Universidad de las Ciencias Informáticas
Facultad 6

Título: Módulo Editor Molecular para la plataforma alasGRATO.

Trabajo de Diploma para optar por el título de

Ingeniero Informático

Autores: Andrés Miguel Bores Hevia

Félix Manuel Bores Hevia

Tutores: Dr. Ramón Carrasco

MSc. Aurelio Collado

Ing. Julio Villaverde

Junio de 2010

"El esfuerzo de utilizar las máquinas para emular el pensamiento humano siempre me ha parecido bastante estúpido. Preferiría usarlas para emular algo mejor."

Edsger Wybe Dijkstra

DECLARACIÓN DE AUTORÍA

Declaramos ser autores de la presente tesis y reconocemos a la Universidad de las Ciencias Informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

<nombre autor>

_____
Firma del Autor

<nombre tutor>

_____
Firma del Tutor

DATOS DE CONTACTO

Autores:

Félix Manuel Bores Hevia **email: fmbores@estudiantes.uci.cu**

Andrés Miguel Bores Hevia **email: ambores@estudiantes.uci.cu**

Tutores:

Ramón Carrasco Velar:

Doctor en Ciencias Químicas, profesor asistente.

e-mail: rcarrasco@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Aurelio Antelo Collado:

Máster en Matemática Aplicada, Ingeniero Industrial, profesor asistente.

e-mail: aantelo@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

Julio Antonio Villaverde Martínez:

Ingeniero en Ciencias Informáticas.

e-mail: jvillaverde@uci.cu

Universidad de las Ciencias Informáticas, Ciudad de La Habana, Cuba.

AGRADECIMIENTOS

Félix:

A mis padres, por haberme enseñado todo lo que necesito para triunfar en la vida. Les debo a ellos lo que soy. Los quiero con todo mi corazón.

A mi abuela, por el infinito amor que siempre me ha dado. Ojala y siempre estés ahí para mí.

A mi hermano, por haberme dado siempre lo mejor y por haberme cuidado siempre. Te quiero.

A mi hermanita, por ser motivo de inspiración. Te quiero.

A mis tíos, por ser una enorme fuente de conocimientos a la que siempre acudiré.

A mis 4 hermanos, Francisco, Frank, Miguel y Víctor, por todo lo que hemos compartido, y por estar ahí siempre que lo necesito.

A Elizabeth, por haber compartido con ella mis mejores momentos en esta universidad y en la vida. Te quiero mucho.

A Luis, Roberto, Dairon, por el gran conocimiento que me han brindado. Los quiero hermanos.

A Juan Carlos, sin tu ayuda hubiera sido realmente difícil lograr esto. Gracias hermano.

A mis tutores Carrasco, Aurelio y Julio, por brindarme su apoyo en todo momento. Gracias de corazón.

A todos los fans de la buena música, por los grandes momentos que hemos vivido juntos.

A todas mis amistades que sería imposible nombrar aquí, pero que han influido mucho en todos mis logros.

Andrés:

No encuentro palabras suficientes para agradecer a las personas que de una forma u otra han ayudado a que hoy este aquí.

Agradezco a:

Mis padres Raquel y Felix y mis abuelos, puedo hacer otra tesis para agradecerles y todavía no llegaría ni a la mitad de lo que se merecen. Solo espero que algún día estén tan orgullosos de mí como yo de ellos. Una vida no es suficiente para demostrarle todo lo que los quiero.

Mi hermano, sólo decir que mi vida no es nada sin ti. Te quiero mucho.

Mi hermana, no sabe lo orgulloso que estoy de ella. Te quiero mucho.

Mis tíos y primos, es una suerte tenerlos. Los quiero mucho.

Frank, Francisco, Víctor y Miguel, mis otros hermanos que siempre han estado a mi lado.

Juan Carlos, sin ti no habría podido lograr todo esto. Eres un hermano más.

Luis, Roberto y Dairon, estoy muy orgulloso de tener “amigos” como ustedes.

Eyleen, por quererme tanto. Siempre te recordaré.

Mis tutores Carrasco, Aurelio y Julio, gracias por confiar en mí y darme todo el apoyo que me dieron.

Todos mis amigos y conocidos, con los que he compartido momentos que nunca olvidaré. No los nombro porque cada uno sabe que tan importante es para mí.

Los que compartimos el mismo gusto por “nuestra música”. He aprendido mucho de todos ustedes.

Quisiera agradecer a otras personas pero por cosas de la vida no puedo hacerlo. Espero que esas personas sepan lo mucho que significan para mí.

A todos muchas gracias.

DEDICATORIA

Dedicamos el presente trabajo:

Primeramente, a nuestra familia, fuente inagotable de amor, cariño, aliento y todo lo necesario para creer que somos las personas más felices del mundo.

A nuestros amigos, que para nosotros son familia también.

A todo lo que nos ha inspirado para ser lo que somos hoy.

RESUMEN

Para los químicos es muy importante editar y visualizar cualquier molécula con la cual estén trabajando. Debido a esto, en cualquier proyecto bioinformático en el cual se trabaje con moléculas, es de mucha necesidad dibujarlas en dos dimensiones y posteriormente visualizarlas en tres dimensiones. Mientras más similares sean las dos perspectivas de visualización más cómodo será para todo aquel usuario que trabaje con las mismas. El proyecto "Plataforma Inteligente para la Predicción de Actividad Biológica de Compuestos Orgánicos", el cual se lleva a cabo en la Facultad 6, específicamente en el grupo de Bioinformática, de la Universidad de las Ciencias Informáticas en conjunto con el CITMA (Ministerio de Ciencia, Tecnología y Medio Ambiente), es un ejemplo de esto, por lo cual se está desarrollando un nuevo Editor Molecular 2D. Este editor acercará las dos perspectivas de visualización, brindando las funcionalidades necesarias para poder crear, editar y visualizar una molécula en 2D, además de agregar algunas otras que facilitarán el trabajo a los especialistas.

PALABRAS CLAVE

Molécula, Editor Molecular, 2D, editar, visualizar.

TABLA DE CONTENIDOS

INTRODUCCIÓN

Una de las áreas que dentro de la bioinformática ha tenido mayor difusión y que en general ha acaparado mayor atención, ha sido la relativa a la edición y visualización de estructuras químicas de manera bidimensional y tridimensional respectivamente. Siendo esta representación gráfica de la estructura de las moléculas la forma de comunicarse técnicamente los químicos de cualquier latitud, mientras más detallada y exacta sea, mejor será el entendimiento de los químicos y así, mejores serán los resultados que los mismos puedan llegar a obtener.

Existen dos formas de representar las moléculas, en 2 y en 3 dimensiones. Las mismas son editadas, es decir, dibujadas en 2 dimensiones para posteriormente visualizarlas en 3 dimensiones, donde se aprecia mejor su estructura y da una idea más real de cómo es su estructura.

En la Universidad de las Ciencias Informáticas (UCI), en el grupo de bioinformática (BioSoft) perteneciente a la Facultad 6, se encuentra el proyecto "Plataforma para la Predicción de Actividad Biológica de Compuestos Orgánicos" (alasGRATO), creado en colaboración conjunta con el CITMA. Uno de los objetivos de este macro proyecto es la visualización y edición de moléculas. Para cumplir con ese objetivo en la plataforma alasGRATO se desarrollaron dos módulos: uno de visualización de compuestos orgánicos en 3 dimensiones alasGRATOViewer y otro de edición en 2 dimensiones alasGRATOEditor. Actualmente para visualizar los compuestos se utiliza la librería JMOL y para editar en 2 dimensiones se utiliza la librería JME, siendo estas dos librerías utilizadas libres por la comunidad. Estos dos módulos desarrollados no cuentan con una apropiada compatibilidad entre ellos en cuanto a apariencia se trata, lo cual dificulta el trabajo eficiente de los usuarios y no están en concordancia con los requisitos de usabilidad que se requieren para la segunda versión de la plataforma alasGRATO.

Por lo expresado anteriormente, se plantea como **problema científico** de este trabajo: ¿Cómo lograr que en la plataforma alasGRATO se puedan editar las moléculas sin perder las preferencias de visualización existentes? Teniendo como **objeto de estudio** entonces la representación gráfica de moléculas. Enmarcándose dentro del **campo de acción** edición y visualización de moléculas en 2D.

Para solucionar el problema anteriormente mencionado se plantea, como **objetivo general,** desarrollar un editor molecular para la plataforma alasGRATO sin perder las preferencias de visualización existentes.

Con vistas a cumplimentar este objetivo general se trazaron los siguientes **objetivos específicos**:

- Refinar los requisitos funcionales existentes para la plataforma alasGRATO.

- Diseñar el editor molecular para la plataforma alasGRATO.
- Implementar el editor molecular para la plataforma alasGRATO.
- Validar el funcionamiento del editor molecular para la plataforma alasGRATO

Para alcanzar dichos objetivos se planteó desarrollar las siguientes **tareas**:

- Elaboración del modelo de dominio.
- Obtención de los requisitos funcionales y no funcionales.
- Elaboración de los prototipos no funcionales.
- Elaboración del diagrama de casos de uso.
- Descripción de los casos de uso.
- Modelación de las clases del sistema.
- Utilización del servicio de conversión a diferentes formatos de la plataforma alasGRATO.
- Implementación del Plug-in para la primera versión de la plataforma alasGRATO.
- Implementación del Plug-in para la segunda versión de la plataforma alasGRATO.
- Realización de las pruebas funcionales al módulo desarrollado.

A continuación se presenta el resumen de cada uno de los cuatro capítulos que presenta el trabajo.

Capítulo 1: Fundamentación Teórica. En este capítulo se presenta el marco teórico estudiado y seleccionado para desarrollar el objeto de estudio. Se presenta una panorámica sobre los conceptos y herramientas necesarios para alcanzar los objetivos planteados.

Capítulo 2: Características del Sistema. Se realiza una breve descripción de la solución propuesta, sus requisitos funcionales y no funcionales, así como los actores que intervienen en ella. Se presenta el modelo de dominio del sistema, el diagrama de caso de usos del sistema y la descripción de los mismos.

Capítulo 3: Diseño del Sistema. Se describe la representación arquitectónica del sistema. Se hace énfasis en los patrones de arquitectura utilizados, así como los patrones de diseño que fueron empleados. Se abordan los temas de arquitectura informacional, usabilidad y se lleva a cabo el diseño del sistema.

Capítulo 4: Describe la implementación del sistema, mostrando los diagramas de componentes. Además se presenta todo lo relacionado con las pruebas realizadas al sistema.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Representación Molecular

El dinamismo y funcionalidad de la representación molecular es planteado por Hoffman y Lazlo [1] en términos del grado ascendente de la complejidad molecular, lo reconocen como una transformación simbólica de la realidad, de carácter gráfico y lingüístico, histórico, artístico y científico. Un primer nivel, referido a la estructura molecular como grafos 2D y 3D, en términos de elementos y valencia. Un segundo, vinculado a la representación como geometría molecular, ángulos y longitudes de enlace son definidos para crear una unidad rígida. Un tercer nivel, centrado en los métodos "ab initio" de la química cuántica.

En una dirección similar, Schummer [2] reconoce como núcleo químico de la química a las sustancias y a las fórmulas estructurales, para diferenciar la estructura conceptual básica de la química de otras estructuras conceptuales que actúan en los niveles físico-químico y químico-biológico. "*Trabajando con sustancias y pensando con fórmulas estructurales*" es la idea relevante planteada por este autor para referirse a las fórmulas estructurales (grafos moleculares), como representaciones químicas apropiadas para realizar la actividad mental inferencial acerca de las propiedades químicas de las sustancias.

Este químico concibe las representaciones con un aspecto metafórico o pictográfico, a los cuales subyacen fuertes sistemas teóricos para interpretar, explicar y predecir sobre los comportamientos de las sustancias en contexto. Concibe a las fórmulas estructurales en las cuales se identifican elementos y grupos funcionales, y la conectividad entre estos, representaciones formalizadas en 2D y 3D. De este modo, las fórmulas estructurales son una primera aproximación molecular, dinámica y funcional, que representa de modo sistemático relaciones químicas en una red de clase de sustancias o de clases de sustancias.

El progreso de la representación molecular en términos de fórmulas estructurales ha crecido a la luz de la lógica de las identidades y diferencias de las partes estructurales de las fórmulas desarrolladas, como representaciones de la identidad y la diferencia química entre dos o más sustancias, apoyada en la topología química y la teoría de grupos. Ha creado conjuntos de reglas para comprender, interpretar y transformar fórmulas estructurales, lo cual ha producido un poderoso instrumento de sistematización y capacidad predictiva. En esta perspectiva la fórmula estructural es denominada grafo topológico y le subyace una teoría según la clase o clases de grupos funcionales en la organización estructural. En esta perspectiva, las fórmulas estructurales son la representación adecuada para realizar predicciones acerca

de las propiedades químicas. Es la manera como -hasta el presente- se han derivado de modo sistemático las transformaciones químicas.

El concepto de flexibilidad de la estructura molecular fundamentado en el pensamiento topológico para sustentar y definir la diversidad de configuraciones de una fórmula estructural, reafirmar y potenciar los conceptos de isomorfismo y homeomorfismo de la representación, contribuir al establecimiento de sistemas de nomenclatura química y a la toma de decisiones con respecto a la estructura de modo coherente con la información química es planteado y argumentado por Turro [3] en 1986, Klein [4] en 1992, Mihalić y Trinajstić [5] en 1992, Randić [6] en 1992 y Eckroth [7] en 1993. La idea de grafo molecular se impone como una representación de moléculas en geometría topológica.

Desde el siglo XIX, cuando se instaura en la química la teoría estructural, las moléculas se diferencian por la disposición de los elementos en una armazón de enlaces de acuerdo a la valencia, correspondiendo a cada organización un conjunto de características químicas.

Turro plantea la creación por parte de los químicos de un procedimiento general con una estructuración intelectual denominada pensamiento topológico. Éste, ha progresado al desarrollar métodos de mapeo de fórmulas estructurales, con el propósito de establecer una correspondencia entre un modelo, expresado de modo topológico, y un fenómeno u observación.

La topología química enfatiza metodologías de mapeo flexible de las fórmulas estructurales, como grafos topológicos, con la característica de ser dinámicos y recursivos, proveer una amplia red conceptual para el seguimiento de eventos químicos con importante éxito en la química orgánica y de coordinación y permitir, la categorización y representación de un buen número de sistemas químicos con base a unas reglas mínimas y fácilmente manipulables. A diferencia del grafo molecular topológico, la forma geométrica es estática y rígida en el correspondiente espacio 2D o 3D, caracterizada por longitudes y ángulos de enlace, cualidades ausentes en el grafo topológico.

Un grafo matemático es un conjunto de puntos y de conectivas entre los puntos lo cual origina una organización estructural, ésta sufre transformaciones sin violentar las conectivas entre puntos y se genera de este modo, a través del proceso, un conjunto de estructuras idénticas y de apariencia disímil. El grafo matemático, es transformado en un objeto químico conceptual, al significar el conjunto de puntos como una colección de elementos químicos y/o funciones químicas, conectados según las reglas de valencia de cada elemento. La valencia u orden de enlace, es el concepto topológico fundamental a partir del cual se realizaron las primeras formulaciones de la teoría estructural de Butlerov en 1861, Van't Hoff. y Lebel en

1874, Kekulé y Couper en 1865, Werner en 1891. Butlerov introdujo el término estructura química para definir una molécula compleja, a partir del conocimiento de su comportamiento químico, como determinada por la naturaleza, cantidad y estructura química de las partículas constituyentes Kluge, Larder [8].

En topología química, un grafo molecular es un conjunto de elementos (núcleos atómicos) y/o grupos funcionales (grupos de elementos), los cuales representan los puntos y de segmentos conectivos que representan enlaces químicos; elementos y enlaces constituyen una pluralidad de estructuras moleculares topológicas. El conjunto de elementos y/o grupos funcionales representa la composición del grafo y la red de enlaces la constitución. Composición y constitución son propiedades del grafo molecular, ampliadas a dos categorías más, configuración y conformación, construidas históricamente por los químicos en el proceso de búsqueda de hacer consistente el grafo molecular con los eventos químicos Turro [3]. Configuración que se refiere a la constitución transformada a un espacio euclidiano 3D y conformación a la propiedad de flexibilidad del grafo molecular.

Un modelo topológicamente correcto puede ser mapeado para todos aquellos fenómenos que se corresponden con los aspectos relevantes del modelo y estos atributos cualitativos ser transferidos a un amplio número de situaciones químicas.

Por eso Turro [3] considera el pensamiento topológico en química como uno de los elementos importantes en el progreso del conocimiento científico, afirma ser un pensamiento cualitativo que al integrarse al pensamiento cuantitativo constituye conocimiento del objeto o fenómeno. Sugiere que el pensamiento topológico libera la intuición de las restricciones del espacio tridimensional y es capaz de proveer una intuición más general y de mayor riqueza para el examen de los fenómenos microscópicos.

1.2 Edición Molecular

Al igual que la visualización de las moléculas, editar moléculas es un área de vital importancia dentro de la Bioinformática para los investigadores, y que ha tenido gran difusión en los últimos tiempos. Es de gran ayuda para los especialistas poder representar, realizarles pruebas, cálculos y ensayos teóricos a los compuestos antes de sintetizarlos.

La modelación "in sílico" se está convirtiendo en un enfoque esencial para el descubrimiento de fármacos optimizados en cuanto a su actividad biológica, efectos colaterales, entre otros. Su capacidad para

descartar avances falsos en las etapas iniciales del descubrimiento de fármacos utilizando la modelación y la informática ahorra costos de oportunidad y dólares. [9]

Aunque la visualización en tres dimensiones ofrece mucha más información a los especialistas, editar moléculas de esta forma resulta un proceso algo largo y complejo. Debido a esto la manera más cómoda y menos engorrosa de editar moléculas es en dos dimensiones. A partir de la representación del compuesto en dos dimensiones es posible visualizarlo en tres dimensiones.

Desde hace algunos años se han venido desarrollando algunas aplicaciones para la edición de moléculas en dos dimensiones, las cuales reciben el nombre de Editores Moleculares. En la actualidad existen varios editores moleculares, los cuales han ido perfeccionando los métodos de representación de las moléculas. Debido a las ventajas que proporcionan, estas aplicaciones se han vuelto una funcionalidad indispensable en las herramientas de laboratorios virtuales para modelación de fármacos.

1.3 Editores Moleculares a nivel mundial

1.3.1 ACD/ChemSketch Freeware

ACD/ChemSketch Freeware [10], versión gratuita de ACD/ChemSketch, es un paquete de dibujo que permite dibujar estructuras químicas, incluyendo estructuras orgánicas, compuestos organometálicos, polímeros, y las estructuras de Markush. También incluye características tales como el cálculo de las propiedades moleculares (por ejemplo, peso molecular, densidad, refractividad molar, etc.), limpieza y visualización de estructuras 2D y 3D y nombrar estructuras (menos de 50 átomos y 3 anillos). Esta versión gratuita de ChemSketch no incluye toda la funcionalidad de la versión comercial, ya que es una versión gratuita que ayuda en la enseñanza de conceptos clave de la química a la escuela secundaria, pregrado y estudiantes de química de postgrado.

Este software no cumple con los requerimientos que posee la plataforma pues no contiene todas las funcionalidades necesarias requeridas ni cumple con las preferencias de visualización.

1.3.2 ACD/ChemSketch

ACD/ChemSketch [11] es una interfaz de dibujo químicamente inteligente que permite dibujar casi cualquier estructura química, incluyendo estructuras orgánicas, compuestos organometálicos, polímeros, y

las estructuras de Markush. Este se utiliza para producir estructuras de aspecto profesional y diagramas para informes y publicaciones. Entre sus principales características están las de dibujar y visualizar estructuras en 2D o reproducirlas en 3D para permitir observarlas desde cualquier ángulo, dibujar reacciones y sistemas de reacción, calcular las cantidades de los reactivos, generar estructuras a partir de cadenas InChI y SMILES y generar nombres IUPAC para moléculas de hasta 50 átomos y estructuras de 3 anillos. ACD/ChemSketch también permite comprobar otras formas tautoméricas de su estructura dibujada, lo cual es importante en la búsqueda de estructuras, en las predicciones etc.

Este software posee muchas funcionalidades que podrían servir para la plataforma pero es privativo, por lo que no se puede utilizar. Además las preferencias de visualización no son las requeridas por la plataforma.

1.3.3. ChemDraw Ultra 12.0 Suite

ChemDrawUltra [12] es un paquete excepcional para dibujar y editar estructuras químicas. Una de las características que posee es la Configuración de Documento, la cual da a este paquete una ventaja en la preparación de sistemas químicos para preparar los manuscritos que serán publicados. Los gráficos preparados en ChemDrawUltra aseguran un aspecto profesional y limpio.

ChemDrawUltra le permite importar una gran variedad de formatos de imagen que se pueden editar o modificar, según sea necesario, incluyendo LMC, GIF, BMP, JPEG, ISIS, MDL, TIF, Galactic y archivos espectros Jcamp. Esta capacidad de ChemDrawUltra evita tener que crear e importar o exportar las imágenes nuevas o varios sistemas de gráficos para crear o editar un dibujo existente.

Este software posee muchas funcionalidades que podrían servir para la plataforma pero es privativo, por lo que no se puede utilizar. Además las preferencias de visualización no son las requeridas por la plataforma.

1.3.4 MarvinSketch

MarvinSketch [13] es un editor avanzado de química, basado en Java para dibujar estructuras químicas, consultas y reacciones. Tiene una rica y creciente lista de funciones de edición y es químicamente consciente. Entre otras se puede citar que soporta un amplio rango de tipos de archivos como son: MOL, MOL2, SDF, RXN, RDF (V2000/V3000), SMILES, SMARTS/SMIRKS (recursivo), MRV, InChi, CML, PDB etc., opciones de copiar y pegar entre diferentes editores, documentos de varias páginas y soporte de impresión, precargado de estructuras de plantillas, comprobación de errores de valencia y de reacción,

funciones de estereoquímica avanzada. MarvinSketch, al estar basado en Java, se puede ejecutar en todos los principales sistemas operativos utilizados en el mundo.
Este software posee muchas funcionalidades que podrían servir para la plataforma pero no cumple con las preferencias de visualización requeridas por la plataforma.

1.3.5 HyperChem Professional 8.0

HyperChem [14] es un sofisticado entorno de modelado molecular que es conocido por su calidad, flexibilidad y facilidad de uso. Entre sus características se encuentran seleccionar, rotar, traducir y cambiar el tamaño de las estructuras con el ratón, reemplazar cualquier átomo de hidrógeno seleccionado con una variedad de sustituyentes, incluyendo sustituyentes personalizados, aplicar restricciones de construcción fácilmente: especificar la longitud de enlace, ángulos de enlace, ángulos de torsión, o la geometría de unión alrededor de un átomo seleccionado, especificar el tipo de átomo, la carga del átomo, la carga formal y de masa atómica y mutar residuos y construir grandes moléculas de forma incremental.
Este software posee muchas funcionalidades que podrían servir para la plataforma pero no cumple con las preferencias de visualización requeridas por la plataforma además de ser privativo.

1.4 Metodología y herramientas

1.4.1 Metodología: Open UP

Open UP [15] es un proceso unificado de desarrollo que aplica acercamientos iterativos e incrementales dentro de un ciclo de vida estructurado. Además abraza una filosofía pragmática, ágil, y enfocada en la naturaleza de colaboración del desarrollo del software. Es una herramienta que se puede ampliar a una gran variedad de tipos de proyecto.
El esfuerzo personal en un proyecto de Open UP se organiza en micro-incrementos, los cuales representan unidades cortas de trabajo que producen un paso constante y mensurable en el progreso del proyecto. El proceso aplica la colaboración intensiva como sistema de desarrollo incremental. Estos micro-incrementos proporcionan un lazo de regeneración extremadamente corto, que conducen a decisiones adaptables dentro de cada iteración. El ciclo de vida de un proyecto en Open UP está estructurado en cuatro fases: inicio, elaboración, construcción y transición.

1.4.2 Lenguaje de modelado: UML

El Lenguaje de Modelado Unificado (UML - Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un producto de software que responde a un enfoque orientado a objetos. Este lenguaje fue creado por un grupo de estudiosos de la Ingeniería de Software formado por: Ivar Jacobson, Grady Booch y James Rumbaugh en el año 1995. Desde entonces, se ha convertido en el estándar internacional para definir, organizar y visualizar los elementos que configuran la arquitectura de una aplicación orientada a objetos.

UML no es un lenguaje de programación, sino un lenguaje de propósito general para el modelado orientado a objetos. También puede considerarse como un lenguaje de modelado visual que permite una abstracción del sistema y sus componentes.

1.4.3 Herramienta CASE: Visual Paradigm

Como herramienta CASE se empleó Visual Paradigm para el trabajo con UML. La herramienta está diseñada para una amplia gama de usuarios que incluye a: ingenieros de software, analistas de sistemas, analistas de negocios y arquitectos de sistemas, interesados en la creación de Grandes Sistemas de Software de manera confiable, a través del paradigma Orientado a Objetos. VP-UML soporta los últimos estándares de anotaciones de JAVA y UML y provee soporte para la generación de código y la ingeniería inversa para Java. Además, VP-UML se integra con Eclipse, Borland® JBuilder®, NetBeans IDE/Sun™ ONE, IntelliJ IDEA™, Oracle JDeveloper y BEA WebLogic Workshop™, para soportar las fases de implementación en el desarrollo de software. Las transiciones del análisis al diseño, y de éste a la implementación, están adecuadamente integradas dentro de la herramienta CASE, de manera que reduce significativamente los esfuerzos de todas las etapas del ciclo de desarrollo de software.

1.4.4 Lenguaje de programación: Java

El lenguaje de programación Java es un lenguaje de propósito general, concurrente, basado en clases y orientado a objetos. Su diseño fue concebido para que los programadores puedan lograr fluidez con el lenguaje. Java ofrece toda la funcionalidad de un lenguaje potente, pero sin las características menos usadas y más confusas de éstos. C++ es un lenguaje que adolece de falta de seguridad, pero C y C++ son lenguajes más difundidos, por ello, Java se diseñó para ser parecido a C++ y así facilitar un rápido y fácil aprendizaje, además, el mismo elimina muchas de las características de otros lenguajes como C++,

para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el garbage collector (reciclador de memoria dinámica o recolector de basura). No es necesario preocuparse de liberar memoria, el reciclador se encarga de ello y como es un thread (hilo) de baja prioridad, cuando entra en acción, permite liberar bloques de memoria muy grandes, lo que reduce la fragmentación de esta.
Una de las características más importante de Java es que posee una arquitectura neutral, es decir el compilador Java compila su código a un fichero objeto de formato independiente de la arquitectura de la máquina en que se ejecutará. Cualquier máquina que tenga el sistema de ejecución (run-time) puede ejecutar ese código objeto, sin importar en modo alguno la máquina en que ha sido generado.
Actualmente existen sistemas run-time para Solaris 2.x, SunOs 4.1.x, Windows 95, Windows NT, Linux, Irix, Aix, Mac, Apple y probablemente haya grupos de desarrollo trabajando para la portabilidad a otras plataformas.
Más allá de la portabilidad básica por ser de arquitectura independiente, Java implementa otros estándares de portabilidad para facilitar el desarrollo. Los enteros son siempre enteros y además, enteros de 32 bits en complemento a 2. Java, además, construye sus interfaces de usuario a través de un sistema abstracto de ventanas, de forma que ellas puedan ser implantadas en entornos Unix, Pc o Mac.

1.4.5 Entornos de desarrollo: Eclipse

Eclipse es uno de los más potentes entornos integrados de desarrollo (IDE) que permite la construcción de aplicaciones en Java. Admite la incorporación de otros plug-ins para obtener un mayor número de funcionalidades, es una herramienta de código abierto, multiplataforma y compatible con los sistemas operativos más utilizados en el mundo. Además posee la capacidad de ser soportado para distintas arquitecturas, control de versiones con cvs o con subversión, resaltado de sintaxis, un potente refactorizado de código, asistentes (wizards): para la creación, exportación e importación de proyectos; permite la integración con la herramienta CASE (Computer Aided Software Engineering, Ingeniería de Software Asistida por computadoras) Visual Paradigm.

CAPÍTULO 2: CARACTERÍSTICAS DEL SISTEMA

2.1 Introducción

En este capítulo se presenta una breve descripción de la solución propuesta, sus requisitos funcionales y no funcionales. Se presenta además el diagrama de casos de uso del sistema, junto con su descripción, y el Modelo de Dominio.

2.2 Breve Descripción del Sistema

El diseño del Editor Molecular permitirá a los usuarios la representación gráfica en 2D de moléculas, con múltiples opciones que facilitarán la construcción y análisis de compuestos orgánicos.

2.3 Modelo de Dominio

En el Modelo de Dominio se representan los conceptos u objetos más importantes dentro del contexto del sistema. Su objetivo fundamental es describir las clases más importantes del sistema. Representa conceptos del mundo real, no de componentes de software. En la figura 2.1 se muestra el diagrama de clases del modelo de dominio, donde se aprecia como el Especialista interactuará con el Editor Molecular, el cual a su vez interactuará con los Compuestos Orgánicos descritos en su totalidad por la información estructural, que es conocida de antemano por el usuario final de la plataforma.

A continuación se describen cada una de las clases que componen del Modelo de Dominio:

Especialista: Persona con los conocimientos requeridos para interactuar con la aplicación.

Editor Molecular: Plug-in utilizado para la edición de los compuestos orgánicos de 2D.

Compuesto Orgánico: Sustancia química basada en cadenas de carbono e hidrógenos. En muchos casos contiene oxígenos, nitrógenos, azufres, fósforos y halógenos.

Información Estructural: Dato o descripción que permite conocer cómo está formado el compuesto en términos de átomos y la relación espacial que existe entre ellos.

Imagen del Compuesto: Representación bidimensional del compuesto; se tiene en cuenta la geometría espacial del mismo.

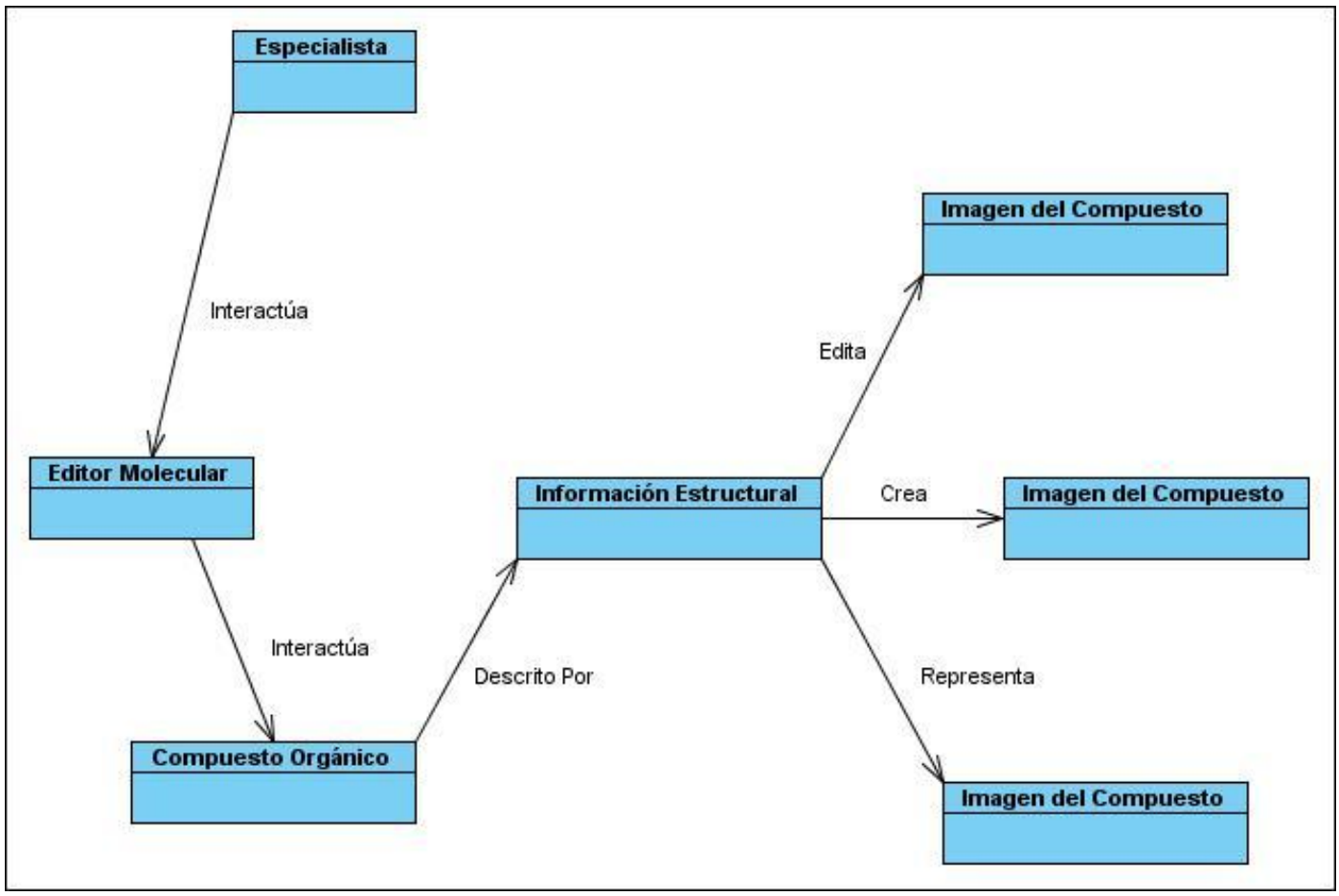


Fig. 2.1 Diagrama de clases del Modelo del Dominio

2.4 Especificación de los Requisitos del Sistema

Los requisitos son capacidades o condiciones que debe ser alcanzada o poseída por un sistema o un componente de un sistema para satisfacer las necesidades de un determinado cliente, contrato o estándar.

2.4.1 Requisitos Funcionales

Los requisitos funcionales de un sistema representan las funcionalidades que este debe implementar, los servicios que proveerá, entre ellos: sus entradas, salidas y excepciones. A continuación se muestran los requisitos funcionales del Editor Molecular:

RF 1: Gestionar Molécula.

RF 1.1: Crear Molécula.
RF 1.2: Modificar Molécula.
RF 1.3: Guardar Molécula.
RF 1.4: Garantizar Hibridación.

RF 2: Dibujar Plantillas.

RF 2.1: Dibujar Ciclo de tres.
RF 2.2: Dibujar Ciclo de cuatro.
RF2.3: Dibujar Ciclo de cinco.
RF 2.4: Dibujar Ciclo de seis.
RF 2.5: Dibujar Ciclo de benceno.
RF 2.6: Dibujar Ciclo de siete.
RF 2.7: Dibujar Ciclo de ocho.

RF 3: Dibujar Enlaces.

RF 3.1: Dibujar Enlace Simple.
RF 3.2: Dibujar Enlace Doble.
RF 3.3: Dibujar Enlace Triple.

RF 4: Eliminar Estructura.

RF 4.1: Eliminar Enlaces o Ciclos.
RF 4.2: Limpiar área de trabajo.

RF 5: Gestionar Átomos.

RF 5.1: Cambiar Átomos.

RF 5.2: Enumerar los Átomos.

RF 5.3: Poner Símbolos a los Átomos.

RF 6: Cambiar Forma de Edición.

RF 6.1: Editar en Bolas y Cilindros.

RF 6.2: Editar en Alambre.

RF 7: Aplicar escala.

RF 7.1: Ampliar Molécula.

RF 7.2: Reducir Molécula.

2.4.2 Requisitos No Funcionales

Los requisitos no funcionales son las propiedades que debe cumplir un sistema, como son la fiabilidad, el tiempo de respuesta, la capacidad de almacenamiento, la capacidad de los dispositivos de entrada/salida, y la representación de datos que se utiliza en las interfaces del sistema.

A continuación se muestran los requisitos no funcionales definidos para la creación del Editor Molecular:

2.4.2.1 Usabilidad

La aplicación está diseñada para ser usada por usuarios con conocimientos de química y para especialistas del campo de diseño de fármacos.

2.4.2.2 Soporte

El sistema debe propiciar su mejoramiento y la incorporación de otras opciones en un futuro.

2.4.2.3 Software

Se debe disponer de sistemas operativos que cuenten con la instalación del Java Runtime Environment (JRE) versión 1.5 o superior.

2.4.2.4 Hardware

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

- Procesador Pentium 4, 2.0 GHz

- 512 MB de RAM
- 100 MB de capacidad del disco duro

2.4.2.5 Requisitos de Licencia

El sistema se utilizará primeramente en la Universidad de Camagüey. Posteriormente, en aquellos centros que -en el país-, se dediquen al diseño de fármacos. Deberá contarse con una licencia en el caso que se decida comercializarse en el extranjero.

2.4.2.6 Requisitos Legales, de Derecho de Autor y otros

Los derechos de autor serán registrados por la UCI.

2.5 Definición de los Casos de Uso del Sistema.

Los casos de uso, independientemente de la implementación, bajo la forma de acciones y reacciones describen el comportamiento del sistema desde el punto de vista del usuario. Son un medio para obtener información de cómo debe trabajar el sistema.

2.5.1 Actores del Sistema

Se le llama actor del sistema a cualquier entidad externa que interactúa con este y demanda alguna funcionalidad. Pueden ser operadores humanos o sistemas externos, así como entidades abstractas como el tiempo.

Actores	**Descripción**
Especialista	Representa al usuario que va a hacer uso del sistema, teniendo la posibilidad de interactuar con todas las funcionalidades de este.

2.5.2 Diagrama de Casos de Uso del Sistema

El diagrama de casos de uso del sistema es utilizado para especificar la comunicación y comportamiento del sistema mediante la interacción con los usuarios y/u otros sistemas. En la figura 2.2 se muestra el diagrama de casos de uso del sistema.

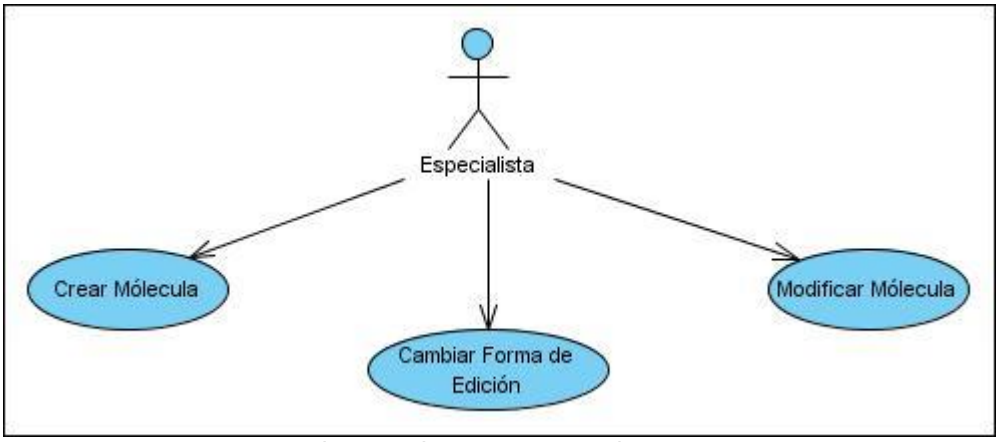


Fig. 2.2 Diagrama Casos de Uso

2.5.3 Descripción de los Casos de Uso del Sistema

Un caso de uso es una técnica utilizada para la captura de requisitos potenciales de un nuevo sistema de software. Cada uno proporciona uno o múltiples escenarios que muestran la forma en que debería interactuar el sistema con los usuarios u otros sistemas para alcanzar un objetivo antes trazado.

2.5.3.1 Descripción del caso de uso: Crear Molécula.

Caso de Uso:	Crear Molécula
Actores:	Especialista
Propósito:	Crear moléculas definidas por el especialista para su posterior uso en la aplicación.
Resumen:	El caso de uso se inicia cuando el especialista ejecuta el módulo Editor Molecular de la opción módulos del Visualizador alasGRATOViewer, sin tener ninguna molécula seleccionada en el árbol de ficheros cargados.
Referencia:	RF1.1, RF1.3, RF1.4, RF2, RF3, RF4, RF5.1
Precondiciones:	Que no existan ficheros seleccionados en el árbol de ficheros cargados del Visualizador.
Prioridad:	Crítico

Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**
	1. El sistema brinda la posibilidad al especialista de comenzar a crear la molécula en el área de trabajo.
2. Si el especialista decide comenzar a crear en el área de trabajo puede seleccionar una de las opciones de la barra de herramientas: • Plantillas. • Elemento Químico.	2.1 Si no existe un ciclo o un enlace creado el sistema crea la representación, según la opción escogida por el especialista. 2.2 En tiempo de ejecución el sistema da la posibilidad de "Eliminar" y el sistema eliminará lo que el especialista seleccione.
3. El especialista selecciona la opción "Guardar Molécula" de la Barra de Herramientas al haber concluido su trabajo.	3.1 El sistema muestra un explorador donde podrá seleccionar el lugar y nombre del fichero donde va a almacenar la molécula.
4. El especialista selecciona el lugar y nombre que tendrá el fichero molecular y presiona Aceptar.	4.1 El sistema muestra una ventana con el siguiente mensaje ¿Desea optimizar la molécula? Si o No.
5. El especialista selecciona la opción Si	5.1 El sistema guarda la molécula en 3D y la adiciona a árbol de ficheros cargados del Visualizador alasGRATOViewer.
Flujo Alternativo.	
Acción del Actor	**Respuesta del Sistema**

	2.1.- Si existe un ciclo o un enlace creado el sistema debe realizar lo siguiente para añadir uno nuevo: • Si se tiene seleccionado un vértice el sistema debe dibujar la opción seleccionada. • Si se tiene seleccionado una arista el sistema debe tomar la arista en común y dibuja la opción seleccionada.
4. El especialista selecciona Cancelar.	4.1 Se retorna al flujo normal de eventos 3.1.
5. El especialista selecciona la opción No.	5.1 El sistema guarda la molécula en 2D y la adiciona a árbol de ficheros cargados del Visualizador alasGRATOViewer.
Poscondiciones:	Una nueva molécula es creada, guardada y añadida al árbol de moléculas cargadas a la aplicación.

2.5.3.2 Descripción del caso de uso: Modificar Molécula.

Caso de Uso:	Modificar Molécula
Actores:	Especialista
Propósito:	Cambiar la estructura de una molécula cargada en el sistema. La misma será utilizada en los procesos de la aplicación.
Resumen:	El caso de uso se inicia cuando el especialista ejecuta el módulo Editor Molecular de la opción módulos del Visualizador alasGRATOViewer, teniendo una molécula seleccionada en el árbol de ficheros cargados. En el área de trabajo aparecerá la molécula dibujada.
Referencia:	RF1.2, RF1.3, RF1.4, RF2, RF3, RF4, RF5.1
Precondiciones:	Que la molécula esté seleccionada en el árbol de ficheros cargados del Visualizador.
Prioridad:	Crítico
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. Si el especialista decide comenzar a modificar la molécula presente en el área de trabajo puede seleccionar una de las opciones de la barra de herramientas: • Plantillas • Elementos Químicos	1.1 Si el especialista selecciona una plantilla y seguidamente selecciona un vértice el sistema debe dibujar la opción seleccionada, tomando el vértice como común si la hibridación del vértice lo permite. 1.2 Si el especialista selecciona un elemento y seguidamente selecciona un vértice el sistema debe cambiar el elemento del vértice por el seleccionado, si la hibridación del vértice lo permite. 1.3 En tiempo de ejecución el sistema da la posibilidad de "Eliminar fragmento" o "Limpiar el área de trabajo": • Si el especialista seleccionó el icono "Eliminar fragmento" de la barra de herramienta el sistema eliminará el fragmento que el especialista seleccione. • Si el especialista seleccionó el icono "Limpiar Área de Trabajo" de la barra de herramienta el sistema limpiará el área de trabajo, dando la posibilidad de crear una nueva molécula
2. El especialista selecciona la opción "Guardar Molécula" de la Barra de Herramientas al haber concluido su trabajo.	2.1 El sistema muestra una ventana con el siguiente mensaje ¿Desea optimizar la molécula? Sí o No.
3. El especialista selecciona la opción Sí.	3.1 El sistema guarda los cambios de la molécula editada en 3D y actualiza los datos de la existente en el árbol de ficheros cargados del Visualizador
Flujo Alternativo.	

Acción del Actor	**Respuesta del Sistema**
	1.1 Si se tiene seleccionado una arista el sistema debe tomar la arista en común y dibujar la opción seleccionada, si la hibridación lo permite. 1.2 Si se tiene seleccionado una arista el sistema no debe realizar nada.
2. El especialista selecciona Cancelar.	2.1 Se retorna al flujo normal de eventos 1.1.
5. El especialista selecciona la opción No.	3.1 El sistema guarda los cambios de la molécula editada en 2D y actualiza los datos de la existente en el árbol de ficheros cargados del Visualizador.
Poscondiciones:	La molécula es modificada, guardada y se actualizan los datos de la existente en al árbol de ficheros cargados a la aplicación.

2.5.3.3 Descripción del caso de uso: Cambiar Forma de Edición.

Caso de Uso:	Cambiar Forma de Edición
Actores:	Especialista
Propósito:	Cambiar la forma de edición de la molécula que se esté editando o creando.
Resumen:	El caso de uso se inicia cuando el especialista selecciona la opción de cambiar la forma de edición presente en la barra de herramientas.
Referencia:	RF5, RF6, RF7
Precondiciones:	Que se esté editando o creando una molécula.
Prioridad:	Secundario
Flujo Normal de Eventos	
Acción del Actor	**Respuesta del Sistema**

1. El especialista puede seleccionar una de las siguientes opciones de la barra de herramientas: • Editar en bolas y cilindros • Editar en alambre • Mostrar número de los átomos • Mostrar símbolos de los átomos • No mostrar número ni símbolos de los átomos • Maximizar la molécula • Minimizar la molécula	1.1 El sistema cambia la forma de edición de la molécula según la opción que haya sido seleccionada por el especialista.
Poscondiciones:	La forma de edición de la molécula cambiará de acuerdo a la opción seleccionada por el especialista.

2.6 Conclusiones

- Se definieron de manera precisa los requisitos que debe cumplir el sistema, y fueron seleccionados los actores y los casos de uso de dicho sistema.
- A través de los modelos de casos de uso se sentaron las bases para el desarrollo de las restantes fases del proceso de diseño e implementación. Y se ganó claridad en cuanto a la concesión del sistema a construir.
- Se relacionaron los conceptos definidos mediante el diagrama de Modelo de Dominio y se reflejó de forma general como se desarrolla este proceso.

CAPÍTULO 3: DISEÑO DEL SISTEMA

3.1 Introducción

En este capítulo se aborda el diseño del mismo así como los patrones que se han utilizado en su realización.

3.2 Patrones

Un patrón es una solución a un problema en un contexto, codifica conocimiento específico acumulado por la experiencia en un dominio. Un sistema bien estructurado está lleno de patrones.

En el sistema se utilizan, fundamentalmente, dos categorías de patrones, estos son los patrones arquitecturales y los patrones de diseño.

Los patrones arquitecturales son aquellos que expresan un esquema organizativo estructural fundamental para sistemas software; mientras que los patrones de diseño expresan esquemas para definir estructuras de diseño (o sus relaciones) con las que construir sistemas software.

3.2.1 Patrón arquitectónico utilizado

A continuación, en la figura 3.1, se expone el patrón arquitectónico empleado.

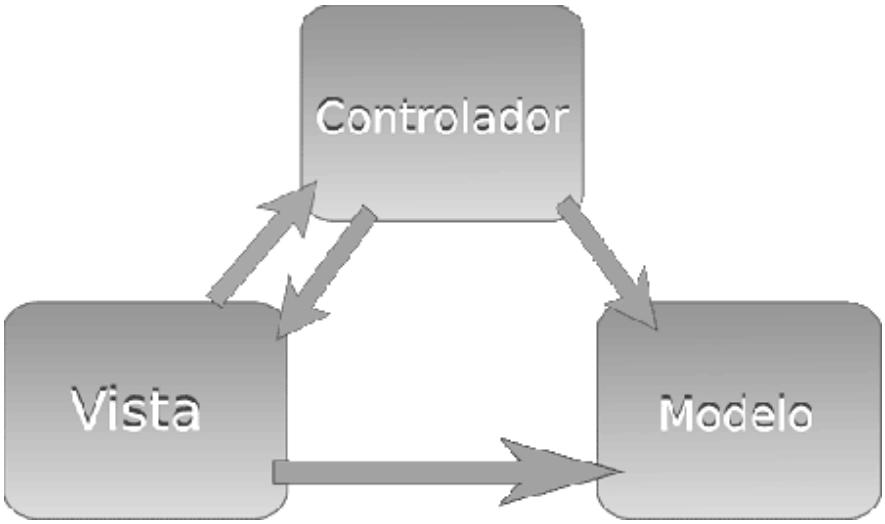


Fig. 3.1 Esquema del patrón Modelo-Vista-Controlador.

Modelo: Representa específicamente el dominio de la información sobre la cual funciona la aplicación. El modelo es otra forma de llamar a la capa de dominio. La lógica de dominio añade significado a los datos. El modelo encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.

Vista: Presenta el modelo en un formato adecuado para interactuar, usualmente con un elemento de interfaz de usuario. Muestra la información al usuario. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

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

En la siguiente figura se muestra como es aplicado este patrón en el sistema.

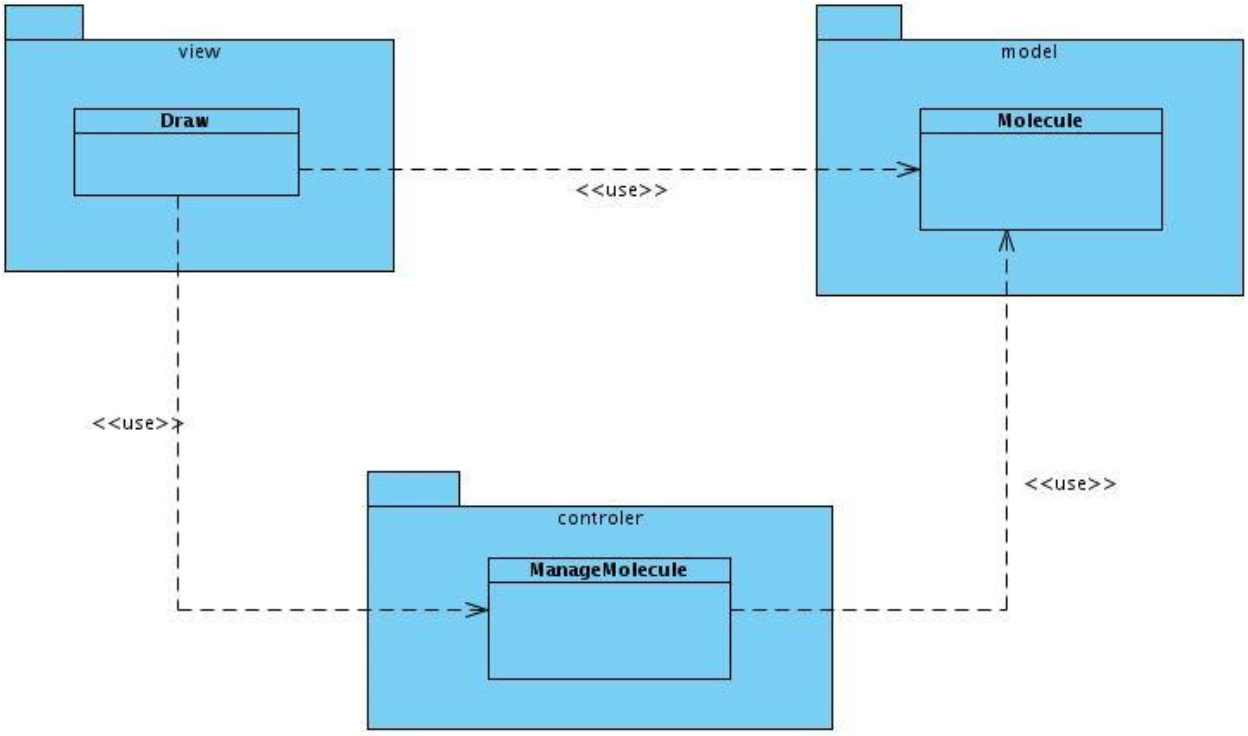


Fig. 3.2 Esquema del patrón Modelo-Vista-Controlador aplicado al sistema.

3.2.2 Patrones de diseño utilizados

A continuación se exponen los patrones de diseño utilizados.

3.2.2.1 Patrón Singleton

El patrón singleton permite asegurar que de una clase habrá solo una instancia, y proporciona un punto de acceso global a ella desde cualquier parte del código.
Tiene como ventajas que es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas (incluso si es variable) y también se reduce el espacio de nombres (frente al uso de variables globales).

3.2.2.2 Patrón Composite

El patrón composite permite tratar uniformemente a los objetos simples y compuestos de una estructura jerárquica recursiva.
Tiene como ventajas que simplifica notablemente al cliente, al no tener éste que distinguir entre objetos simples y compuestos y favorece la extensibilidad, ya que es muy fácil añadir nuevos tipos de componentes, tanto simples como compuestos.

3.2.2.3 Patrón Creador

El patrón creador guía la asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que se debe conectar con el objeto producido en cualquier evento.
Tiene como una de sus ventajas que da soporte al bajo acoplamiento.

3.2.2.4 Patrón Bajo Acoplamiento

El acoplamiento es una medida de la fuerza con que una clase está conectada a otras clases, con que las conoce y con que recurre a ellas. Acoplamiento bajo significa que una clase no depende de muchas clases.
Tiene como ventajas que los cambios de las clases afines no ocasionen cambios locales y sean fáciles de reutilizar puesto que no dependen de otras clases.

3.2.2.5 Patrón Alta Cohesión

La cohesión es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas que no realicen un trabajo enorme.
Tiene como ventajas que son fáciles de comprender, fácil de reutilizar y de conservar.

3.2.2.6 Patrón Controlador

El patrón controlador asigna la responsabilidad de manejar un sistema de mensajes a una clase representando una de las siguientes alternativas:

- Representar todo el sistema.
- Representar todos los negocios de la organización.
- Representar algo en el mundo real que es activo (por ejemplo el rol de una persona) que puede estar envuelta en la tarea.
- Representar un manejador artificial de todos los eventos del sistema.

Tiene como ventaja que incrementa el potencial de los elementos que pueden ser rehusados.

3.2.2.7 Patrón Experto

El patrón experto asigna una responsabilidad al experto en información, es decir, la clase que tiene la información necesaria para cumplir con la responsabilidad. El problema que resuelve el patrón experto está referido al principio más básico mediante el cual las responsabilidades son asignadas en el diseño orientado a objetos.
Tiene como ventajas que la encapsulación es mantenida, desde que los objetos usan sus propias informaciones para realizar tareas. Esto permite poco acoplamiento, lo cual conduce a sistemas más robustos y de mantenimiento mucho más fácil. La alta cohesión también es soportada.

3.3 Modelo de Diseño

El Modelo de Diseño es una abstracción del Modelo de Implementación y su código fuente, el cual se emplea, fundamentalmente, para representar y documentar su diseño. Es usado como entrada esencial en las actividades relacionadas con la implementación. Representa a los casos de uso en el dominio de la solución. Está compuesto por: Diagramas de Clases del Diseño, Diagrama de Despliegue y Diagramas de

Interacción (Colaboración y/o Secuencia) del Diseño. Estos últimos también llamados realización de casos de uso. A continuación se abordarán los temas relacionados con el plug-in Editor Molecular.

3.3.1 Diagramas de Clases

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

3.3.1.1 Diagrama de Clases del Caso de Uso Crear Molécula

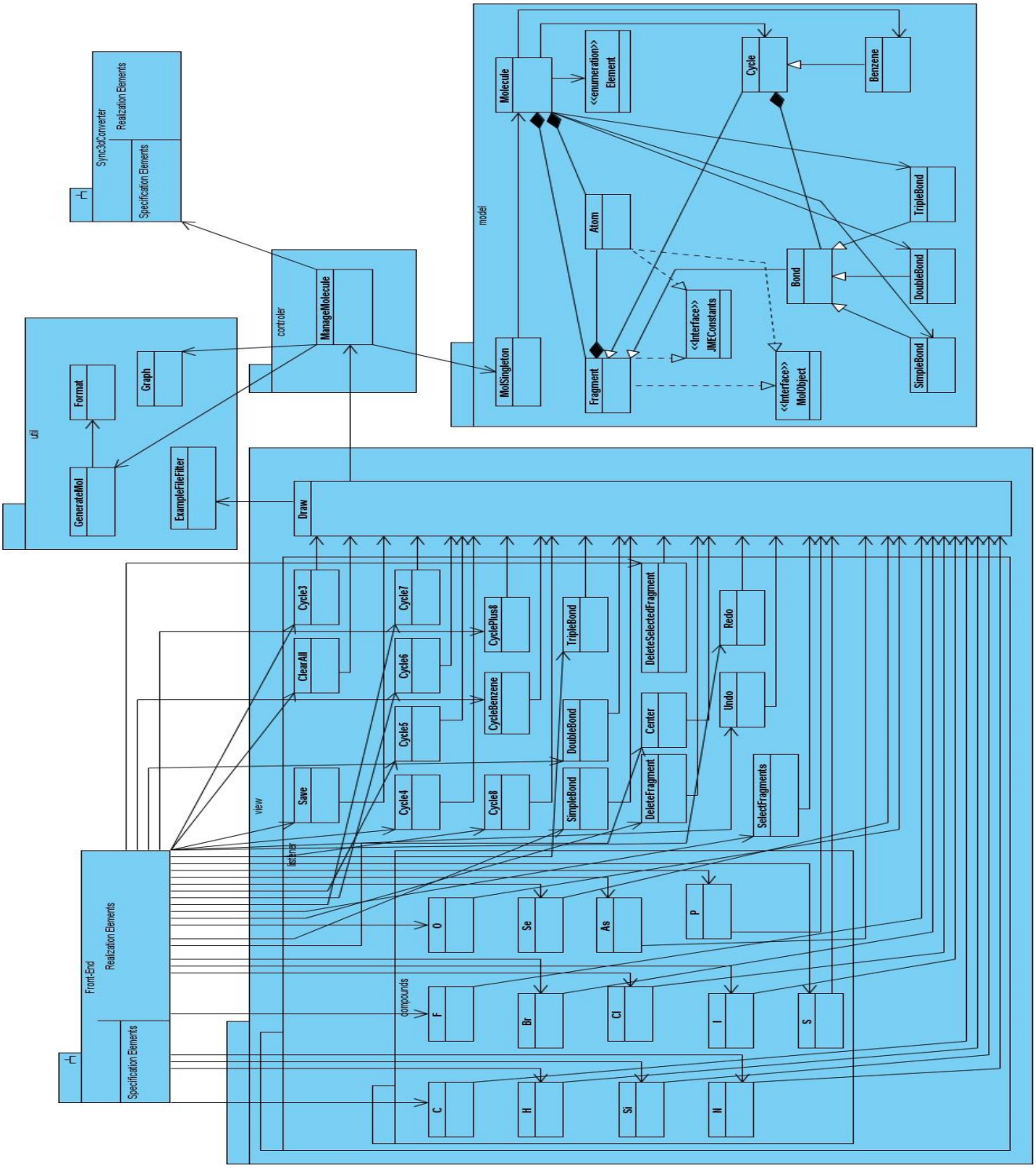


Fig. 3.3 Diagrama de Clases del Caso de Uso Crear Molécula.

3.3.1.2 Diagrama de Clases del Caso de Uso Modificar Molécula

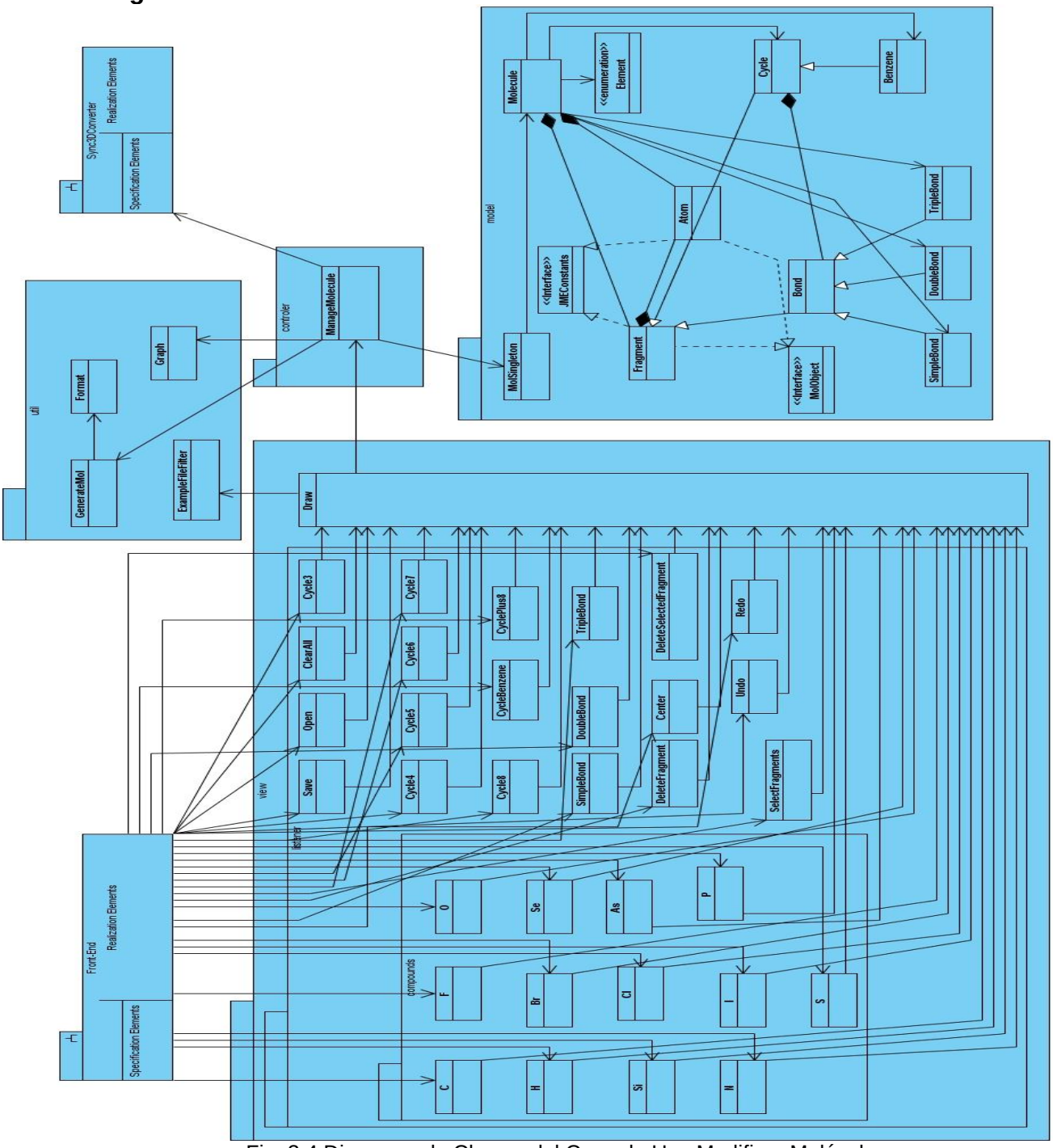


Fig. 3.4 Diagrama de Clases del Caso de Uso Modificar Molécula.

3.3.1.3 Diagrama de Clases del Caso de Uso Cambiar forma de edición

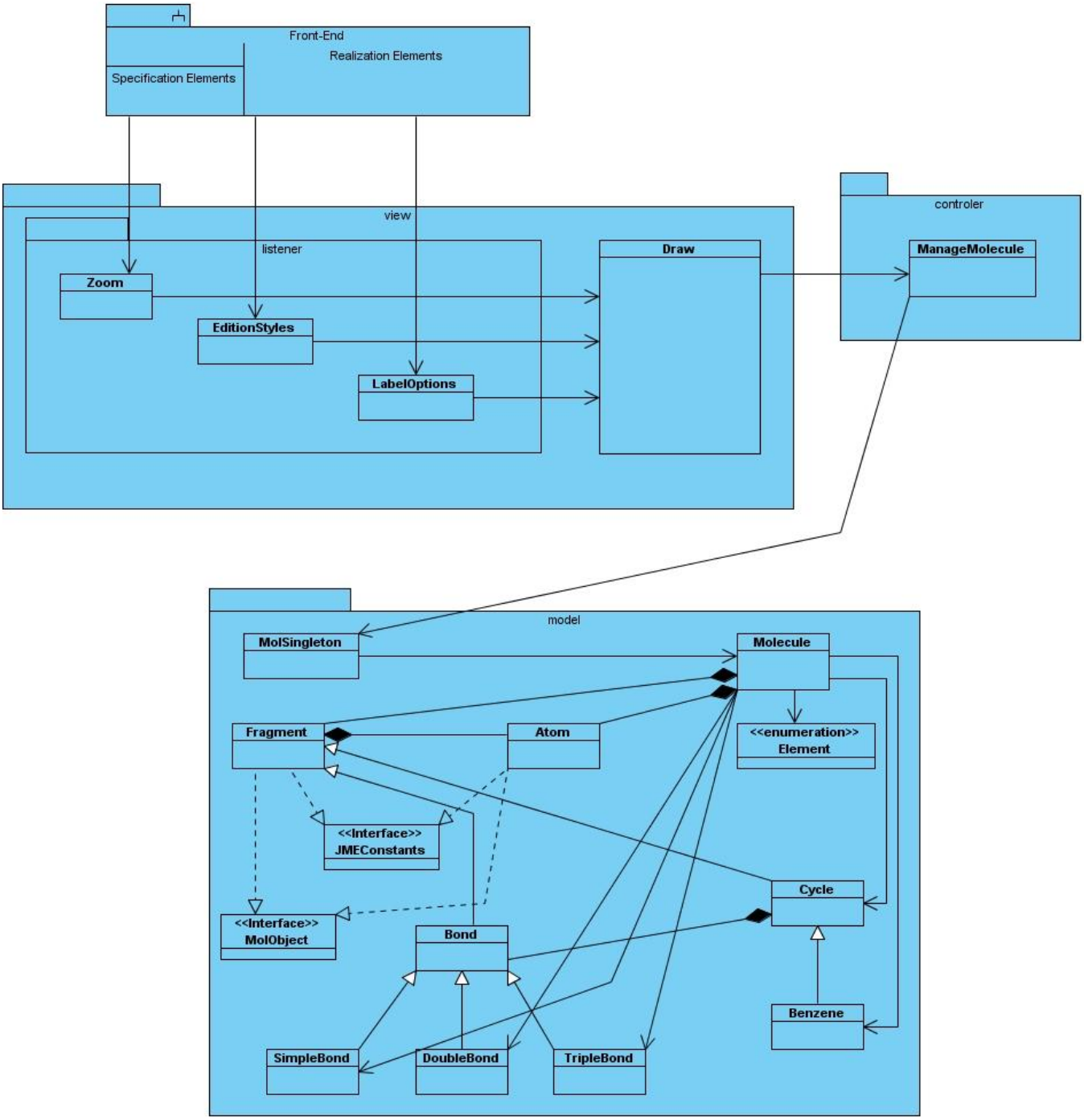


Fig. 3.5 Diagrama de Clases del Caso de Cambiar formas de edición.

3.3.2 Diagramas de Interacción

En los diagramas de interacción se muestra un patrón de interacción entre objetos. Hay dos tipos de diagrama de interacción, ambos basados en la misma información, pero cada uno enfatizando un aspecto particular: Diagramas de Secuencia y Diagramas de Colaboración.
A continuación se presentarán solamente los Diagramas de Secuencia.

3.3.2.1 Diagramas de Secuencia

Un Diagrama de Secuencia muestra una interacción ordenada según la secuencia temporal de eventos. En particular, muestra los objetos participantes en la interacción y los mensajes que intercambian ordenados según su secuencia en el tiempo. El eje vertical representa el tiempo, y en el eje horizontal se colocan los objetos y actores participantes en la interacción, sin un orden prefijado. Cada objeto o actor tiene una línea vertical, y los mensajes se representan mediante flechas entre los distintos objetos. El tiempo fluye de arriba hacia abajo. Se pueden colocar etiquetas (como restricciones de tiempo, descripciones de acciones etc.) bien en el margen izquierdo o bien junto a las transiciones o activaciones a las que se refieren.

3.3.2.1.1 Diagrama de Secuencia del Caso de Uso Crear Molécula

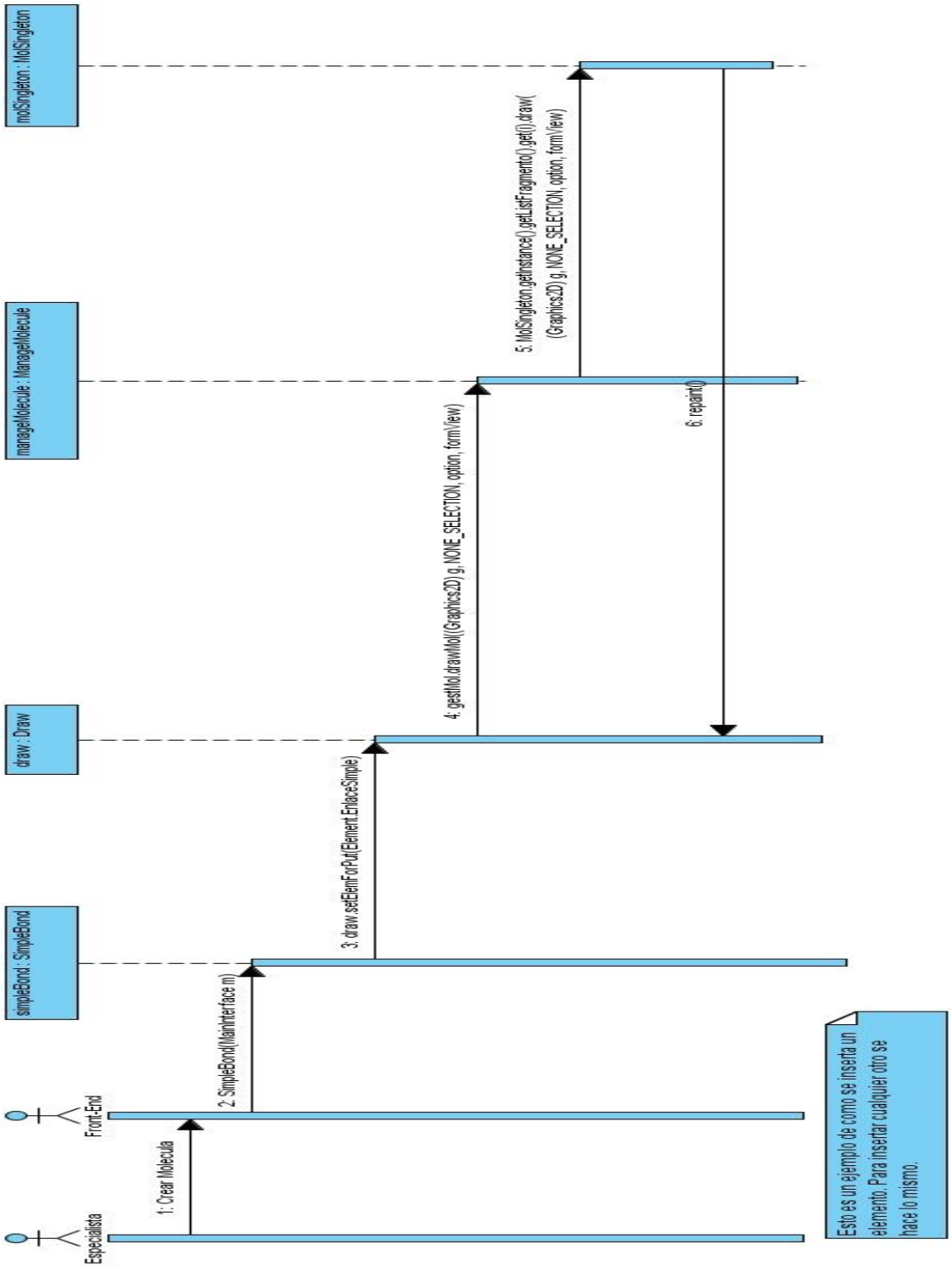


Fig. 3.6 Diagrama de Secuencia del Caso de Uso Crear Molécula

3.3.2.1.2 Diagrama de Secuencia del Caso de Uso Modificar Molécula

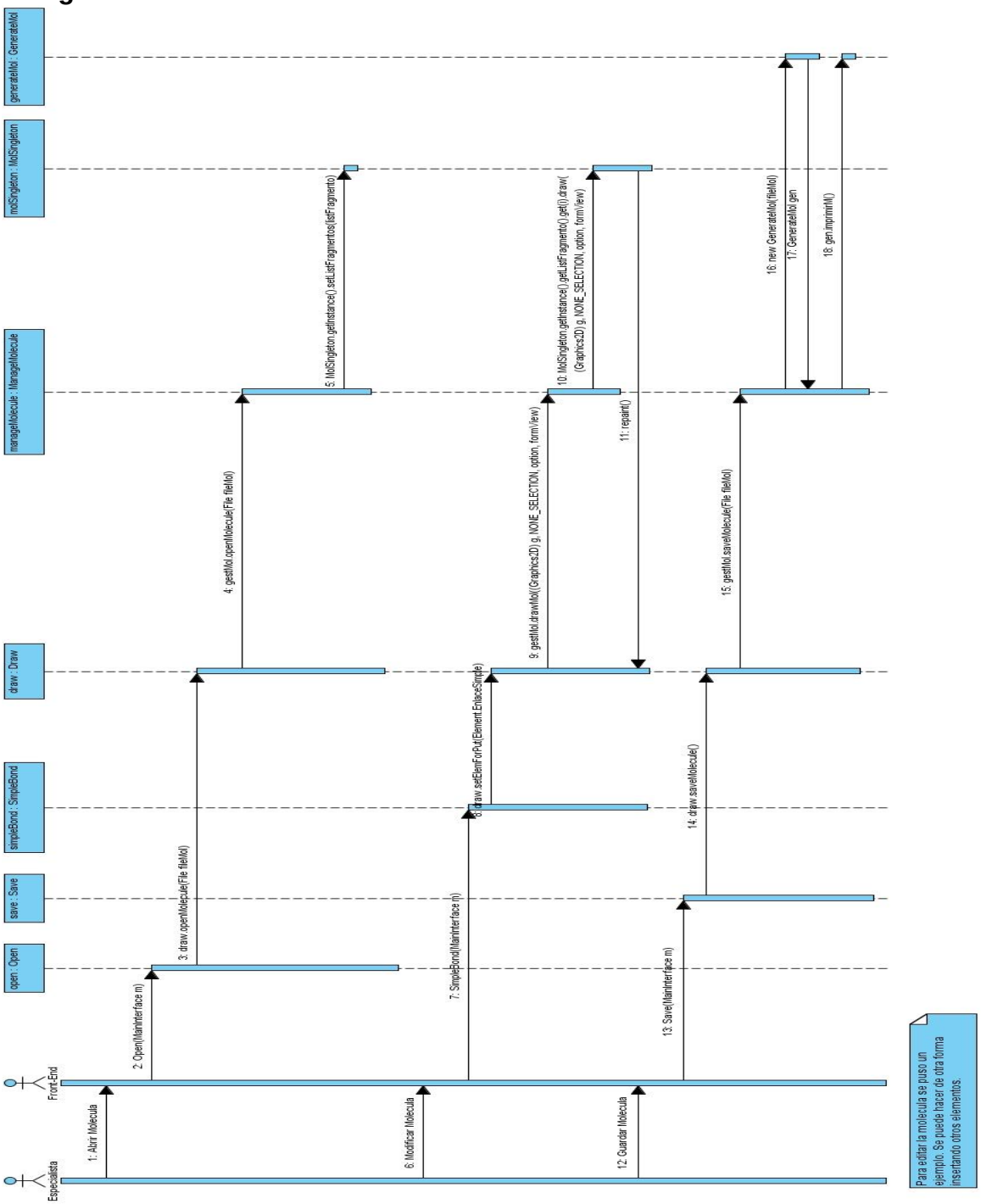


Fig. 3.7 Diagrama de Secuencia del Caso de Uso Modificar Molécula.

3.3.2.1.3 Diagrama de Secuencia del Caso de Uso Cambiar forma de edición

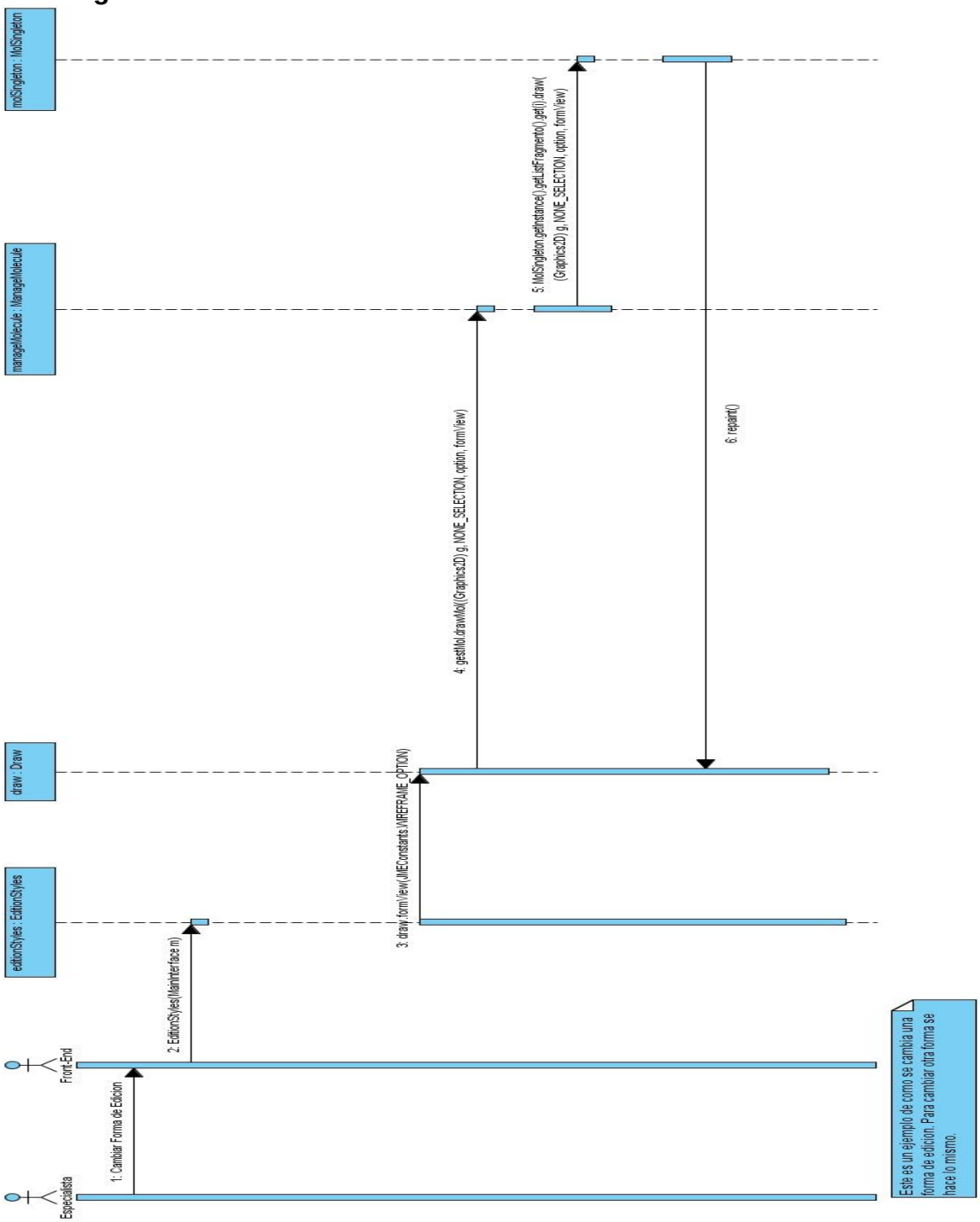


Fig. 3.8 Diagrama de Secuencia del Caso de Uso Cambiar forma de edición.

3.3.3 Modelo de Despliegue

La vista de despliegue define la arquitectura física del sistema por medio de nodos interconectados. Estos nodos son elementos de hardware sobre los cuales pueden ejecutarse los elementos software.

El despliegue del componente desarrollado se realizará como se define en el documento de arquitectura del proyecto alasGRATO.

En la siguiente figura se muestra el modelo de despliegue.

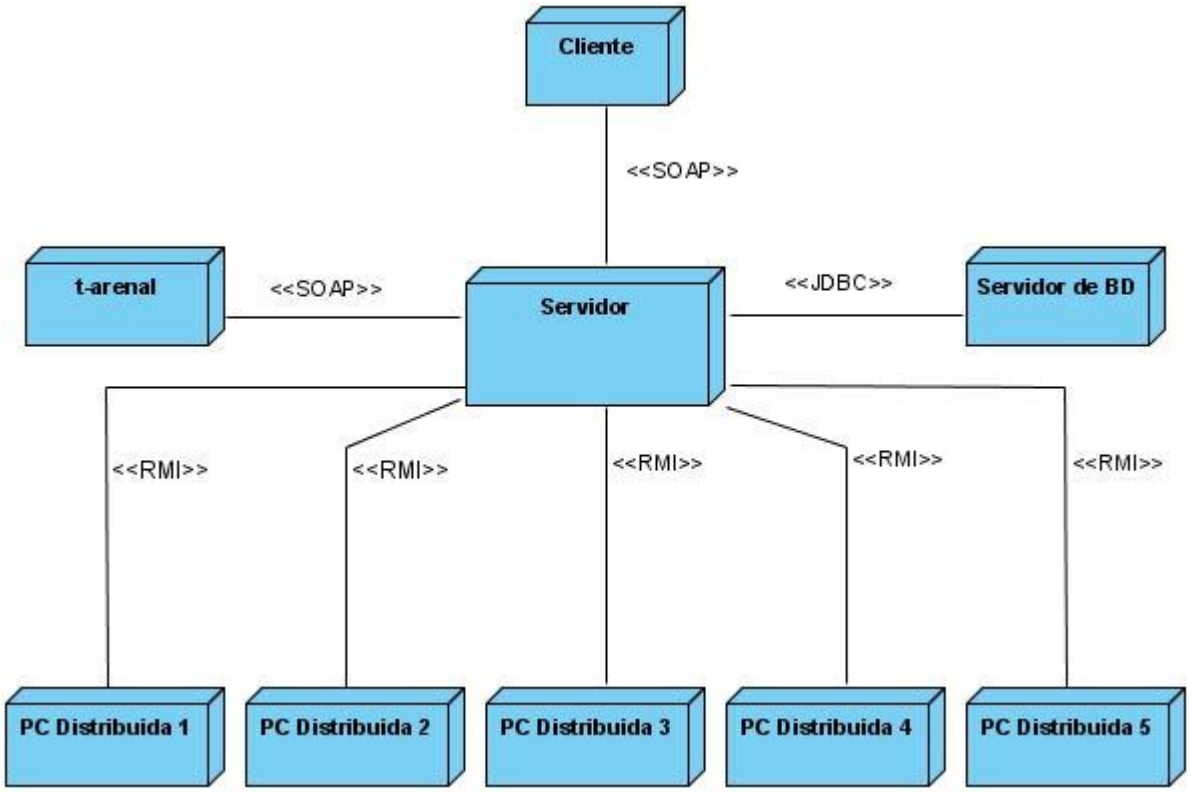


Fig. 3.9 Modelo de despliegue.

3.4 Conclusiones

- Se utilizó el patrón arquitectónico Modelo-Vista-Controlador para la implementación del plug-in Editor Molecular.
- Se utilizaron los siguientes patrones de diseño: Singleton, Composite, Controlador, Experto, Creador, Alta Cohesión, Bajo Acoplamiento.

CAPÍTULO 4: IMPLEMENTACIÓN Y PRUEBA

4.1 Introducción

En este capítulo se describe cómo fue implementado el sistema mostrando los diagramas de componentes y el prototipo funcional del mismo. También se presenta lo relacionado con las pruebas realizadas al sistema.

4.2 Implementación

4.2.1 Diagramas de componentes

Un diagrama de componentes es un diagrama que muestra un conjunto de elementos del modelo tales como componentes, subsistemas de implementación y sus relaciones. Muestran las organizaciones y dependencias lógicas entre componentes de software, sean estos de código fuente, binarios, archivos, bibliotecas cargadas dinámicamente o ejecutables.

A continuación, en la figura. 4.1, se muestra el diagrama de componentes del sistema. Para una mejor comprensión del mismo se separa el diagrama de componentes del plug-in Editor Molecular, el cual se encuentra en el Anexo 2.

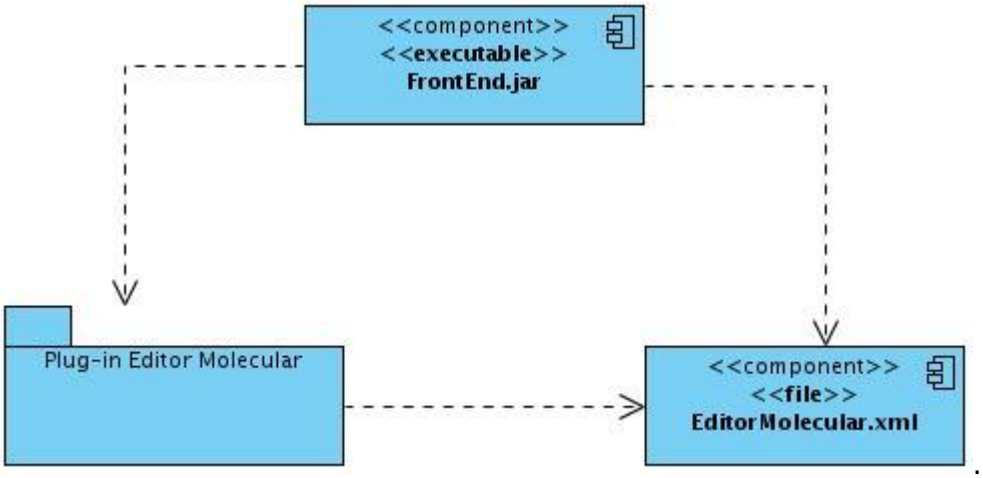


Fig. 4.1 Diagrama de componentes del sistema.

4.2.2 Prototipo funcional del plug-in Editor Molecular

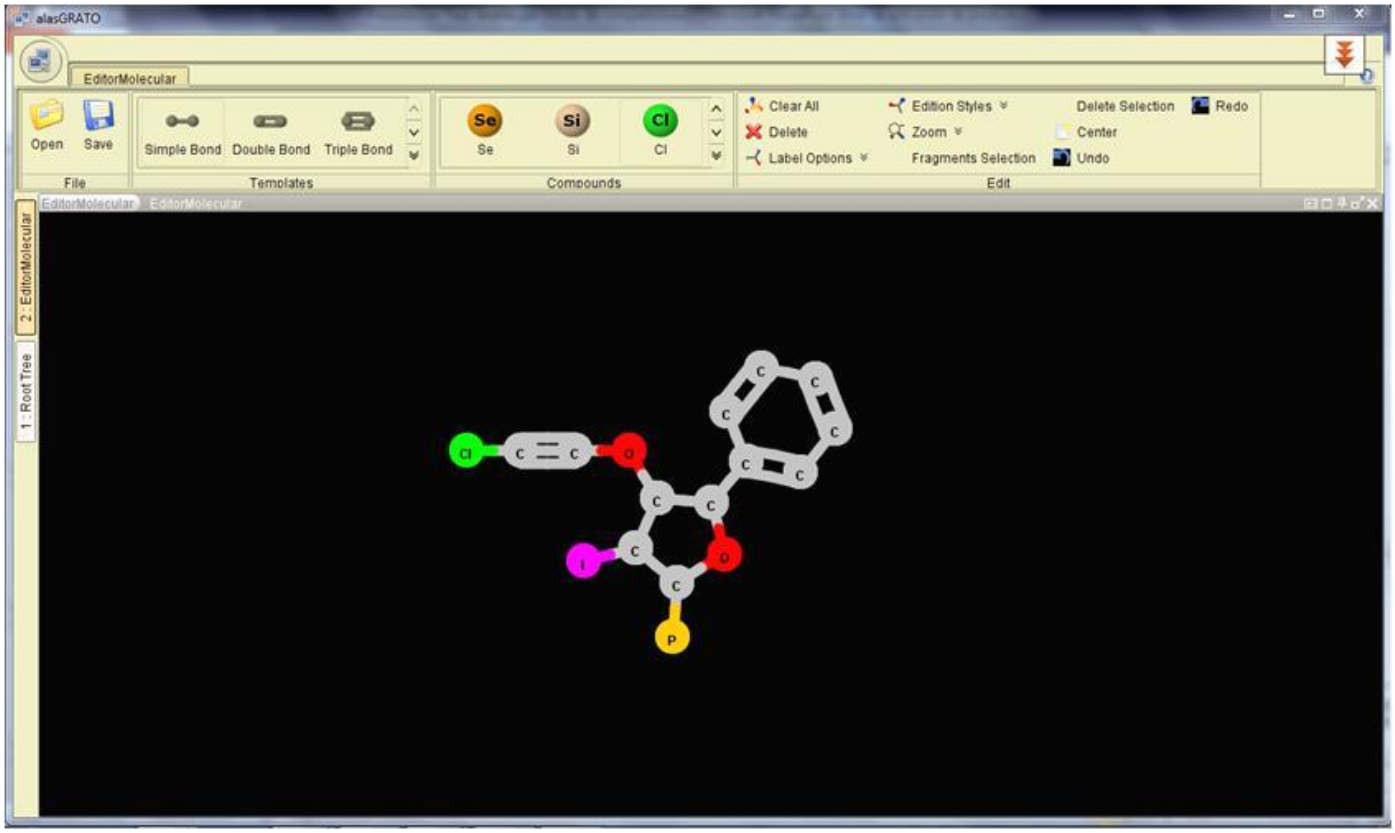


Fig. 4.2 Prototipo funcional del plug-in Editor Molecular.

4.3 Pruebas del Sistema

Las pruebas son una actividad en la cual un sistema o componente es ejecutado bajo unas condiciones o requerimientos especificados, los resultados son observados y registrados, y una evaluación es hecha de algún aspecto del sistema o componente.

La prueba de software es un elemento crítico para la garantía de la calidad del software y representa una revisión final de las especificaciones del diseño y de la codificación.

Es importante destacar que la prueba no puede asegurar la ausencia de defectos; solo puede asegurar que existen defectos en el software.

4.3.1 Plan de Prueba

Un plan de pruebas está constituido por un conjunto de pruebas. Cada prueba debe dejar claro: qué tipo de propiedades se quieren probar (corrección, robustez, fiabilidad, amigabilidad); cómo se mide el resultado; especificar en qué consiste la prueba (hasta el último detalle de cómo se ejecuta) y definir cuál es el resultado que se espera. En lo adelante se abordará todo lo relacionado con las pruebas realizadas al sistema.

4.3.1.1 Configuración del entorno de Prueba

La configuración del entorno donde se vayan a ejecutar las diferentes pruebas que se realizan a un software es un aspecto muy importante dentro del proceso de pruebas, pues si no se analizan bien los recursos de software y hardware que necesita el producto que se está construyendo, a la hora de probarlo se prescindirá de los elementos necesarios para la ejecución de un proceso de pruebas exitoso.

Por ello se tuvo en cuenta, a la hora de llevar a cabo todo el proceso de pruebas, algunos requerimientos de hardware y de software que hicieron posible un mejor desarrollo de las mismas, en aras de que se lograran minimizar los errores de la aplicación en desarrollo. Algunos de los requerimientos que se consideraron necesarios para las pruebas son los referenciados a continuación:

Requerimientos de software:

Una PC con Windows XP o superior.

Una PC con Linux (Ubuntu).

Máquina Virtual de Java 1.6.

Requerimientos de hardware:

PC con Microprocesador Pentium IV a 2.41 GHz o superior.

Con 160 GB de Disco Duro.

512 MB de memoria RAM o superior.

Fecha de Inicio	**Fecha de Fin**	**Actividades**	**Personal Implicado**
28/04/10	28/04/10	Aceptación y firma del Plan de Prueba.	Andrés M. Bores Hevia Felix M. Bores Hevia

			José R. Carrasco
30/04/10	30/04/10	Verificación de las condiciones previas para el inicio de las pruebas.	Andrés M. Bores Hevia Felix M. Bores Hevia José R. Carrasco

Primera Iteración del plug-in Editor Molecular.

02/05/10	05/05/10	Ejecución de las pruebas de Caja Negra al caso de uso Crear Molécula.	Andrés M. Bores Hevia Felix M. Bores Hevia
05/05/10	10/05/10	Ejecución de las pruebas de Caja Negra al caso de uso Modificar Molécula.	Andrés M. Bores Hevia Felix M. Bores Hevia
10/05/10	11/05/10	Ejecución de las pruebas de Caja Negra al caso de uso Cambiar Forma de Edición.	Andrés M. Bores Hevia Felix M. Bores Hevia
12/05/10	12/05/10	Análisis de las no conformidades y las solicitudes de cambio.	Andrés M. Bores Hevia Felix M. Bores Hevia

4.3.2 Diseño de las Pruebas de Unidad (Caja Negra)

Las pruebas de Caja Negra se refieren a las pruebas que se llevan a cabo sobre la interfaz del software. O sea, los casos de prueba pretenden demostrar que las funciones del software son operativas, que la entrada se acepta de forma adecuada y que se produce un resultado correcto, así como que la integridad de la información externa se mantiene.

Caso de Uso: Crear Molécula

Secciones a probar en el caso de uso.

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**

SC1: Crear Molécula	EC 1.1: Crear Molécula	Cuando el especialista ejecuta el módulo Editor Molecular de la opción módulos del Visualizador alasGRATOViewer, sin tener ninguna molécula seleccionada en el árbol de ficheros cargados.	Selecciona las plantillas en la barra de herramienta.

El diseño de las pruebas por cada sección definida anteriormente se encuentra en el Anexo 3.

Caso de Uso: Modificar Molécula

Secciones a probar en el caso de uso.

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**
SC2: Modificar Molécula	EC 2.1: Modificar Molécula	Cuando el especialista ejecuta el módulo Editor Molecular de la opción módulos del Visualizador alasGRATOViewer, teniendo una molécula seleccionada en el árbol de ficheros cargados. En el área de trabajo aparecerá la molécula dibujada.	Selecciona abrir molécula

El diseño de las pruebas por cada sección definida anteriormente se encuentra en el Anexo 4.

Caso de Uso: Cambiar forma de edición.

Secciones a probar en el caso de uso.

Nombre de la sección	**Escenarios de la sección**	**Descripción de la funcionalidad**	**Flujo central**

SC3: Cambiar forma de edición	EC 3.1: Cambiar forma de edición	Cuando el especialista selecciona la opción de cambiar la forma de edición presente en la barra de herramientas.	Selecciona Cambiar la Forma de Edición en la barra de herramientas.

El diseño de las pruebas por cada sección definida anteriormente se encuentra en el Anexo 5.

4.3.2.1 No Conformidades detectadas

Elemento	**No**	**No Conformidad**	**Aspecto Correspondiente**	**Etapa de Detección**	**Significativo**	**No Significativo**
Interfaz 1	1	No validaba la molécula del fichero.	Cuando se intentaba abrir el fichero que contenía la molécula.	Implementación	X	
Interfaz 2	2	No mostraba correctamente los Id de los átomos.	Cuando se cambiaba para la Forma de visualizar la etiqueta de los átomos.	Implementación	X	
Interfaz 3	3	No centraba correctamente.	Cuando se oprimía el botón Centrar.	Implementación	X	
Interfaz 4	4	No Realizaba correctamente la Opción Rehacer y Deshacer.	Cuando se oprimía los botones Undo y Redo.	Implementación	X	
Interfaz 5	5	No eliminaba todos los Objetos en el área seleccionada para eliminar	Cuando se oprimía el botón Borrar Área Seleccionada	Implementación	X	
Interfaz 6	6	No verificaba correctamente la Hibridación	Cuando se intentaba cambiar o insertar algún	Implementación	X	

		de los átomos	elemento en la molécula.			
Interfaz 7	7	Permitía Guardar una molécula fraccionada	Cuando se oprimía el botón Guardar.	Implementación	X	
Interfaz 8	8	No cambiaba el enlace común entre dos ciclos.	Cuando se insertaba un ciclo a partir de un enlace.	Pruebas	X	

4.3 Conclusiones

- Se logró implementar el plug-in Editor Molecular con éxito.
- Se realizaron las pruebas de Caja Negra al sistema, detectándose algunas no conformidades las cuales fueron registradas y solucionadas obteniendo un buen funcionamiento del software implementado.

CONCLUSIONES

- Se diseñó e implementó una aplicación multiplataforma para la visualización y edición de compuestos orgánicos sin perder las preferencias de visualización existentes en la plataforma.
- Se desarrolló el plug-in para la segunda versión de la plataforma y se integró a la misma.
- Se realizaron satisfactoriamente las pruebas de funcionalidad a la aplicación.

RECOMENDACIONES

1. Continuar añadiendo restricciones químicas al Editor Molecular para así mejorar la edición.
2. Continuar con la implementación de nuevas funcionalidades, con el objetivo de ampliar la potencialidad del editor.
3. Poner a disposición de varios especialistas el editor para que sea probado, con el objetivo de obtener sugerencias para mejorar las funcionalidades del editor.

REFERENCIAS BIBLIOGRÁFICAS

1. HOFFMANN, R., LAZLO, P. Representation in Chemistry. *Angewandte Chemie, 1991,* 30(1): 1-16.

2. SCHUMMER, J. The Chemical Core of Chemistry I: A Conceptual Approach. *Hyle-An International for the Philosophy of Chemistry,* 1998, 4(2): 129-162.

3. TURRO, N. J., Geometric and topological Thinking in Organic Chemistry. *Angewandte Chemie*, 1998, 25: 882-901.

4. KLEIN, D. J. Aromaticity via Kekule Structures and Conjugated Circuits. *Journal of Chemical Education*, 1992, 69(9): 691-700.

5. MIHALIC, Z., TRINAJSTIC, N. A Graph-Theoretical Approach to Structure-Property Relationships. *Journal of Chemical Education*, 1992, 69(9): 701-712.

6. RANIC, M. Chemical Structure- What Is “She”?. *Journal of Chemical Education*, 1992, 69(9): 713-718.

7. ECKROTH, D. The Correct von Baeyer Name for (Buckminster)fullerane. *Journal of Chemical Education*, 1993, 70(8): 609–611.

8. KLUGE, F. LARDER, D. A. M. Butlerov: On the Chemical Structure of Substances. *Journal of Chemical Education*, 1971, 48(5): 289-291.

9. PEREZ VALDEZ, Y. R. y BRAVO RODRIGUEZ, Y. C. *Interfaz para la visualización y edición de estructuras químicas.* Tesis de Diploma inédita. Universidad de las Ciencias Informáticas, 2007.

10. ACD/ChemSketch Freeware. [2009] Disponible en: http://reactivereports.org/download/chemsketch/

11. ACD/ChemSketch. [2009] Disponible en: http://www.acdlabs.com/products/draw_nom/draw/chemsketch/

12. ChemDraw Ultra 12.0 Suite. [2009] Disponible en: http://www.hearne.com.au/products/chemdraw/edition/ultra/

13. MarvinSketch. [2009] Disponible en: http://www.chemaxon.com/products/marvin/marvinsketch/

14. HyperChem. [2009] Disponible en: http://www.hyper.com/

15. Open UP. [2009]. Disponible en: http://www.openup.es/

BIBLIOGRAFÍA

1. HOFFMANN, R., LAZLO, P. Representation in Chemistry. *Angewandte Chemie, 1991,* 30(1): 1-16.

2. SCHUMMER, J. The Chemical Core of Chemistry I: A Conceptual Approach. *Hyle-An International for the Philosophy of Chemistry,* 1998, 4(2): 129-162.

3. TURRO, N. J., Geometric and topological Thinking in Organic Chemistry. *Angewandte Chemie*, 1998, 25: 882-901.

4. KLEIN, D. J. Aromaticity via Kekule Structures and Conjugated Circuits. *Journal of Chemical Education*, 1992, 69(9): 691-700.

5. MIHALIC, Z., TRINAJSTIC, N. A Graph-Theoretical Approach to Structure-Property Relationships. *Journal of Chemical Education*, 1992, 69(9): 701-712.

6. RANIC, M. Chemical Structure- What Is "She"?. *Journal of Chemical Education*, 1992, 69(9): 713 -718.

7. ECKROTH, D. The Correct von Baeyer Name for (Buckminster)fullerane. *Journal of Chemical Education*, 1993, 70(8): 609-611.

8. KLUGE, F. LARDER, D. A. M. Butlerov: On the Chemical Structure of Substances. *Journal of Chemical Education*, 1971, 48(5): 289-291.

9. PEREZ VALDEZ, Y. R. y BRAVO RODRIGUEZ, Y. C. *Interfaz para la visualización y edición de estructuras químicas.* Tesis de Diploma inédita. Universidad de las Ciencias Informáticas, 2007.

10. ACD/ChemSketch Freeware. [2009] Disponible en: http://reactivereports.org/download/chemsketch/

11. ACD/ChemSketch. [2009] Disponible en: http://www.acdlabs.com/products/draw_nom/draw/chemsketch/

12. ChemDraw Ultra 12.0 Suite. [2009] Disponible en: http://www.hearne.com.au/products/chemdraw/edition/ultra/

13. MarvinSketch. [2009] Disponible en: http://www.chemaxon.com/products/marvin/marvinsketch/

14. HyperChem. [2009] Disponible en: http://www.hyper.com/

15. Open UP. [2009]. Disponible en: http://www.openup.es/

16. ASCUI, I.C. Java y la Programación Orientada a Objetos. [2010]. Disponible en: http://iverclaros.blog.galeon.com/1141775160/.

17. PRESSMAN, R.S. *Ingeniería del Software. Un enfoque práctico*, edición V, 2005, 1.

18. ZUKOWSKI, J. *Programación Java 2 j2se 1.4,* 2007, 1,2,3:1-975.

ANEXOS

Anexo 1: Descripción de las clases del diseño.

CLASE : Atom

ATRIBUTOS	TIPO
-hib	int
-angs	double[]
-ang	double
-op	int
-value	String
-id	int
-conected	ArrayList<Atom>

OPERACIONES	DESCRIPCIÓN
+Atom(p : Point, id : int)	Este método es el constructor de la clase.
+getConected() : ArrayList<Atom>	Este método devuelve una lista de los átomos conectados.
+setConected(conected : ArrayList<Atom>) : void	Este método estable la lista de átomos conectados.
+getId() : int	Este método devuelve el id del átomo.
+setId(id : int) : void	Este método establece el id del átomo.
+setAng(ang : double, op : int, repAng : boolean) : void	Este método establece el ángulo de dibujo del átomo.
+pointAtAngle(ang : double) : Point	Este método determina un punto a partir de un ángulo.
+pointAtAngle(ang : double, signoY : int, signoX : int) : Point	Este método determina un punto a partir de un ángulo.
+getNextPoints() : Point[]	Este método determina los posibles puntos donde se pueda colocar un átomo.
+isMouseEnter(p : Point) : boolean	Este método determina si el mouse está tocando el átomo.
+getColor() : Color	Este método devuelve el color del átomo.
+draw(g : Graphics2D, mode : int, option : int, fromView : int) : void	Este método dibuja el átomo.
+drawString(g : Graphics2D, formView : int, opction : int) : void	Este método dibuja una cadena.
+equals(obj : Object) : boolean	
+equalsPoint(p : Point) : boolean	Este método determina si dos puntos son iguales.
+getOp() : int	Este método determina la dirección en que será pintado el átomo.
+getValue() : String	Este método devuelve el valor del átomo.

+setValue(value : String) : void	Este método establece el valor del átomo.
+getAng() : double	Este método devuelve el ángulo de dibujo del átomo.
+getHib() : int	Este método devuelve la hibridación del átomo.
+getHibTotal() : int	Este método devuelve la hibridación total del átomo.
+getHibValue(valor : String) : int	Este método devuelve la hibridación de n compuesto.
+setHib(value : int) : void	Este método establece la hibridación del átomo.
+setHibMas(value : int) : void	Este método aumenta la hibridación del átomo en un valor dado.
+setHibMenos(value : int) : void	Este método disminuye la hibridación del átomo en un valor dado.
+delete(list : List<Atom>) : void	Este método borra el átomo.
+deleteAtomConected(at : Atom) : void	Este método borra los átomos conectados.
+addAtmConected(at : Atom) : void	Este método adiciona un átomo conectado.
+clone() : Object	
+Clone() : Atom	Este método clona el átomo.
+isInArea(start : Point, end : Point) : boolean	Este método determina si un átomo está en un área determinada.
+hasAng() : boolean	Este método determina si el átomo tiene ángulo a partir del cual se creo.

CLASE : Benzene

OPERACIONES	DESCRIPCIÓN
+Benzene(patomo : ArrayList<Atom>)	Este método es el constructor de la clase.
+createCicle(list : List<Atom>) : void	Este método crea el ciclo.
+generateCiclo(p : Point, cant : int, startAng : double, idCount : int) : void	Este método genera un ciclo.
+generateCicloAtomo(a : Atom, p : Point, cant : int, startAng : double, idCount : int) : void	Este método genera un ciclo a partir de un átomo.
+createCicleEnlace(link : Bond, list : List<Atom>) : void	Este método genera un ciclo a partir de un enlace.
+getRad(cant : int) : double	Este método devuelve el radio.
+generateCicloEnlace(e : Bond, p : Point, cant : int, startAng : double, idCount : int) : void	Este método genera un ciclo a partir de un enlace.

CLASE : Cycle

ATRIBUTOS	TIPO
#center	Point

#radio	double
#startAng	double
#angInc	double
#listEnlace	ArrayList<Bond>

OPERACIONES	DESCRIPCIÓN
+Cycle(patomo : ArrayList<Atom>)	Este método es el constructor de la clase.
+setCenter(center : Point) : void	Este método establece el centro del ciclo
+setRadio(v : double) : void	Este método establece el radio del ciclo.
+setStartAng(startAng : double) : void	Este método establece el ángulo inicial de dibujo del ciclo.
+setAngInc(angInc : double) : void	Este método establece el ángulo de incremento para dibujar el ciclo.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el ciclo.
+generateCicloEnlace(e : Bond, p : Point, cant : int, startAng : double, idCount : int) : void	Este método genera un ciclo a partir de un enlace.
+generateCicloAtomo(a : Atom, p : Point, cant : int, startAng : double, idCount : int) : void	Este método genera un ciclo a partir de un átomo.
+generateCiclo(p : Point, cant : int, double, startAng : int idCount) : void	Este método genera un ciclo.
+createCicle(list : List<Atom>) : void	Este método crea un ciclo.
+createCicleEnlace(link : Bond, list : List<Atom>) : void	Este método crea un ciclo a partir de un enlace.
+getRad(cant : int) : double	Este método devuelve el radio.
+getRadio() : double	Este método devuelve el radio.
+getListEnlace() : ArrayList<Bond>	Este método devuelve la lista de enlaces del ciclo.
+setListEnlace(list : ArrayList<Bond>) : void	Este método establece la lista de enlaces del ciclo.
+getEnlace(pos : int) : Bond	Este método devuelve un enlace.
+isMouseEnter(p : Point) : boolean	Este método determina si el ciclo está siendo tocado por le mouse.
+changeEnlace(link : Bond, cant : int) : void	Este método cambia un enlace.
+getEnlace(p1 : Point, p2 : Point) : Bond	Este método devuelve un enlace.
+getCenter() : Point	Este método devuelve el centro del ciclo.

CLASE : Bond

OPERACIONES	DESCRIPCIÓN
+Bond(patomo : ArrayList<Atom>)	Este método es el constructor.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el enlace.
+getHibEnlace() : int	Este método devuelve la hibridación del enlace.

+Clone(listAtomos : ArrayList<Atom>) : Bond	Este método devuelve una copia el enlace.
+isConected(link : Bond) : boolean	Este método determina si un enlace está conectado con otro.
+isMouseEnter(p : Point) : boolean	Este método determina si el enlace está siendo tocado por el mouse.
+equals(Object obj) : boolean	Este método determina si 2 enlaces son iguales.
+listPoint() : ArrayList<Point>	Este método devuelve la lista de punto del enlace.
+delete(list : List<Atom>) : void	Este método borra el enlace.
+isInArea(start : Point, end : Point) : boolean	Este método determina si el enlace está en una área determinada.

CLASE : DoubleBond

OPERACIONES	DESCRIPCIÓN
+ DoubleBond(patomo : ArrayList<Atom>)	Este método es el constructor.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el enlace.
+getHibEnlace() : int	Este método devuelve la hibridación del enlace.
+Clone(listAtomos : ArrayList<Atom>) : Bond	Este método devuelve una copia el enlace.

CLASE : SimpleBond

OPERACIONES	DESCRIPCIÓN
+ SimpleBond(patomo : ArrayList<Atom>)	Este método es el constructor.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el enlace.
+getHibEnlace() : int	Este método devuelve la hibridación del enlace.
+Clone(listAtomos : ArrayList<Atom>) : Bond	Este método devuelve una copia el enlace.

CLASE : TripleBond

OPERACIONES	DESCRIPCIÓN
+ TripleBond(patomo : ArrayList<Atom>)	Este método es el constructor.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el enlace.
+getHibEnlace() : int	Este método devuelve la hibridación del enlace.
+Clone(listAtomos : ArrayList<Atom>) : Bond	Este método devuelve una copia el enlace.

CLASE : Fragment

ATRIBUTOS	TIPO
#listAtomo	ArrayList<Atom>

OPERACIONES	DESCRIPCIÓN
+Fragment(patomo : ArrayList<Atom>)	Este método es el constructor.
+getAtomo(pos : int) : Atom	Este método devuelve un átomo.
+getAtomos() : ArrayList<Atom>	Este método devuelve la lista de átomos.
+setAtomo(pos : int, valor : Atom) : void	Este método establece un átomo en la lista de átomos.
+draw(g : Graphics2D, mode : int, option : int, formView : int) : void	Este método dibuja el fragmento.
+isMouseEnter(p : Point) : boolean	Este método determina si el fragmento está siendo tocado por el mouse.
+listPoint() : ArrayList<Point>	Este método devuelve la lista de puntos del fragmento.
+Clone(listAtomos : ArrayList<Atom>) : Fragment	Este método devuelve una copia del fragmento.

CLASE : GenerateMol

ATRIBUTOS	TIPO
-ubicacion	String
-nombre	String

OPERACIONES	DESCRIPCIÓN
+GenerateMol(location : String)	Este método es el constructor.
+imprimirM() : void	Este método guarda en un fichero la molécula dibujada.
+getUbicacion() : String	Este método devuelve la ubicación del fichero.
+setUbicacion(ubicacion : String) : void	Este método establece la ubicación del fichero.

CLASE : MolSingleton

ATRIBUTOS	TIPO
-mol	Molecule

OPERACIONES	DESCRIPCIÓN
-MolSingleton()	Este método es el constructor.
+getInstance() : Molecule	Este método devuelve una instancia de la molécula.
+changeInstance(m : Molecule) : void	Este método cambia la instancia de la molécula.

CLASE : MolObject

OPERACIONES	DESCRIPCIÓN
+isMouseEnter(Point p) : boolean	Este método determina si un objeto está siendo tocado por el mouse.

+draw(g : Graphics2D, mode : int, option : int, fromView : int) : void	Este método dibuja el objeto.
+isInArea(start : Point, end : Point) : boolean	Este método determina si un objeto está en un área determinada.

CLASE : ManageMolecule

OPERACIONES	DESCRIPCIÓN
+ManageMolecule()	Constructor de la clase.
+setMolecule(mol : Molecule) : void	Este método cambia la molécula que está siendo editada.
+getMolecule() : Molecule	Este método devuelve la molécula que está siendo editada.
+setCantidadCicloMas8(int value) : void	Este método establece la cantidad de lados para crear un ciclo de más de 8 lados.
+setIdCount(id : int) : void	Este método estable un nuevo idCount.
+crearMolecula(fileMol : File) : void	Este método crea la molécula a partir de un fichero.
-AEnlaces(listEnlaces : ArrayList<Bond>, listX : ArrayList<Integer>, listY : ArrayList<Integer>, listAtomo : ArrayList<Atom>, hashTable: Hashtable<Integer, ArrayList<String>>, marcados : ArrayList<Boolean>, valor : ArrayList<Integer>, contador : int) : ArrayList<Bond>	Este método devuelve todos los enlaces de una molécula cuando es cargada.
-BuscarEnlace(listEnlaces : ArrayList<Bond>, id : int) : int	Este método devuelve la posición de un enlace en una lista de enlaces el cual posea un átomo específico.
-BuscarCiclos(listEnlaces : ArrayList<Bond>, listCiclo : ArrayList<Integer>, enlace : int, enlacesVisitados : boolean[]) : boolean	Este método busca ciclos en la molécula.
+getListFragmento() : ArrayList<Fragment>	Este método devuelve la lista de fragmentos de la molécula
+getEnlace(a1 : Atom, a2 : Atom) : Bond	Este método busca el enlace que posea los átomos a1 y a2 y lo retorna.
+CrearEnlace(from : Atom, atom : Atom, cant : int) : void	Este método crea un enlace a partir de un átomo y lo adiciona a la lista de fragmentos.
-ModificarEnlace(from : Atom, atom : Atom, cant : int, pos : int) : void	Este método cambia un enlace de la molécula por otro.
+getObjectTouch(p : Point) : Objeto	Este método devuelve el objeto que está siendo tocado por el mouse.
+getEnlaceTouch(p : Point) : Objeto	Este método devuelve el enlace que está siendo tocado por el mouse
+getMinObjectTouch(p : Point) : Objeto	Este método devuelve el objeto más

	pequeño que está siendo tocado por el mouse.
-getAtomTouch(p : Point) : Atomo	Devuelve el átomo que está siendo tocado por un punto.
+createNewElement(elem : Element, p : Point) : void	Este método agrega un objeto a la molécula cuando está vacía.
+changeAtomValue(from : MolObject, value : String) : void	Este método cambia el valor de un átomo
+putNewElement(elem : Element, from : MolObject, p : Point) : void	Este método adiciona un nuevo objeto a la molécula, a partir de otro objeto.
+putBenzeno(link : Bond) : void	Este método adiciona un benceno a partir de un enlace.
+putBenzeno(at : Atom) : void	Este método adiciona un benceno a la molécula a partir de un átomo.
+putCiclo(link : Bond, cant : int) : void	Este método adiciona un ciclo a la molécula a partir de un enlace.
-cantEnlacesComun(c : Cycle) : int	Este método devuelve la cantidad de átomos en común que posee un ciclo con los demás objetos que posee la molécula.
-cyclesEquals(c : Cycle) : boolean	Este método es para saber si este ciclo es igual al pasado por parámetro
+putCiclo(at : Atom, cant : int) : void	Este método adiciona un ciclo a partir de un átomo.
+putEnlace(ciclo : Cycle, link : Bond, cant : int) : void	Este método cambia un enlace a un ciclo.
+putEnlace(link : Bond, cant : int) : void	Este método adiciona un nuevo enlace a la molécula a partir de un enlace.
+putEnlace(from : Atom, cant : int) : void	Este método adiciona un nuevo enlace a la molécula a partir de un átomo.
-isInTripleLink(a : Atom) : boolean	Este método verifica si un átomo pertenece a un triple enlace.
-getHibEnlace(lk : Bond) : int	Este método retorna la hibridación de un enlace.
-getAngle(p1 : Point, p2 : Point) : double	
+deleteObjTemp(obj : MolObject) : void	Elimina el objeto.
+isAllConected() : boolean	Método para saber si la molécula es conecta (que no esté particionada).
+getEnlacePoint(p1 : Point, p2 : Point) : Enlace	Este método busca un enlace que contenga 2 puntos iguales a los pasados por parámetros.
+middlePoint() : Point	Este método devuelve el punto medio de la molécula.
+clearAll() : void	Este método limpia la molécula, es decir, borra todos sus elementos.

+isClear() : boolean	Este método verifica si la molécula está vacía o no.
+deleteObjInSelectedArea(start : Point, end : Point) : void	Este método borra todos los objetos que estén dentro de un área.
+drawMol(g : Graphics2D, noneSelection : int, option : int, formView : int) : void	Este método manda a dibujar la molécula.
+saveMolecule(fileMol : File) : void	Este método salva la molécula en edición.
+optimizeMolecule(fileMol : File) : void	Este método optimiza la molécula que se va a salvar.
+openMolecule(fileMol : File) : void	Este método se utiliza para cargar una molécula desde un fichero.
-setAngles(frags : List<Fragment>) : void	Este método establece el ángulo a partir del cual se crearon los átomos de cada fragmento.

CLASE : Molecule

ATRIBUTOS	TIPO
-listFragmento	ArrayList<Fragment>
-listAtom	ArrayList<Atom>
-idCount	int
-cantCicloMas8	int

OPERACIONES	DESCRIPCIÓN
+Molecule	Este método es el constructor de la clase.
+setListAtom(list : ArrayList<Atom>) : void	Este método establece una nueva lista de átomos.
+setListFragmentos(list : ArrayList<Fragment>) : void	Este método establece una nueva lista de fragmentos.
+setCantidadCicloMas8(value : int) : void	Este método establece la cantidad de lados para crear un ciclo de más de 8 lados.
+setIdCount(id : int) : void	Este método establece un nuevo idCount.
+getListFragmento() : ArrayList<Fragment>	
+getListAtom() : ArrayList<Atom>	
+getIdCount() : int	
+getCantCicloMas8() : int	
+addFragment(f : Fragment) : void	Este método adiciona un nuevo Fragment a la molécula.
+addAtom(a : Atom) : void	Este método adiciona un nuevo átomo a la molécula.
+containAtom(a : Atom) : boolean	Este método verifica si un átomo está ya en la lista de átomos.
+changeFragment(pos : int, f : Fragment) : void	Este método cambia un fragmento por otro

	en la lista de fragmentos.
+removeFragment(obj : MolObject) : void	
+clear() : void	Este método limpia la molécula. Es decir, elimina todos sus fragmentos.
+isClear() : boolean	Este método verifica si la molécula está limpia. Es decir, que no contenga ningún fragmento.
+Clone() : Molecule	Este método crea una copia de la molécula

CLASE : Draw

ATRIBUTOS	TIPO
-mouse	Point
-startMove	Point
-mode	int
-multiSelecMode	boolean
-toPaintSelectArea	boolean
-startSelect	Point
-endSelect	Point
-value	String
-option	int
-formView	int
-dx	int
-dy	int
-actMvX	int
-actMvY	int
-zoom	double
-elem	Element
-middleTransformedPointDA	Point
-dxMP	int
-dyMP	int
-transform	AffineTransform
-undoStack	Stack<Molecule>
-redoStack	Stack<Molecule>
-gestMol	ManageMolecule

OPERACIONES	DESCRIPCIÓN
+Draw()	Este método es el constructor de la clase.
+isMultiSelecMode() : boolean	Este método verifica si el modo de selección múltiple está activado.
+changeMultiSelecMode() : void	Este método cambia el modo de selección múltiple. Lo activa si esta desactivado y viceversa.
+zoom(z : boolean) : void	Este método realiza el zoom.

-zoomIn() : void	Este método hace un zoom in.
-zoomOut() : void	Este método hace un zoom out.
+getTransformPoint(pn : Point) : Point	Este método transforma un punto a sus coordenadas reales.
#paintComponent(g : Graphics) : void	
+setElemForPut(elem : Element) : void	Este método establece el nuevo elemento para adicionar a la molécula.
+setValueForChange(value : String) : void	Este método establece un nuevo valor a cambiar de un átomo.
+ConvertirMol(fileMol : File) : void	
+viewLabel(option : int) : void	Este método establece la forma de ver el valor de los átomos.
+getViewLabel() : int	Este método devuelve la forma de ver el valor de los átomos.
+clearAll() : void	Este método vacía la molécula.
+formView(formView : formView int) : void	Este método establece la forma de dibujar la molécula.
+getFormView() : int	Este método devuelve la forma de dibujar la molécula.
+setDrawMode(mode : int) : void	Este método establece el modo de dibujo. Pueden ser borrar o seleccionar, y por defecto está el seleccionar.
+setCantidadCicloMas8(value : int) : void	Este método establece la cantidad de lados del ciclo, cuando es más de 8, que se va a adicionar.
+undo() : void	Este método deshace la última acción realizada sobre la molécula.
+redo() : void	Este método rehace la última acción que se deshizo sobre la molécula.
+center() : void	Este método centra la molécula.
+drawSelectedArea(g2d : Graphics2D, alpha : float, type : int) : void	Este método dibuja el área que el usuario selecciona para borrar.
+deleteSelectedObj() : void	Este método borra los elementos que están dentro del área marcada.
-makeComposite(alpha : float, type : int) : AlphaComposite	Este método devuelve una instancia de AlphaComposite, para aplicar transparencia.
+getGestMol() : ManageMolecule	Este método devuelve la instancia de la clase ManageMolecule.
+saveMolecule() : void	Este método salva la molécula en edición.

CLASE : JMEConstants

ATRIBUTOS	TIPO
+NONE_SELECTION	int

+DEFAULT_SELECTION	int
+ERASE_SELECTION	int
+RADIO	int
+LINE_RADIO	int
+DEFAULT_ATOM_VALUE	String
+ENLACE_LENGTH	int
+Font FONT	int
+NONE_OPTION	int
+NUMBER_OPTION	int
+SYMBOL_OPTION	int
+BALLS_OPTION	int
+WIREFRAME_OPTION	int
+TRANSPARENCY	float
+TRANSPARENCY_TYPE	int
+TRANSPARENCY_COLOR	Color
+CARBONO	String

CLASE : Format

ATRIBUTOS	TIPO
-nf	NumberFormat
-campo	int
-precision	int

OPERACIONES	DESCRIPCIÓN
+Format(c : int, p : int)	Este método es un constructor de la clase.
+ Format(c : int)	Este método es un constructor de la clase.
+fmt(ent : String) : String	Este método aplica el formato deseado a una String.
+fmtd(ent : String) : String	Este método aplica el formato deseado a una String.
+fmt(c : char) : String	Este método aplica el formato deseado a un char.
+fmt(db : double) : String	Este método aplica el formato deseado a un double.
+fmt(fl : float) : String	Este método aplica el formato deseado a un float.
+fmt(ij : int) : String	Este método aplica el formato deseado a un int.
+fmt(l : long) : String	Este método aplica el formato deseado a un long.
+fmt(by : byte) : String	Este método aplica el formato deseado a un byte.
+fmt(bo : boolean) : String	Este método aplica el formato deseado a un boolean.

CLASE : Graph

ATRIBUTOS	TIPO
-matAdy	boolean[][]
-cnt	int
-orden	int[]
-cantVert	int

OPERACIONES	DESCRIPCIÓN
+Graph(boolean[][] matAdy)	Este método es el constructor de la clase.
-void searchC(int v)	Este método realiza el DFS.
+boolean isConected(int ini)	Este método verifica si el grafo es conecto o no.

CLASE : As

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+As(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Br

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+Br(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cl

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+Cl(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : F

ATRIBUTOS	TIPO

-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+F(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : S

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+S(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : C

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+C(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : O

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+O(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : N

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+N(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : P

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+P(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : I

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+I(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : H

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+H(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Se

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+Se(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Si

ATRIBUTOS	TIPO
-m	MainInterface
-value	String

OPERACIONES	DESCRIPCIÓN
+Si(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Center

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Center(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : ClearAll

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+ClearAll(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle3

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle3(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle4

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle4(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle5

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle5(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle6

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle6(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle7

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle7(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Cycle8

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Cycle8(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : CycleBenzene

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+CycleBenzene(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : CyclePlus8

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+CyclePlus8(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : DeleteFragment

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+DeleteFragment(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : DeleteSelectedFragment

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+DeleteSelectedFragment(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : SimpleBond

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+SimpleBond(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : DoubleBond

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+DoubleBond(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : TripleBond

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+TripleBond(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : SelectFragments

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+SelectFragments(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Undo

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Undo(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Redo

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Redo(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Save

ATRIBUTOS	TIPO
-m	MainInterface

OPERACIONES	DESCRIPCIÓN
+Save(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : EditionStyles

ATRIBUTOS	TIPO
-m	MainInterface
-draw	Draw
-alambre	JCheckBoxMenuItem
-bolas	JCheckBoxMenuItem

OPERACIONES	DESCRIPCIÓN
+EditionStyles(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : LabelOptions

ATRIBUTOS	TIPO
-m	MainInterface
-draw	Draw
-numero	JCheckBoxMenuItem
-simbolo	JCheckBoxMenuItem
-none	JCheckBoxMenuItem

OPERACIONES	DESCRIPCIÓN
+LabelOptions(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : Zoom

ATRIBUTOS	TIPO
-m	MainInterface

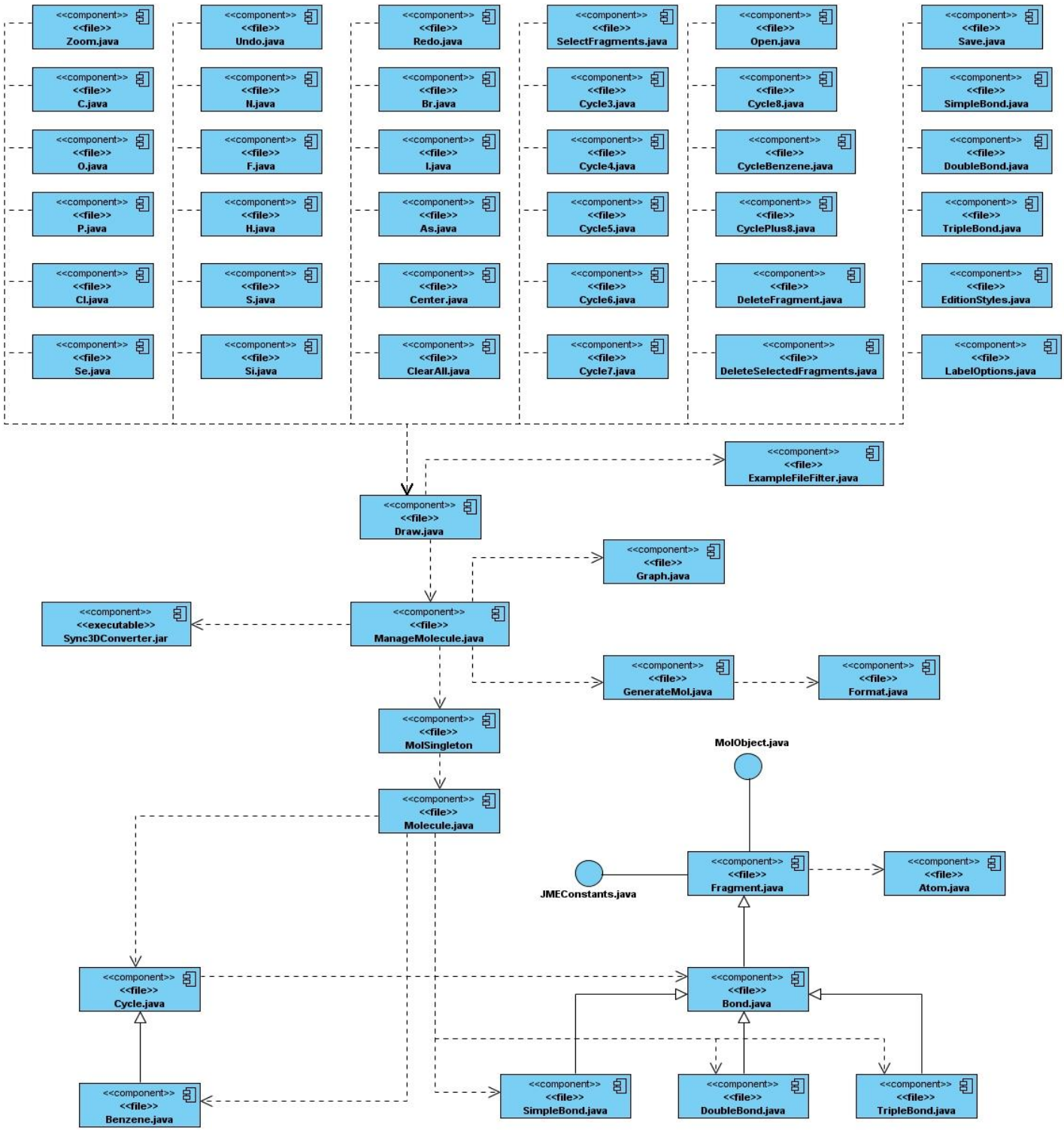
OPERACIONES	DESCRIPCIÓN
+Zoom(m : MainInterface)	Este método es el constructor de la clase.
+void actionPerformed(ActionEvent e)	

CLASE : ExampleFileFilter

ATRIBUTOS	TIPO
-typeUnknown	String
-hiddenFile	String
-filters	Hashtable
-description	String
-fullDescription	String
-useExtensionsInDescription	boolean

OPERACIONES	DESCRIPCIÓN
+ExampleFileFilter()	Este método es un constructor de la clase.
+ExampleFileFilter(extension : String)	Este método es un constructor de la clase.
+ExampleFileFilter(extension : String, description : String)	Este método es un constructor de la clase.
+ExampleFileFilter(filters : String[])	Este método es un constructor de la clase.
+ExampleFileFilter(filters : String[],description : String)	Este método es un constructor de la clase.
+accept(f : File) : boolean	Este método determina si un fichero es aceptado.
+getExtension(File f) : String	Este método devuelve la extensión de un fichero.
+addExtension(extension : String) : void	Este método adiciona una extensión.
+getDescription() : String	Este método devuelve la descripción.
+setDescription(description : String) : void	Este método establece la descripción.
+setExtensionListInDescription(b : boolean) : void	Este método establece si se usan extensiones en la descripción.
+isExtensionListInDescription() : boolean	Este método devuelve si se está usando o no extensiones en la descripción.

Anexo 2: Diagrama de Componentes Editor Molecular.


<<component>> <<file>> Zoom.java
<<component>> <<file>> Undo.java
<<component>> <<file>> Redo.java
<<component>> <<file>> SelectFragments.java
<<component>> <<file>> Open.java
<<component>> <<file>> Save.java
<<component>> <<file>> C.java
<<component>> <<file>> N.java
<<component>> <<file>> Br.java
<<component>> <<file>> Cycle3.java
<<component>> <<file>> Cycle8.java
<<component>> <<file>> SimpleBond.java
<<component>> <<file>> O.java
<<component>> <<file>> F.java
<<component>> <<file>> I.java
<<component>> <<file>> Cycle4.java
<<component>> <<file>> CycleBenzene.java
<<component>> <<file>> DoubleBond.java
<<component>> <<file>> P.java
<<component>> <<file>> H.java
<<component>> <<file>> As.java
<<component>> <<file>> Cycle5.java
<<component>> <<file>> CyclePlus8.java
<<component>> <<file>> TripleBond.java
<<component>> <<file>> Cl.java
<<component>> <<file>> S.java
<<component>> <<file>> Center.java
<<component>> <<file>> Cycle6.java
<<component>> <<file>> DeleteFragment.java
<<component>> <<file>> EditionStyles.java
<<component>> <<file>> Se.java
<<component>> <<file>> Si.java
<<component>> <<file>> ClearAll.java
<<component>> <<file>> Cycle7.java
<<component>> <<file>> DeleteSelectedFragments.java
<<component>> <<file>> LabelOptions.java
<<component>> <<file>> ExampleFileFilter.java
<<component>> <<file>> Draw.java
<<component>> <<file>> Graph.java
<<component>> <<executable>> Sync3DConverter.jar
<<component>> <<file>> ManageMolecule.java
<<component>> <<file>> GenerateMol.java
<<component>> <<file>> Format.java
<<component>> <<file>> MolSingleton
MolObject.java
<<component>> <<file>> Molecule.java
JMEConstants.java
<<component>> <<file>> Fragment.java
<<component>> <<file>> Atom.java
<<component>> <<file>> Cycle.java
<<component>> <<file>> Bond.java
<<component>> <<file>> Benzene.java
<<component>> <<file>> SimpleBond.java
<<component>> <<file>> DoubleBond.java
<<component>> <<file>> TripleBond.java

Anexo 3: Diseño de las pruebas por cada sección del caso de uso Crear Molécula.

SC 1: Crear Molécula.

Variables

1- Elemento: elem = Element.EnlaceSimple
2- Punto: p.x = 10 p.y = 20
3- Modo dibujo: mode = *DEFAULT_SELECTION*

Id del escenario	**Escenario**	**V1**	**V2**	**V3**	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 1.1	Crear Molécula	V	V	V	El sistema dibuja un Enlace Simple.	Se dibujó el Enlace Simple.

Anexo 4: Diseño de las pruebas por cada sección del caso de uso Modificar Molécula.

SC 2: Modificar Molécula.

Variables

1- File: f
2- Modo dibujo: mode = *DEFAULT_SELECTION*

Id del escenario	**Escenario**	**V1**	**V2**	**Respuesta del Sistema**	**Resultado de la Prueba**
EC 2.1	Modificar Molécula	V	V	El sistema dibuja una molécula que es leída de un fichero mol.	Se dibujó la molécula correctamente.

Anexo 4: Diseño de las pruebas por cada sección del caso de uso Crear Molécula.

SC 3: Cambiar forma de edición.

Variables

1- Modo dibujo: mode = *DEFAULT_SELECTION*
2- Opción dibujo: option = *NUMBER_OPTION*
3- Forma dibujo: formView = *BALLS_OPTION*

Id del	**Escenario**	**V1**	**V2**	**V3**	**Respuesta del Sistema**	**Resultado**

escenario						**de la Prueba**
EC 3.1	Cambiar forma de edición	V	V	V	El sistema cambia la forma de visualizar la molécula a bolas y números.	Se cambió la forma de visualización a formas y números.

GLOSARIO

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

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

Compuestos Orgánicos: Sustancias químicas basadas en cadenas de carbono e hidrógenos. En muchos casos contienen oxígenos, nitrógenos, azufres, fósforos y halógenos.

Especialista: Persona capacitada para interactuar con la aplicación.

Información Estructural: Dato o descripción que permite conocer cómo está formado el compuesto en términos de átomos y la relación espacial que existe entre ellos.

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

UCI: Universidad de las Ciencias Informáticas.