-plugins y redefinir los métodos *getNode* y *createComponent.* El *getNode* se utiliza para personalizar el nodo en cuanto a la sección de la paleta que va a pertenecer, el estereotipo que va a presentar y el nombre por el cual el usuario lo identifica; el *createComponent* es el encargado de crear el componente en el área de trabajo y especificar parte del comportamiento que va a tener este en dicha área. A continuación se muestra.

```
@ServiceProvider(service = NodesManagement.class)
public class Service implements NodesManagement {
    @Override
    public CustomNode getNode() {
        return new CustomNode("Entrada",
                              new NodeData("Entrada Datos",
                                           getClass().getResource("/molecular/descriptor/entry_data/imagenes/Loading.png"),
                                           "load_data"));
    }
    @Override
    public Component createComponent(TransferHandler.TransferSupport ts, List<Component> list, ComponentPanel pnl) {
        FlowComponent flowComponent = null;
        try {
            flowComponent = (FlowComponent) ts.getTransferable().getTransferData(FlowComponent.DATA_FLAVOR);
        } catch (UnsupportedFlavorException ex) {
            Exceptions.printStackTrace(ex);
        } catch (IOException ex) {
            Exceptions.printStackTrace(ex);
        }
        Component component = null;
        if (flowComponent.getType().equals("load_data")) {
            component = new FileEntryComponent(pnl, list, flowComponent.getName(), ts.getDropLocation().getDropPoint());
        }

        return component;
    }
}
```

Figura: 15 Implementación de la clase interfaz *NodesManagement.*

3. El módulo creado debe ser añadido como dependencia al módulo comunication-node y definir con cuales componentes se podrá relacionar.

4. El módulo creado debe ser añadido como dependencia al módulo persistence-data para que pueda ser cargado y salvado el componente en el área de trabajo.

3.5 Vistas de la aplicación

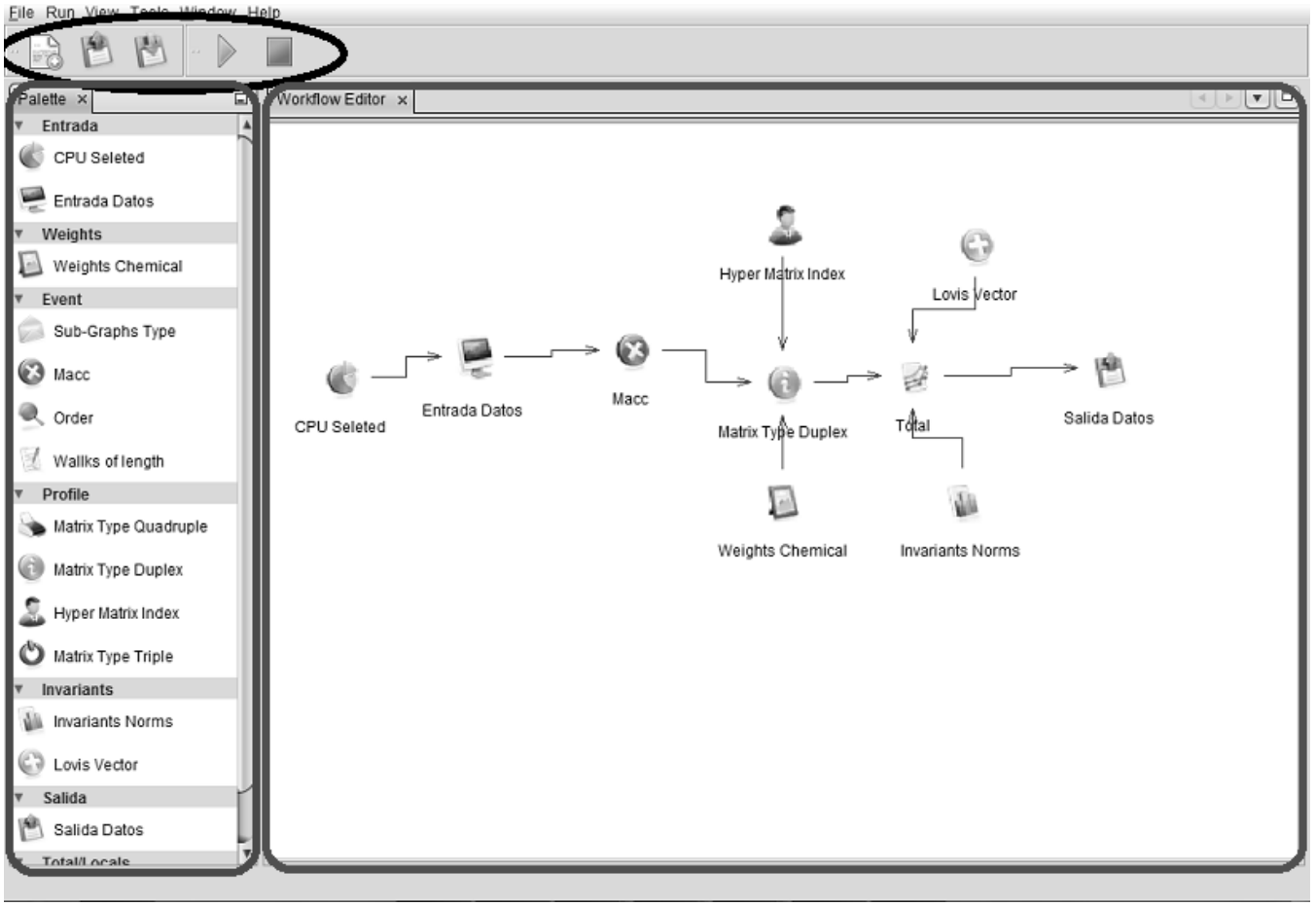

Figura: 16 Vista principal de la aplicación

La interfaz gráfica provee tres áreas fundamentales:

1. El área marcada en la parte superior es donde se encuentran los botones con imágenes sugerentes a las acciones relacionadas con la gestión del flujo diseñado y el área de trabajo (adicionar nueva área de trabajo, cargar flujo de trabajo, salvar flujo de trabajo, ejecutar flujo de trabajo y parar ejecución del flujo de trabajo), presentando Tooltip para proporcionar una mayor información al usuario.

2. El área marcada a la izquierda es la paleta con todos los componentes asociados al proceso de cálculo de los descriptores moleculares basados en teoría de la información, dando una clara representación al usuario de todos los componentes y evitando que este haga uso de su propia memoria, no forzándolos a recordar eventos innecesarios.

3. El área marcada a la derecha es donde se diseña gráficamente y se configura el proceso de cálculo de los descriptores moleculares basados en teoría de la información. A continuación se presenta una imagen.

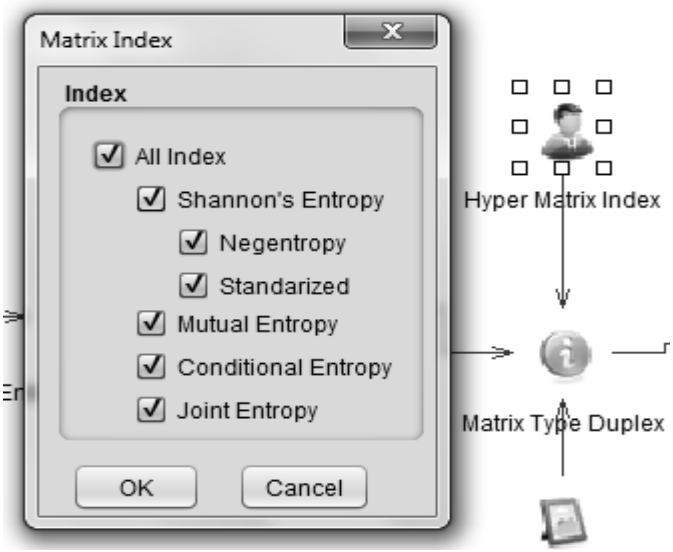

Figura: 17 Configuración del componente Hyper Matrix Index.

3.6 Pruebas de calidad del software.

El instrumento adecuado para determinar el estado de la calidad de un producto de software es el proceso de pruebas. En el mismo se ejecutan pruebas dirigidas a componentes del sistema de software en su totalidad, con el objetivo de medir el grado en que la aplicación cumple con los requisitos (30).

Cuando un sistema tiene terminado una serie de clases y componentes el software debe ser probado para descubrir y corregir el número máximo de errores posible antes de su entrega al cliente. El objetivo es

diseñar una serie de casos de prueba que tengan una alta probabilidad de encontrar errores. Para ello se aplican una serie de técnicas y métodos de pruebas del software.

3.6.1 Diseño de prueba de caja negra

Las pruebas de caja negra permitirán identificar las posibles fallas en el funcionamiento del sistema. Estas pruebas se centran principalmente en las características del sistema independientemente del código. Para llevar a cabo estas pruebas es necesario tener conocimiento sobre los escenarios de prueba y secciones que serán probadas (31).

Estas pruebas se realizan a nivel de sistema, el tipo de prueba es funcional, donde se emplea el método de caja negra, específicamente haciendo uso de la técnica partición de equivalencia. Esta técnica es muy efectiva al realizar pruebas sobre la interfaz de la plataforma, pues prueban la validez de acciones realizadas sobre el sistema tomando como punto de observación la respuesta que da este. Pretenden demostrar que las funciones de la plataforma son operativas (31).

Tabla 8 Caso de prueba para el caso de uso Gestionar componentes en el área de trabajo.

Nombre la sección	Escenario	Descripción de la funcionalidad	Flujo central
SC1 Adicionar componente.	EC 1.1: Se adiciona el componente al área de trabajo.	El investigador debe seleccionar el componente en la paleta e indicar el lugar en el área de trabajo donde quiere colocarlo.	1- Ir a la paleta de componentes. 2- Seleccionar el componente con click primario. 3- Arrastrar el componente hacia el área de trabajo e indicar el lugar donde se quiere colocar.
SC2 Eliminar	EC 2.1: Se elimina el	El investigador debe seleccionar el	1- Ir al área de trabajo.

componente	componente del área de trabajo sin relación.	componente en el área de trabajo e indicar eliminarlo.	2- Seleccionar el componente con click secundario. 3- Seleccionar la opción: delete connection.
	EC 2.2: Se elimina componente del área de trabajo con relaciones.	El investigador debe seleccionar el componente en el área de trabajo e indicar eliminarlo. Si el componente presenta relación con otros componentes, también son eliminadas las relaciones.	1- Ir al área de trabajo. 2- Seleccionar el componente con click secundario. 3- Seleccionar la opción: delete conection.
SC3 Modificar dimensiones del componente	EC 3.1: Se modifican las dimensiones del componente en el área de trabajo.	El investigador debe seleccionar el componente en el área de trabajo y modificar sus dimensiones	1- Ir al área de trabajo. 2- Seleccionar el componente con click primario. 3- Modificar sus dimensiones.
SC4 Relacionar componentes	EC 4.1: Se relacionan componentes en el área de trabajo correctamente.	El investigador debe seleccionar dos componentes a relacionar en el área de trabajo e indicar el sentido de su relación.	1- Ir al área de trabajo. 2- Seleccionar los componentes a relacionar con click

			primario. 3- Seleccionar el componente en dependencia del sentido de la conexión con click secundario. 4- Seleccionar la opción: connection.
	EC 4.2: Se relacionan componentes en el área de trabajo incorrectamente.	El investigador debe seleccionar dos componentes a relacionar en el área de trabajo e indicar el sentido de su relación. El sistema no da la opción de poder relacionar los componentes.	1- Ir al área de trabajo. 2- Seleccionar los componentes a relacionar con click primario. 3- Seleccionar el componente en dependencia del sentido de la conexión con click secundario. 4- Seleccionar la opción: connection.
SC5 Eliminar relación.	EC 5.1: Se elimina la relación entre componentes en el área de trabajo.	El investigador debe seleccionar la relación e indicar eliminarla.	1- Ir al área de trabajo. 2- Seleccionar la conexión con click secundario. 3- seleccionar la opción: Delete.

Resultados de las pruebas de caja negra

Después de realizar las pruebas de caja negra mediante los casos de prueba asociados a cada caso de uso, se comprobó el correcto funcionamiento de la aplicación, verificando que cada uno de sus requisitos se ejecutara correctamente. A continuación se muestra en la figura que en la primera iteración fueron detectadas 7 no conformidades las cuales se resolvieron en su totalidad, en una segunda iteración se detectaron 4 no conformidades las cuales fueron resueltas y en una última y tercera iteración no se detectaron no conformidades.

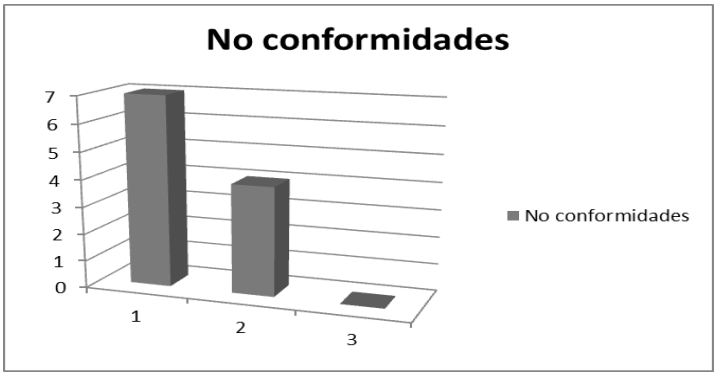

Figura: 18 Gráfica de la aplicación de los casos de pruebas de caja negra por iteraciones

3.6.2 Pruebas de integración

Las Pruebas de Integración son aquellas que permiten probar en conjunto distintos subsistemas funcionales o componentes del proyecto para verificar que interactúan de manera correcta y que se ajustan a los requisitos especificados. Este tipo de pruebas deberán ejecutarse una vez se haya asegurado el funcionamiento correcto de cada componente implicado por separado, es decir, una vez se hayan ejecutado sin errores las pruebas unitarias de estos componentes.

Para evaluar el comportamiento del módulo una vez integrado se realizarán pruebas funcionales, se mantiene la utilización del mismo tipo de prueba, método y herramienta utilizados en las pruebas a nivel de desarrollador. Estas pruebas tienen el objetivo de verificar si durante la integración de un módulo con otro presentan alguna dificultad para su correcto funcionamiento (32).

Ejecución de las pruebas de integración

Se realizaron dos iteraciones de pruebas funcionales aplicándose los 4 casos de prueba diseñados y se detectaron cinco no conformidades en la primera iteración, de ellas tres de código y dos de interfaz. En la revisión realizada en la segunda iteración se verificó que las no conformidades de la etapa anterior estuvieran resueltas y no se detectaron nuevas no conformidades. Esto demuestra que el sistema está libre de errores.

Figura: 19Gráfica de la aplicación de las pruebas de integración por iteraciones

3.7 Conclusiones

En el desarrollo del presente capítulo la realización del modelo de implementación de la aplicación permitió demostrar los componentes del sistema y sus relaciones, a través del diagrama de componentes. Los estándares de codificación y los estilos de programación utilizados fueron descritos para hacer más fácil el entendimiento del código por los programadores y para permitir un mejor mantenimiento a la aplicación en un futuro. Para evaluar la calidad de la aplicación se validó la completitud de los requisitos con las pruebas funcionales al software. Se identificaron un total de 14 no conformidades, las cuales fueron resueltas paulatinamente obteniéndose un producto libre de errores y listo para su ejecución.

Conclusiones generales

El desarrollo del presente trabajo se transitó por una serie de etapas que permitieron dar cumplimiento al objetivo planteado, arribando a las siguientes conclusiones:

- El estudio de los principales conceptos relacionados con las interfaces gráficas de usuario y el análisis de algunas herramientas permitió sentar las bases para el desarrollo de la interfaz gráfica de usuario que facilitará el proceso de cálculo de los descriptores moleculares basados en teoría de la información mediante el diseño de flujos de trabajos.

- A partir de la realización del análisis y diseño de la interfaz gráfica de usuario se obtuvo como resultado los diagramas y artefactos necesarios para guiar el desarrollo del mismo.

- Con el diseño y ejecución de las pruebas a nivel de desarrollador y de integración, se logró validar el correcto funcionamiento de la aplicación informática para la realización del diseño gráfico del proceso de cálculo de los descriptores moleculares.

- Se obtuvo una aplicación informática basada en la Plataforma Netbeans construida a partir de varios módulos que permite el diseño gráfico del proceso de cálculo de los descriptores moleculares basados en teoría de la información y la adición de nuevos módulos en tiempo de ejecución para el enriquecimiento de las funcionalidades de esta.

Recomendaciones

Dados los resultados de la investigación, en relación con el problema que aborda, se sugiere:

1. Continuar el estudio de las potencialidades que brinda la Plataforma Netbeans para la mejora en la interacción de los usuarios con el software propuesto y la adición de nuevos módulos en tiempo de ejecución para el enriquecimiento de las funcionalidades de esta.
2. Brindar la posibilidad de que el usuario pueda realizar varios procesos de cálculos a partir de una misma base de datos de moléculas.

Referencias bibliográficas

1. **Barigye, Stephen Jones.** *TEORÍA DE INFORMACIÓN EN LA CODIFICACIÓN DE LA ESTRUCTURA QUIMICA.* [Tesis presentada en opción al grado científico de Doctor en Ciencias Químicas] Santa Clara : s.n., 2013.

2. **Trujillo, Alex Giovanny Peniche.** *La quimioinformática, una herramienta eficiente para desarrollar los medicamentos del futuro.* [Documento] s.l. : Fundación Universitaria del Área Andina, 2011. ISSN 1900-9380.

3. **Todeschini, Roberto y Consonni, Viviana.** *Handbook of Molecular Descriptor.* [WILEY-VCH] 2008.

4. **Carbonell, Moises Mañas.** FundéuBBVA. *fundeu.es.* [En línea] 10 de 1 de 2013. [Citado el: 23 de 3 de 2014.] http://www.fundeu.es/escribireninternet/interfaz-grafica-de-usuario-gui/.

5. **Aimacaña, Carlos Toledo.** Interfaz de Usuario. *Monografías.com.* [En línea] Monografías.com S.A. [Citado el: 15 de 2 de 2014.] http://www.monografías.com/trabajos6/inus/inus.shtml.

6.**Sabatini, Alejandro Gustavo.** Intercacción Humano Máquina. *interfacemindbraincomputer.wikifoundry.com.* [En línea] [Citado el: 15 de 3 de 2014.] http://interfacemindbraincomputer.wikifoundry.com/page/2.A.1.1.2.2.Interfaces%3A+Desarrollo+basado+en+modelos

7. **Espinosa, Roberto.** El Rincon del BI. [En línea] 10 de 5 de 2010. [Citado el: 15 de 1 de 2014.] http://churriwifi.wordpress.com/2010/05/10/16-3-construccion-procesos-etl-utilizando-kettle-pentaho-data-integration/..

8. **Hall, Mark y Reutemann, Peter.** *WEKA KnowledgeFlow Tutorial.* Waikato : s.n., 2008.

9. **Roberth G. Figueroa, Camilo J. Solís,Armando A. Cabrera3.** *METODOLOGÍAS TRADICIONALES VS. METODOLOGÍAS ÁGILES.* Loja : Universidad Técnica Particular de Loja, Escuela de Ciencias en Computación.

10. **Roque, Diana Monné y Garnache, Alberto.** *OpenUp como metodología aplicada en el desarrollo de losproductos del software del Departamento de integración de soluciones.* Habana : s.n., 2012.

11. **Rumbaugh, James, Jacobson, Ivar y Booch, Grady.** *El lenguaje unificado de modelado.Manual de referencia.*

12. Software.com.ar. [En línea] [Citado el: 23 de 2 de 2014.] www.software.com.ar/visual-paradigm-para-uml.html..

13. sunmartimoran.wordpress. [En línea] 25 de 1 de 2013. [Citado el: 4 de 3 de 2014.] http://sunmartimoran.wordpress.com/2013/01/25/entorno-de-desarrollo-integrado-o-ide/.

Referencias bibliográficas

14. kioskea.net. [En línea] 2 de 2014. [Citado el: 25 de 3 de 2014.] es.kioskea.net/contents/304-lenguajes-de-programacion.

15. Developer Network. *msdn.microsoft.com.* [En línea] [Citado el: 23 de 3 de 2014.] msdn.microsoft.com/es-es/library/z1zx9t92.aspx.

16. Java. *infor.uva.es.* [En línea] [Citado el: 12 de 2 de 2014.] www.infor.uva.es/~jmrr/tgp/java/JAVA.html.

17. Curso C++. *zator.com.* [En línea] Zator Systems. [Citado el: 15 de 2 de 2014.] www.zator.com/Cpp/E1_2.htm.

18. NetBeans. *netbeans.org.* [En línea] Oracle Corporation. [Citado el: 17 de 2 de 2014.] netbeans.org/features/index.html.

19. Genbeta : det. *genbetadev.com.* [En línea] 14 de 1 de 2014. [Citado el: 2 de 3 de 2014.] http://www.genbetadev.com/herramientas/eclipse-ide.

20. **Bock, Heyko.** *The Definitive Guide to NetBeans Plataform 7.*

21. **Somerville, Ian.** *Ingenieria del software 7 edición.* Madrid : Pearson Educación, 2005. ISBN/84-7829-074-5.

22. Marco de Desarrollo de la Junta de Andalucía. *juntadeandalucia.es.* [En línea] [Citado el: 4 de 3 de 2014.] http://www.juntadeandalucia.es/servicios/madeja/contenido/recurso/416.

23. **ROB, STEVENS.** *Utilización de UML en Ingienería del Software con Objetos y Componentes.* s.l. : Addison Wesley, 2007.

24. **Cervantes, Humberto.** SG. [En línea] 15 de 4 de 2010. [Citado el: 8 de 4 de 2014.] http://sg.com.mx/revista/27/arquitectura-software.

25. **Vignaga, Andrés y Perovich, Daniel.** *Enfoque metodológico para el desarrollo basado en componentes.* [Documento] Uruguay : Instituto de Computación, Facultad de Ingieneria.

26. **Larman, Craig.** *UML Y Patrones.*

27. wikispaces. [En línea] [Citado el: 13 de 4 de 2014.] http://wikiuml.wikispaces.com/Diagrama+de+Componentes.

28. Pergamino Virtual. [En línea] [Citado el: 5 de 5 de 2014.] http://www.pergaminovirtual.com.ar/definicion/Codigo_Fuente.html.

Referencias bibliográficas

29. Microsof Developen Network. [En línea] [Citado el: 6 de 4 de 2014.] http://msdn.microsoft.com/es-es/library/aa291591%28v=vs.71%29.aspx.

30. Pruebas de Software. [En línea] [Citado el: 6 de 5 de 2014.] http://www.pruebasdesoftware.com/laspruebasdesoftware.htm.

31. GlobeTesting. [En línea] [Citado el: 7 de 5 de 2014.] http://www.globetesting.com/2012/08/pruebas-de-caja-negra/.

32. **Somerville, Ian.** *Ingienería del software.* Madrid(España) : Pearson Education S.A, 2005. ISBN: 84-7829-074-5.

33. Tecnología y Synergix. [En línea] [Citado el: 8 de 5 de 2014.] http://synergix.wordpress.com/2008/07/10/modelo-de-dominio/.