Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación
Centro de Computación Gráfica

Extensión de un Constructor Molecular para Química Computacional

Trabajo Especial de Grado
presentado ante la ilustre
Universidad Central de Venezuela
por el bachiller
Eleazar Ernesto Matheus Camacho
para optar al título de
Licenciado en Computación

Tutores
Dr. Ernesto Coto
M.Sc. Víctor Sojo

Caracas, Julio de 2012

Universidad Central de Venezuela
Facultad de Ciencias
Escuela de Computación

ACTA DEL VEREDICTO

Quienes suscriben, miembros del jurado designado por el Consejo de la Escuela de Computación para examinar el Trabajo Especial de Grado, presentado por el Bachiller Eleazar Ernesto Matheus Camacho, C.I. 16.972.090, con el título "Extensión de un Constructor Molecular para Química Computacional", a los fines de cumplir con el requisito legal para optar al título de Licenciado en Computación, dejan constancia de lo siguiente:

Leído el trabajo por cada uno de los miembros del jurado, se fijó el día 13 de Julio de 2012, a las 11:00 a.m., para que su autor lo defendiera en forma pública, en el Centro de Computación Gráfica, lo cual éste realizó mediante una exposición oral de su contenido, y luego respondió satisfactoriamente a las preguntas que les fueron formuladas por el jurado, todo ello conforme a lo dispuesto en la Ley de Universidades y demás normativas vigentes de la Universidad Central de Venezuela. Finalizada la defensa pública del Trabajo Especial de Grado, el jurado decidió APROBARLO.

En fe de lo cual se levanta la presente acta, en Caracas el 13 de Julio de 2012, dejándose también constancia de que actuó como coordinador del jurado el Dr. Ernesto Coto, tutor del trabajo. Igualmente se hace constar que actuó como tutor el M.Sc. Victor Sojo.

_____
Dr. Ernesto Coto
Escuela de Computación, UCV
(Tutor Firmante)

_____
Prof. Esmitt Ramírez
Escuela de Computación, UCV
(Jurado)

_____
Profa. Alecia Acosta
Escuela de Computación, UCV
(Jurado)

RESUMEN

La química computacional es una rama de la química que utiliza principios de la computación para resolver problemas químicos. La solución de problemas químicos ha beneficiado al hombre con la creación de productos que mejoran su calidad de vida. Para facilitar el estudio de la gran variedad de componentes en el universo y para no poner en peligro en algunos casos a los investigadores, la química computacional realiza cálculos sobre los modelos digitales de las moléculas de esos componentes, generados por sistemas interactivos llamados constructores moleculares. A medida que los modelos de moléculas han sido más fáciles de construir, los problemas químicos se han resuelto en menos tiempo, mejorando la calidad de vida del hombre más rápidamente.

Hoy en día existen institutos de investigación dedicados a resolver una gran variedad de problemas relacionados con la química computacional en el menor tiempo posible, desarrollando herramientas cada vez más avanzadas. Tal es el caso del IVIC (Instituto Venezolano de Investigaciones Científicas).

El Laboratorio de Química Computacional del IVIC genera con regularidad modelos digitales de moléculas para ser utilizados por programas de cálculo, por lo cual está buscando constantemente constructores moleculares cada vez más fáciles de aprender a utilizar que le permitan generar los modelos de una manera más eficiente.

Este trabajo presenta el desarrollo de *Jmol Builder*, una versión de *Jmol* con un constructor más complejo e intuitivo, accesible vía web, de código abierto, de fácil extensión y respaldado por una activa comunidad de desarrolladores, que contiene una amplia gama de funcionalidades útiles para el químico computacional. Fue desarrollado para ser un componente fundamental de una plataforma integradora web para química computacional llamada *IVIChem*.

La utilidad de *Jmol Builder* es demostrada con muy buenos resultados a partir de una serie de encuestas basadas en la escala de Likert respondidas por investigadores del IVIC. Además, la aplicación es conocida y apoyada por la comunidad de desarrolladores de *Jmol*.

Palabras Clave: Química computacional, Constructor molecular, Web, *Jmol*, *Java*, *IVIChem*.

DEDICATORIA

A mis padres, por creer en mí y por ayudarme a llegar al final de la meta, apoyándome en todo el transcurso de este trabajo. Sin sus excelentes consejos, hubiera sido muy difícil alcanzar este triunfo en mi vida. Realmente admiro su deseo de superación y entrega. Ellos han sido y seguirán siendo para mí una guía a lo largo mis días.

AGRADECIMIENTOS

La finalización de este trabajo no hubiese sido posible sin la ayuda de las personas que citaré a continuación.

A mis hermanos, por mejorar mis días con su buen humor, por ayudarme a descrubrir mi propósito y por hacerme creer que realmente tendré un buen futuro, impulsándome a ser cada día mejor.

A mis primos, por la alegría que me brindaron con su compañía y por hacer de mí una mejor persona a través de su ejemplo de dedicación y esfuerzo.

A mis tíos y abuelos, por preocuparse por mi bienestar, por sus consejos y por darme ejemplos dignos de triunfo sobre momentos difíciles.

A mis amigos y compañeros, por ser una fuente de energía cuando más la necesitaba, dándome excelentes recuerdos de felicidad.

A mis profesores, por instruirme, por sus valiosos comentarios y por hacer de mí un mejor profesional, digno de la Universidad Central de Venezuela.

A la comunidad de trabajadores del Servicio de Orientación de la Facultad de Ciencias de esta Institución, por ayudarme a descubrir mi verdadero potencial, por facilitarme el acceso a la información requerida para alcanzar mis objetivos y por haber fomentado en mí el anhelo de triunfo en la vida.

Finalmente, a todos aquellos que de una u otra manera me acompañaron en esta tesis.

Me siento profundamente agradecido por su apoyo.

TABLA DE CONTENIDO

INTRODUCCIÓN

La computación genera innumerables beneficios en todos los procesos humanos como la comunicación de datos, el acceso a información oportuna, la organización, los medios de almacenamiento masivo, entre otros [1]. Con el desarrollo de cada vez más programas y aplicaciones que, basándose en conceptos de abstracción, ofrecen el uso de interfaces multimedia agradables y de fácil comunicación, las computadoras se acercan más al público [2].

En los años 40 el uso de las computadoras hizo posible la solución de ecuaciones de onda para el desarrollo de sistemas complejos relacionados con la química [3]. La química computacional es una rama de la química que utiliza principios de computación para resolver problemas químicos. También utiliza programas eficientes que modelan conceptos de la química teórica y realizan cálculos sobre las estructuras y propiedades de las moléculas. Es común que los resultados de esos programas sólo complementen la información obtenida con experimentos químicos; sin embargo, en algunos casos predicen fenómenos químicos nuevos.

La química computacional es utilizada principalmente en el diseño de nuevos fármacos y materiales [3]. Gracias a ella los tiempos de investigación se han acortado [4].

La visualización de moléculas a través de imágenes ayuda a entender su estructura y función [5]; y permite que los investigadores avancen en el análisis de cómo funcionan las proteínas y otras moléculas en el cuerpo [6]. Cuando se trata con moléculas de gran escala, es aún más útil su visualización y control en tres dimensiones. Es por esa razón que desde hace mucho tiempo los químicos utilizan modelos moleculares físicos, que son instrumentos científicos construidos con pelotas de madera y varillas, o maquetas de modelos de llenado de espacio, que se han utilizado por más de 125 años para representar, comunicar y razonar sobre las leyes de la física que influyen sobre las estructuras químicas. Estos modelos podían ser costosos y difíciles de construir, sobretodo si se pensaba representar una macromolécula, que podía estar conformada por miles o millones de átomos.

El primer sistema interactivo de visualización de moléculas fue construido a principios del año 1965 por Cyrus Levinthal, un empleado del *MIT*. El programa de Levinthal le permitía al usuario dirigir un dibujo tridimensional del modelo de una molécula basado en líneas, con un globo de plástico que podía ser rotado o empujado en cualquier dirección para cambiar la velocidad y orientación de la rotación del modelo. Colocando un lápiz óptico en el área de visualización, el usuario podía seleccionar de un menú, elegir un objeto o ver más de cerca secciones importantes del modelo.

Desde que el programa de Levinthal fue construido, sistemas interactivos de visualización de moléculas gradualmente han sido incorporados en todas las áreas de la educación e investigación sobre química, reemplazando de esa forma a los modelos moleculares físicos como la manera estándar de representación. Esto ha transformado la manera como los investigadores de química construyen y usan los modelos moleculares.

Los modelos generados por computadora y las simulaciones están siendo utilizados para construir estructuras moleculares físicamente válidas, según la química teórica y la física clásica, que ayudan a predecir el comportamiento de dichas estructuras y al diseño de nuevos experimentos. Igualmente, los avances de hoy en visualización interactiva molecular también se han estado enfocando tanto en la interfaz entre el científico y el modelo molecular, como en la interfaz entre el *software* y los datos del modelo.

Actualmente existen sistemas interactivos que permiten visualizar, construir y navegar sobre simulaciones de modelos de moléculas, estudiando de esa manera la geometría y el comportamiento de esas estructuras, que reguralmente son complejas. Otros sistemas incorporan dispositivos de tacto capaces de permitirle al investigador, no sólo sentir fuerzas moleculares sino también empujar y halar átomos, alterando de esa forma el comportamiento de una simulación en tiempo real. Los más actuales, curiosamente, trabajan en conjunto con modelos moleculares físicos. Los investigadores los han utilizado como la interfaz de un *sistema háptico* interactivo de visualización de moléculas. La posición y la manera como el científico manipula el modelo es capturada y relacionada con el área de visualización del sistema en la computadora. El resultado es una excelente integración de los modelos físicos y virtuales utilizando los mejores atributos de los dos [7].

Sin embargo, los sistemas no hápticos siguen siendo extremadamente útiles. En la actualidad existe una gran variedad de aplicaciones de escritorio y aplicaciones web que pueden ser útiles para los químicos computacionales. Por supuesto, no todas estas aplicaciones tienen las mismas capacidades y herramientas. Algunas son sofisticadas pero muy costosas y de código propietario, mientras que otras son gratuitas y de código abierto, pero con pocas funcionalidades.

En este trabajo se desarrolló un constructor molecular web, para el químico computacional, de código abierto y de fácil extensión, que permite: generar modelos de moléculas con una barra de herramientas, agregar fragmentos que luego une automáticamente al modelo generado, seleccionar átomos de una tabla periódica interactiva, manipular átomos seleccionados con un control de direcciones, utilizar nuevos accesos directos de teclado, integrar el sistema con otro constructor llamado *JME* y trabajar con distintas resoluciones. El constructor desarrollado está siendo utilizado por investigadores del Laboratorio de Química Computacional del Centro de Química del Instituto Venezolano de Investigaciones Científicas (IVIC), para generar los datos de entrada de los programas de cálculo de química computacional que usan con regularidad.

Este trabajo está organizado de la siguiente forma: el Capítulo 1 describe las facilidades que la química computacional ofrece al área de investigación y enseñanza de la química tradicional, además de los primeros constructores moleculares relacionados con los que se utilizan en la actualidad; el Capítulo 2 hace referencia al planteamiento de este trabajo de investigación; el Capítulo 3 presentará el diseño y la implementación del constructor molecular que se desarrolló en este trabajo; las pruebas y los resultados obtenidos serán discutidos en el Capítulo 4. Por último, en el Capítulo 5 se describen las conclusiones y los trabajos a futuro.

CAPÍTULO I: QUÍMICA COMPUTACIONAL

1.1. Concepto y aplicación

La química es la ciencia que estudia la composición, estructura y propiedades de la materia, además de los cambios que esta experimenta durante las reacciones y su relación con la energía. Debido a la diversidad de la materia, que está compuesta de átomos, los químicos frecuentemente estudian cómo los átomos de diferentes elementos químicos de la tabla periódica interactúan para formar moléculas y cómo ellas mismas interactúan unas con otras.

Las primeras experiencias del hombre como químico se dieron con la utilización del fuego en la transformación de la materia. La obtención de hierro a partir del mineral y de vidrio a partir de arena son algunos de los ejemplos más claros. Poco a poco el hombre se dio cuenta de que otras sustancias también tienen este poder de transformación. Se dedicó con gran empeño a buscar una sustancia que transformara un metal en oro, lo que llevó a la creación de la alquimia. La acumulación de experiencias alquímicas jugó un papel vital en el futuro establecimiento de la química.

La química es una ciencia empírica ya que estudia la materia por medio del método científico, es decir, por medio de la observación, la cuantificación y, sobre todo, la experimentación. En su sentido más amplio, la química estudia las diversas sustancias que existen en nuestro planeta así como las reacciones que las transforman en otras sustancias. Por otra parte, la química estudia la estructura de las sustancias a su nivel molecular y sus propiedades [8] [9].

El estudio de una sustancia depende de la complejidad de sus componentes y la cantidad a estudiar. Si se busca estudiar las propiedades de una molécula de pocos átomos, sus propiedades se pueden obtener resolviendo ecuaciones básicas de química cuántica. Para moléculas más grandes, las ecuaciones de química cuántica dejan de ser prácticas. En ese caso, se utilizan principios de la física clásica para investigar su estructura y propiedades [10].

Desde la segunda mitad del siglo 20 la química ha sido una de las áreas en las que se han usado computadoras para investigaciones científicas. Varios campos de la química han ido incrementando el uso de computadoras y técnicas computacionales en la realización de sus investigaciones.

Hoy en día, se ha llegado a un punto donde, en muchos casos, el químico puede sustituir el tubo de ensayo por la computadora. Eso no significa que el enfoque con computadoras se deba considerar como un rival de las técnicas tradicionales. Usualmente, los dos enfoques son complementarios, uno facilitando información que el otro no puede. A veces un experimento puede ser muy peligroso, o hasta imposible de realizar en un laboratorio. Los métodos de la química computacional

entonces pueden ser la única manera de obtener la información química deseada. Por ejemplo, es bastante difícil trabajar con compuestos de berilio en un laboratorio debido a una toxicidad que no se debe subestimar. En cambio, el átomo de berilio se puede manipular fácilmente con métodos de la química cuántica computacional de gran exactitud ya que contiene sólo 4 electrones. Muchos no pueden ser estudiados con técnicas estándares de laboratorio, como estereoscopía en infrarrojo y resonancia magnética nuclear, debido a que reaccionan muy rápidamente como para ser aislados. Sin embargo, el estudio de compuestos inestables con la química computacional usualmente no es muy difícil de realizar [10].

La química computacional emplea herramientas muy diferentes a las de la química tradicional y, muy frecuentemente, los datos le son suministrados, seleccionándolos de bases de datos existentes. Las características de moléculas y átomos son suministradas mediante modelos de barras, enlaces, esferas y puntos. Frecuentemente, programas computacionales adecuados, basados en los modelos de la mecánica cuántica, proporcionan datos suficientes para que los químicos formulen respuestas en términos de la química tradicional.

Además, la química computacional ofrece un espectro de utilización muy amplio. Por ejemplo, se pueden hacer cálculos de modelado y predicción de estructuras nuevas de sistemas orgánicos e inorgánicos. Entre estos sistemas están incluidos: metales, macromoléculas, proteínas, policristales, moléculas de actividad farmacológica, moléculas de actividad relacionada con la capacidad de respuesta a excitaciones, estímulos o causas muy pequeñas; etc. Con la química computacional se puede estudiar desde la explicación microscópica de propiedades físicas de un metal dado hasta los enlaces químicos que determinan las funciones de moléculas que sustentan la vida.

Entre los objetivos de muchas investigaciones de importancia actual se encuentran la búsqueda de nuevas moléculas que puedan contar con primicia en el mercado y el estudio de los mecanismos de acción de estos componentes a nivel molecular a través de la farmacología molecular, la inmunología molecular, los nuevos métodos de síntesis y la modelación matemática.

Los métodos de la química computacional constituyen herramientas de gran potencia y efectividad en el diseño de fármacos, la elucidación de la estructura de moléculas de alta complejidad, y de los mecanismos de reacción, con la ventaja de ser muy baratos y generalmente rápidos, por demás de no generar daños al medio ambiente. Esta diversidad permite incentivar su uso y la introducción de un conocimiento general básico, útil para profesionales de cualquier área de la química, la biología, la farmacología, la ingeniería de materiales y la medicina, entre múltiples otras disciplinas.

La química computacional también está llamada a cambiar el enfoque mecánico clásico determinista en la enseñanza, fomentando la adquisición de habilidades de

abstracción, diseño de experimentos y representación molecular en los alumnos [11].

La rama de la química que se ha estado describiendo realiza además cálculos de diversas propiedades sobre modelos de moléculas. El rol del constructor es generar las distribuciones tridimensionales de átomos en el espacio. Sería casi imposible o sumamente dificultoso para una persona generar esas coordenadas a mano, y la posibilidad de cometer errores sería enorme.

1.2. Visualizadores y constructores moleculares

Los visualizadores y constructores moleculares son aplicaciones para química computacional desarrolladas para ahorrar a los investigadores tiempo valioso y recursos económicos.

La investigación de los visualizadores y/o constructores moleculares que se describen en este apartado contribuye a la recopilación de las ventajas y las desventajas que tenían los primeros constructores moleculares, y a la explicación de la necesidad de algunas funcionalidades de los constructores moleculares existentes.

Los constructores moleculares de interés para este trabajo son constructores accesibles vía web, además de relacionados con constructores existentes gratuitos y de código abierto. Esto se debe a que se quieren constructores que puedan ser modificados fácilmente en el futuro según las necesidades de los investigadores, que no sean propietarios para evitar el pago de una licencia de alto costo, y que puedan utilizarse con la participación remota de investigadores en distintos sectores dentro y fuera del país.

La recopilación de las principales funcionalidades de las aplicaciones que se describen a continuación determinará cuál de ellas será incorporada a la solución del problema que se plantea en la propuesta de este trabajo.

1.2.1. WWW Molecular Editor

Como la construcción y edición de moléculas es indispensable para sistemas de información química, y en 1994 no existía una herramienta de ese estilo disponible para la *World Wide Web* (*WWW*), la organización *Molinspiration* decidió desarrollar su propio editor molecular web. El editor tenía como base un mapa interactivo que se muestra en la Fig. 1.

El *WWW Molecular Editor* permitía la construcción sencilla de moléculas orgánicas en 2D agregando átomos, lazos, anillos y grupos funcionales; y conectándolos entre sí. El usuario podía seleccionar la acción deseada en un

menú, y luego escoger el lugar apropiado en el área de dibujo. Todo el procesamiento real se realizaba en segundo plano en el servidor, adonde se enviaban las coordenadas del punto elegido. El programa en el servidor realizaba los cambios solicitados y retornaba una imagen *GIF* de la estructura modificada. La desventaja de este enfoque era que el programa requería una nueva conexión con el servidor por cada cambio en la estructura.

La creación de moléculas era sencilla, y este editor web simple era utilizado y recomendado por químicos alrededor del mundo [12].

WWW Molecular Editor es el predecesor de *JME*, otro constructor que se describirá más adelante.

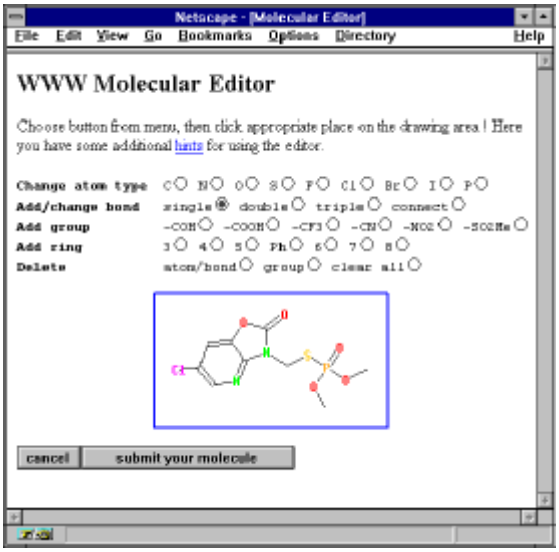


Figura 1: *WWW Molecular Editor*

1.2.2. Xmol

Es una utilidad desarrollada por el Centro de Supercomputadoras de *Minnesota*, para crear y visualizar imágenes gráficas de las moléculas. Aunque los gráficos producidos por *Xmol* son de baja calidad, ofrece muchas características, tales como la capacidad para crear animaciones para visualizar caminos de reacción o vibraciones moleculares. *Xmol* reconoce muchos formatos de entrada, incluyendo archivos de entrada y de salida para el popular paquete de química computacional, *Gaussian* [13].

Xmol permite a los investigadores y estudiantes ver modelos moleculares 3D producidos por otros paquetes de *software*, e imprimir copias de visualización de moléculas en *Encapsulated PostScript*. Los modelos moleculares pueden ser trasladados y rotados en una variedad de formas. Es posible crear animaciones de

archivos de datos de múltiples pasos, al igual que los cálculos de las distancias entre átomos, ángulos de enlace y ángulos de torsión. Hay disponibilidad de módulos de edición, permitiendo al usuario cambiar el tipo y la carga de átomos individuales [14].

Aunque se distribuyeron los programas ejecutables, el código fuente de *Xmol* no estaba disponible para los usuarios y, puesto que el programa no se ha mantenido, las versiones binarias gratuitas se han vuelto obsoletas. El abandono de *Xmol* dejó una necesidad de una herramienta similar que pudiera construir modelos de moléculas. A diferencia de Xmol, esa herramienta sería de gran interés para este trabajo.

1.2.3. ChemWriter

Es un editor de estructuras químicas en 2D, diseñado para trabajar con aplicaciones web. Existen dos versiones: un *applet* de *Java*, que se muestra en la Fig. 2, y un editor en línea. Es rápido en cargar información y es fácil de aprender para usuarios nuevos. *ChemWriter* hace que sea sencillo crear interfaces ideales para aplicaciones químicas que manejen grandes volúmenes de datos. Es utilizado por organizaciones alrededor del mundo, y puede verse en funcionamiento en páginas web populares [15].

Aunque existe una versión gratuita y totalmente funcional para ser utilizada durante el desarrollo y la prueba de aplicaciones web propias, se requiere pagar para obtener su licencia y no es de código abierto [16].

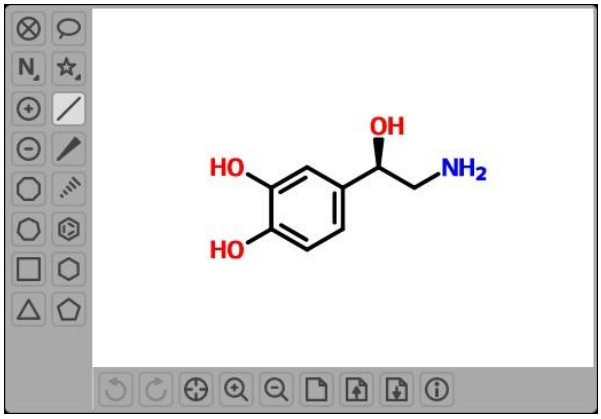


Figura 2: *ChemWriter*

1.2.4. JChemPaint

JChemPaint (*JCP*) es un editor y visualizador gratuito de estructuras químicas en 2D, de código abierto y escrito en *Java* [17]. Por esa razón corre en sistemas

Windows, *Mac OS X*, *Linux* y *Unix*. Existe una aplicación *Java* (editor) y dos versiones como *applets* de *Java* (editor y visualizador) que pueden ser integradas en páginas web.

El *applet JChemPaint Editor* le permite al usuario dibujar estructuras químicas, importar y exportar datos sobre esas estructuras en formatos de texto plano como: *SMILES*, *Molfile* y *CML*. También permite la carga y despliegue de una estructura química, que luego puede ser editada por el usuario. La Fig. 3 es una captura de pantalla del editor.

El *applet JChemPaint Viewer* sólo le permite al usuario visualizar la estructura química mas no editarla.

La aplicación *JChemPaint*, escrita en *Java*, es independiente de las páginas web. Ofrece todas las facilidades de *JCP Editor*.

JCP es un programa muy útil para la construcción y visualización de estructuras moleculares. Sin embargo, no calcula ni la distancia ni el ángulo entre los átomos de las estructuras que recibe de entrada, ni de las que construye.

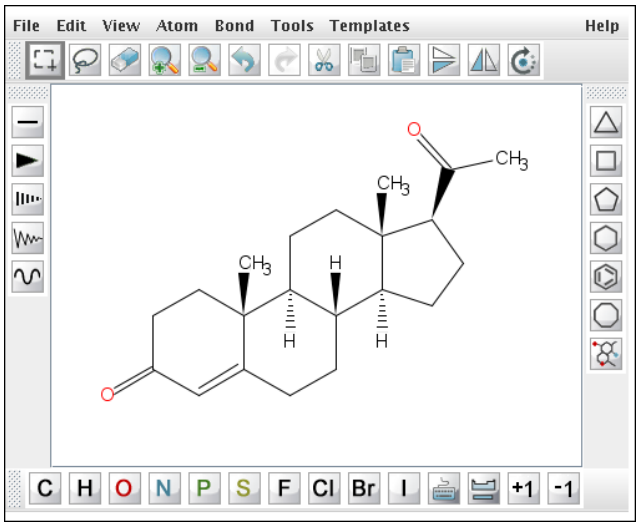


Figura 3: *JChemPaint Editor*

1.2.5. JsMolEditor

JsMolEditor es un editor molecular en línea que representa estructuras moleculares en dos dimensiones. El editor se muestra en la Fig. 4. Es diferente a todos los *plugins*, ya que tiene la peculiaridad de ser el primero en ser desarrollado exclusivamente en *JavaScript*. Por tanto es mucho más fácil de instalar y utilizar que la versión *applet* de *JCP*, porque no es necesaria la instalación de un sistema de tiempo de ejecución. Funciona en todos los navegadores modernos como: *Firefox*, *Safari*, *Google Chrome*, *Opera* e *Internet Explorer*; y en todas las

plataformas. Además, a diferencia de *ChemWriter*, es de código abierto y es gratuito, pero tiene la desventaja de permitir únicamente la visualización y edición de estructuras moleculares en dos dimensiones [18].

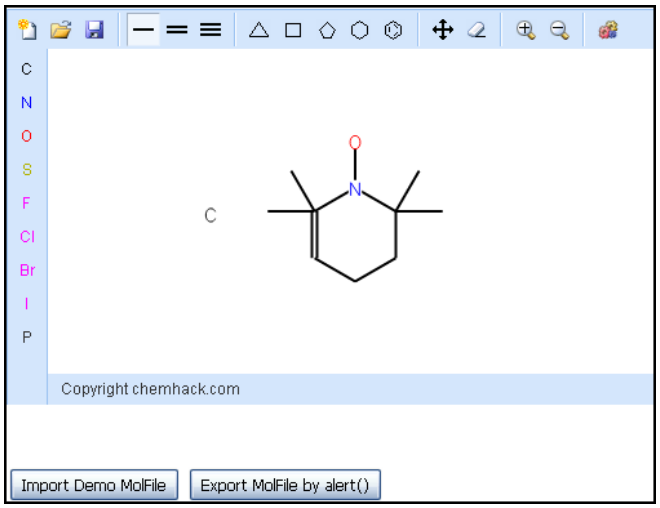


Figura 4: *JsMolEditor*

1.2.6. Java Molecular Editor (JME)

El editor molecular *JME* es un *applet* de *Java* que permite dibujar y editar moléculas y reacciones (incluyendo la generación de consultas sobre parte de la estructura molecular) y representar moléculas directamente en una página *HTML* en 2D (ver Fig. 5). El editor puede generar archivos *Daylight SMILES* o *MDL Molfile* de las estructuras creadas.

Debido a un gran número de solicitudes, el *applet* ahora es disponible a todo público y se convirtió en un estándar en cuanto a la entrada de estructuras moleculares en la *Internet* alrededor del mundo.

JME tiene algunas desventajas. No calcula ni la distancia ni el ángulo entre los átomos de las estructuras que recibe de entrada, ni de las que construye y, a diferencia del editor *JCP*, no es de código abierto [19].

1.2.7. MarvinSketch

MarvinSketch, de la compañía *ChemAxon*, es un editor avanzado para dibujar estructuras químicas, consultas (*queries*), y reacciones en 2D, desarrollado en *Java*. La Fig. 6 es una captura de pantalla de la aplicación. Tiene una gran lista de herramientas para editar, la cual sigue creciendo, y es *químicamente consciente*, ya que le permite a los usuarios utilizar funciones químicas que realizan cálculos basados en estructuras químicas llamadas directamente desde la aplicación;

verifica errores en las valencias y en las reacciones; su diseño permite búsquedas en estructuras químicas; tiene implementados: isótopos, cargas de radicales, pares solitarios y grupos R; permite etiquetar, ya sea manualmente o automáticamente, los átomos involucrados en una reacción, y permite el uso de funciones de estereoquímica avanzada como enlaces dobles *E/Z*.

Los cálculos que realiza sobre las estructuras químicas los hace con ayuda de *plugins* de cálculo de *ChemAxon*. Existe una versión independiente de los navegadores y otra que es un *applet* [20]. Aun así, es un *software* propietario. Por tanto sólo las compañías de gran envergadura pueden utilizarlo.

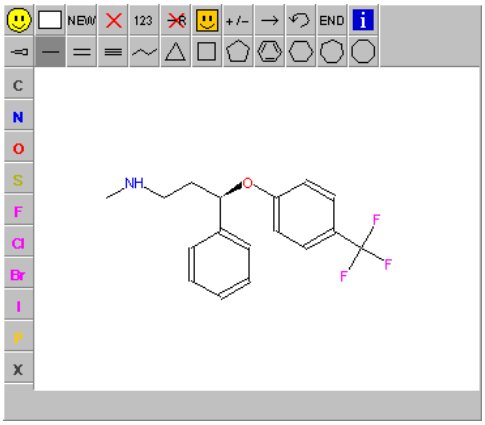


Figura 5: Editor de Moléculas de *Java* (*JME*)

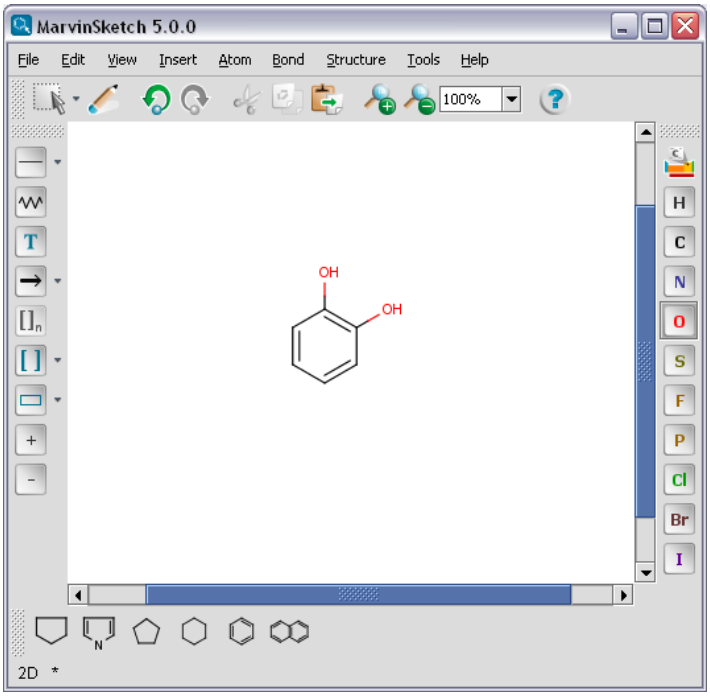


Figura 6: *MarvinSketch*

1.2.8. PubChem

Es una base de datos de moléculas, que es operada y mantenida por el *National Center for Biotechnology Information* (*NCBI*), parte de la *National Library of Medicine* (Biblioteca Nacional de Medicina), y parte a su vez de los *National Institutes of Health* (Institutos Nacionales de Salud) de Estados Unidos. Se puede consultar gratuitamente a través de Internet, además que permite descargar los datos de millones de estructuras vía *FTP*.

Tiene un sistema propio de edición de moléculas en dos dimensiones basado en los formatos químicos más comunes para representar estructuras químicas en bases de datos digitales: *SMILES*, *SMARTS* e *International Chemical Identifier* (InChl) de la organización *IUPAC*; permitiendo así su importación y exportación para realizar búsquedas flexibles y rápidas de estructuras y fragmentos en la base de datos [21]. También permite la importación de archivos *PDB* (*Protein Data Bank*), que son archivos que contienen información sobre estructuras de proteínas, ácidos nucleicos o ensamblajes complejos descubiertos experimentalmente por biólogos y bioquímicos de todo el mundo. Sin embargo, el editor de *PubChem* no permite la exportación de archivos *PDB* [22].

PubChem permite trabajar con una gran variedad de elementos de la tabla periódica como: metales alcalinos, metales alcalino-térreos, la familia del Boro, carbonoides, nitrogenoides, anfígenos, halógenos, gases nobles, algunos elementos de transición, algunos elementos de la serie lantánida y algunos elementos de la serie actínida [23]. El constructor se muestra en la Fig. 7.

El sistema es gratuito pero no es de código abierto. Además, sólo muestra las moléculas en 2D. Para que un investigador químico pueda realizar cálculos sobre las moléculas se necesitan las coordenadas tridimensionales de sus átomos. Como el constructor de este trabajo debe generar datos de entrada para programas de cálculo, la aplicación que se considere para ser extendida debe permitir la visualización de moléculas en tres dimensiones [22].

1.2.9. Ascalaph Designer

Es un paquete gratuito de modelado molecular de propósito general para el diseño y simulación de moléculas en tres dimensiones (ver Fig. 8). Proporciona un entorno gráfico para los programas comunes de modelado cuántico y clásico de moléculas. Realiza cálculos sobre las estructuras moleculares, tales como la distancia y el ángulo entre pares de átomos [24]. Además permite optimizar la estructura del modelo.

Sin embargo, no es de código abierto. Aunque lo fuera, no es intuitivo y el tiempo de respuesta de la versión para *Windows*, en una computadora con un procesador *Core 2 Duo* de 2.10 *GHz*, es bastante lento.

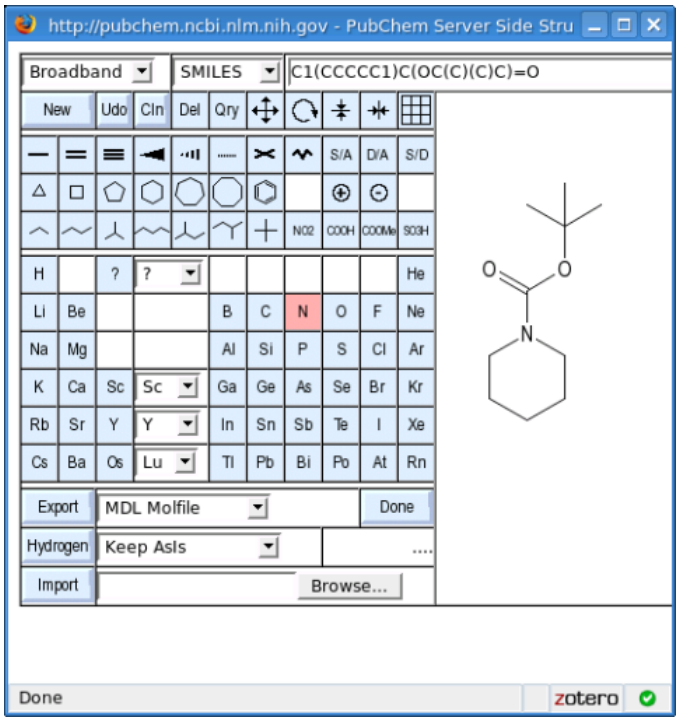


Figura 7: *PubChem*

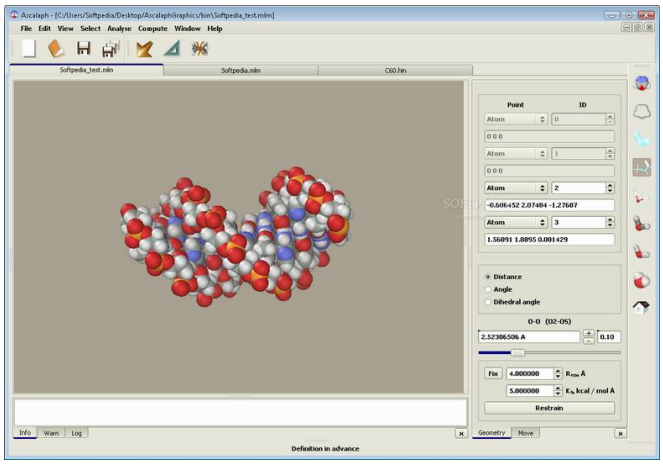


Figura 8: *Ascalaph Designer*

1.2.10. Avogadro

Es un editor molecular avanzado diseñado para un uso multiplataforma en química computacional, modelado molecular, bioinformática, ciencias de los materiales y áreas relacionadas. La Fig. 9 es una captura de pantalla de la aplicación. Ofrece una representación flexible y una arquitectura de *plugins* para desarrolladores que incluye representación, herramientas interactivas, comandos y *scripts* de *Python*. Es gratuito y de código abierto, y fue desarrollado para ser utilizado con facilidad por estudiantes e investigadores avanzados. Ofrece la

posibilidad de optimizar la geometría de la estructura importada o construida y calcular la distancia entre átomos, el ángulo plano entre pares de átomos y el ángulo diedro entre 4 átomos. Tiene una biblioteca que puede ser utilizada en programas desarrollados en *C++* y *Python*, y puede ser extendido con *plugins* y *scripts* [25]. Además, permite la selección de una gran variedad de fragmentos para ser agregados al modelo, e incluye una funcionalidad llamada "Editor Cartesiano" con la que se puede modificar las coordenadas de cualquier átomo de la estructura.

Sin embargo, no se ejecuta en páginas web y, para modificar su código fuente, se requiere un conocimiento sobre una gran variedad de aplicaciones, entre las cuales se encuentran: *Git*, *CMake*, *Qt4*, *Eigen2* y *OpenLabel*. Por tanto, se necesitaría mucho tiempo para modificar *Avogadro* a las necesidades de este trabajo [25].

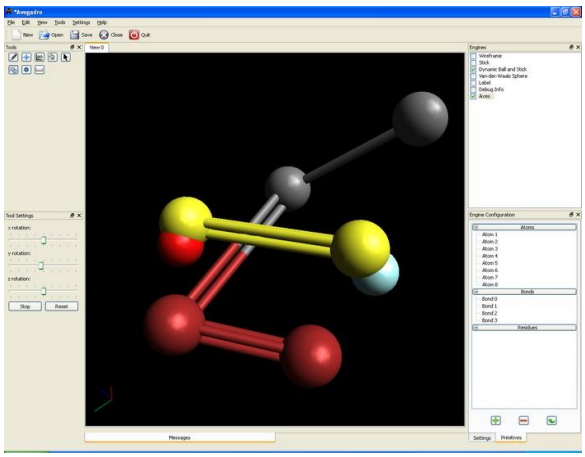


Figura 9: *Avogadro*

1.2.11. BALLView

Es una herramienta gratuita que permite visualizar y editar moléculas en 3D, como se puede observar en la Fig. 10. Provee una visualización rápida de estructuras moleculares basada en *OpenGL*, métodos de mecánica molecular (minimización de energía, simulación dinámica molecular utilizando *AMBER*, *CHARMM* y campos de fuerza *MMFF94*), edición de moléculas, además de cálculo y visualización de propiedades electroestáticas (*FDPB*) [26].

BALLView es una aplicación de escritorio y también es un *framework* para el desarrollo de visualización molecular. Es de código abierto, disponible bajo la licencia *GPL* [27] para *Linux*, *Windows* y *Mac OS X*.

Una opción de la aplicación accede a una extensa documentación, parecida a la ayuda de *Microsoft Office*. Ofrece la posibilidad de crear animaciones y el tiempo de respuesta es rápido en comparación con *Ascalaph Designer*. A diferencia de

Avogadro, sólo se necesita conocimiento sobre *C++* y *OpenGL* para modificar el código fuente de la aplicación.

BALLView no fue incorporado a la solución del problema que se plantea en la propuesta de este trabajo porque el uso de la aplicación no es intuitivo, especialmente durante la construcción de una molécula, razón por la cual se necesita consultar la documentación con más frecuencia de lo necesario. Además, no se ejecuta en páginas web [26].

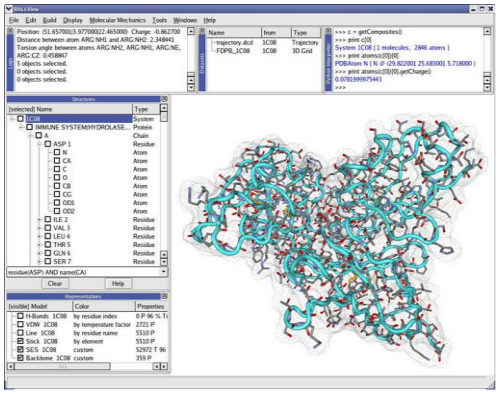


Figura 10: *BALLView*

1.2.12. Jmol

Jmol fue concebido originalmente como un sustituto plenamente funcional de *Xmol*. Fue creado procurando evitar los mismos problemas de *Xmol*, haciéndolo de código abierto. Además, actualmente existe una comunidad de usuarios de *Jmol* que crearon y mantienen un sitio web donde se encuentra recopilada una gran cantidad de información de la aplicación [28]. También existen listas de difusión por correo de usuarios y desarrolladores que intercambian ideas y experiencias, piden ayuda, colaboran con comentarios y sugerencias, y mucho más [29]. Aun así *Jmol* no ha cumplido por completo el objetivo de reemplazar funcionalmente a *Xmol*, pero reproduce muchas de sus prestaciones más útiles e incluso tiene el potencial para superar la funcionalidad de *Xmol* por ser de código abierto [30]. La aplicación incluye módulos para leer una variedad de tipos de archivos y la producción de programas de química cuántica, animación de archivos de múltiples fotogramas y modos de cálculo de normales a través de programas cuánticos [31]. La Fig. 11 es una captura de pantalla de la aplicación.

A diferencia de las aplicaciones que se han mencionado anteriormente, *Jmol* cumple con los requisitos necesarios para desarrollar un constructor molecular para química computacional que cumpla con las necesidades de este trabajo por razones que se explicarán en el planteamiento del Trabajo Especial de Grado.

1.2.13. Comparación

A continuación se muestra una tabla comparativa entre todas las aplicaciones que se han mencionado hasta ahora. Las columnas de la tabla corresponden a características importantes de las aplicaciones descritas. El color gris indica que la aplicación correspondiente cuenta con dicha característica.

	Visualizador	Editor	Gratuita	Código abierto	Trabaja con Reacciones	Calcula ángulos entre átomos	3D	Fácilmente Extensible	Web	Exporta Molfile	Importa Molfile	Plantillas	Tabla periódica	Portable
WWW Molecular Editor	■	■							■					
Xmol	■		■		■	■	■							
ChemWriter	■	■	■						■	■		■		■
JChemPaint	■	■	■	■				■	■	■	■	■	■	■
JsMolEditor	■	■	■	■				■	■	■	■	■		■
JME	■	■	■		■				■	■	■	■		■
MarvinSketch	■	■			■	■			■	■	■	■	■	■
PubChem	■	■	■						■	■	■	■	■	
Ascalaph Designer	■	■	■	■	■	■	■					■	■	■
Avogadro	■	■	■	■		■	■			■	■	■	■	■
BALLView	■	■	■	■	■	■	■	■			■	■	■	■
Jmol	■	■	■	■	■	■	■	■	■	■	■			■

Tabla 1: Tabla de comparación entre los visualizadores y constructores moleculares descritos

Páginas web: http://www.molinspiration.com (*WWW Molecular Editor*), http://www.hpc.unm.edu (*Xmol*), http://metamolecular.com (*ChemWriter*), http://sourceforge.net (*JChemPaint*), http://chemhack.com (*JsMolEditor*), http://www.molinspiration.com (*JME*), http://www.chemaxon.com (*MarvinSketch*), http://pubchem.ncbi.nlm.nih.gov (*PubChem*), http://www.biomolecular-modeling.com (*Ascalaph Designer*), http://avogadro.openmolecules.net (*Avogadro*), http://www.ball-project.org (*BALLView*), http://jmol.sourceforge.net (*Jmol*)

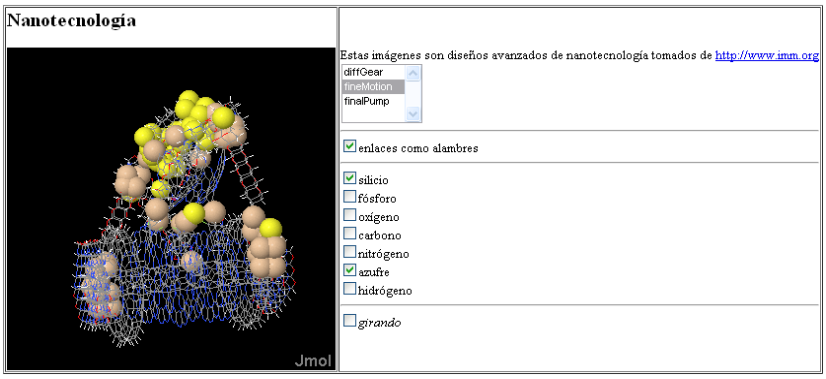


Figura 11: *Jmol*

CAPÍTULO II: PLANTEAMIENTO DEL TRABAJO ESPECIAL DE GRADO

2.1. Planteamiento del problema

El Laboratorio de Química Computacional del Centro de Química del Instituto Venezolano de Investigaciones Científicas (IVIC) necesitaba un constructor molecular para química computacional que cumpliera con las siguientes características: gratuito, de código abierto, ejecutable en páginas web y capaz de generar coordenadas tridimensionales.

A pesar de que ya existían aplicaciones que cumplían con algunos de estos requerimientos, la mayoría eran propietarias y por lo tanto involucraban el pago de una costosa licencia. Respecto a aquellas que eran gratuitas, no eran lo suficientemente poderosas, no generaban modelos tridimensionales o no eran de código abierto. Eso imposibilitaba la extensión de dichas aplicaciones, para ejecutar nuevas simulaciones y experimentos.

Luego, el problema que se deseaba resolver en este TEG era desarrollar un nuevo constructor molecular o extender alguno de los existentes, para que contara con las principales funcionalidades de los constructores actuales y que fuera de fácil extensión en el futuro, para satisfacer las necesidades presentes y futuras del Laboratorio de Química Computacional del IVIC, respecto de la simulación y experimentaciones con nuevos compuestos. Igualmente era importante que la solución propuesta fuera accesible vía web, para lograr la participación remota de investigadores en otras localidades del país y el mundo, a través de la implementación en *IVIChem*, la plataforma integradora.

2.2. Objetivo general

Desarrollar o extender un constructor molecular para química computacional que se ejecute en páginas web.

2.3. Objetivos específicos

- Recopilar las principales funcionalidades de los constructores y/o visualizadores moleculares más conocidos, para seleccionar cuáles de ellos serán incorporados a la solución del problema.
- Estudiar la posibilidad de extender alguno de los constructores y/o visualizadores de código abierto más conocidos, para solucionar el problema.
- De no ser posible conseguir modificar alguno de los constructores y/o visualizadores existentes hasta obtener el constructor molecular adecuado, diseñar y desarrollar un nuevo constructor.

- Ajustar el nuevo constructor en función de las necesidades del grupo de investigación del Laboratorio de Química Computacional del IVIC.
- Realizar pruebas de la solución, tanto vía remota como local, y realizar ajustes de ser necesario.

2.4. Solución propuesta

Después del estudio realizado a los constructores moleculares que se presentaron en el capítulo anterior, *Jmol* fue el constructor molecular que se consideró para ser extendido, debido a que:

1. Es una aplicación gratuita y totalmente funcional para ser utilizada durante el desarrollo y la prueba de aplicaciones web propias.
2. Es de código abierto.
3. Trabaja con reacciones.
4. Realiza cálculos sobre las estructuras moleculares, tales como la distancia y el ángulo entre pares de átomos, información de gran relevancia para los investigadores.
5. Permite la visualización de estructuras moleculares en tres dimensiones.
6. Está desarrollado con el lenguaje de programación *Java*, lo que lo hace multiplataforma, y además depende de la ejecución de *applets* estándar soportados por los navegadores más comunes.
7. Se ejecuta en páginas web.

Una vez comprobado que *Jmol* era el constructor molecular ideal, se propuso reunirse con el personal del IVIC periódicamente, recibiendo retroalimentación de los usuarios de la aplicación, de tal forma que ellos guiaran el desarrollo del constructor y así se satisficieran las necesidades del Laboratorio de Química Computacional del Centro de Química del IVIC.

Después de varios meses, se llegó a la conclusión de que *Jmol*, efectivamente, podía ser extendido, incluso con cierta facilidad. Resultó que el programa ya contenía un constructor molecular básico, aunque con una interfaz poco amigable, basada en un menú emergente.

En los próximos capítulos se describe en detalle el desarrollo de *Jmol Builder*, la extensión de *Jmol* desarrollada durante este TEG como solución al problema planteado.

CAPÍTULO III: DISEÑO E IMPLEMENTACIÓN

La mayoría del tiempo de desarrollo invertido en *Jmol Builder* consistió en la adición de nuevas clases y funcionalidades, siempre con la idea en mente de que *Jmol Builder* fuera separable de *Jmol*. Esto es, que la jerarquía de clases de *Jmol* no se viera afectada por la incorporación de las nuevas clases. De esta forma, el usuario podría ejecutar *Jmol Builder*, pero también podría ejecutar la versión estándar de *Jmol* si así lo desea. Para ello, se decidió incorporar a *Jmol* una barra de herramientas con la mayoría de las funcionalidades más usadas en el sistema original, además de algunas herramientas totalmente nuevas y útiles, que ahora ayudan a construir un modelo con mayor facilidad.

Las nuevas funcionalidades que pertenecen a *Jmol Builder* son: la barra de herramientas, nuevos accesos directos de teclado, nuevos *scripts* para la integración con *JME*, la posibilidad de trabajar con distintas resoluciones, una tabla periódica interactiva, un controlador de rotación y traslación de átomos seleccionados; y una herramienta que toma un fragmento seleccionado y lo une al modelo automáticamente.

3.1. Plataforma de Hardware y Software

El *hardware* que se empleó para la implementación del constructor molecular consistió en computadoras de escritorio comunes sin requerimientos particulares de *software* o *hardware*.

Existen varias versiones de *Jmol* disponibles para ser modificadas. En este trabajo se extendió la versión 12.1.50 [32].

Jmol fue extendido utilizando un ambiente de desarrollo integrado o *IDE* (*Integrated Development Environment*), de código abierto y multiplataforma, llamado *Eclipse* [33]. La comunidad de *Jmol* recomienda *Eclipse* porque: es fácil de instalar, tiene varias herramientas requeridas por *Jmol*, es un *IDE* poderoso, y *Jmol* fue configurado como un proyecto de *Eclipse*. Actualmente existen varias versiones. La que se utilizó para este trabajo fue la 3.5.2, aunque debería funcionar en versiones de la 3.1 en adelante [28].

Se instaló el *plugin* para *Eclipse*, *Subversive SVN Team Provider* 0.7.8, junto con el *plugin Subversive SVN Connectors* 2.2.1, de *Polarion Software* [33]. Gracias a la nueva funcionalidad, *Connector Discovery*, implementada a partir de la versión 0.7.8 de *Subversive SVN Team Provider*, el proceso de instalación fue mucho más sencillo [34].

El lenguaje de programación empleado fue *Java*, por tanto la aplicación es compatible con múltiples plataformas e independiente del sistema operativo, que

puede ser *Microsoft Windows*, *Mac OS X* o *Linux*. La versión que se utilizó del *JDK* o *SDK* de *Java*, fue *Java SE 6 Update 26* [28].

3.2. Modelo de Requerimientos

Jmol Builder originalmente fue pensado para agregar un constructor molecular a *Jmol*, que sería utilizado con una barra de herramientas. La barra ofrecería herramientas para rotar, mover y posiblemente ver más de cerca o reemplazar átomos del modelo; para seleccionar átomos de una tabla periódica interactiva; y para indicar el átomo actualmente seleccionado y una lista de los átomos más usados: C, N, O, F, P, S, Cl, Br e I. La Fig. 12 muestra el primer bosquejo de la aplicación.

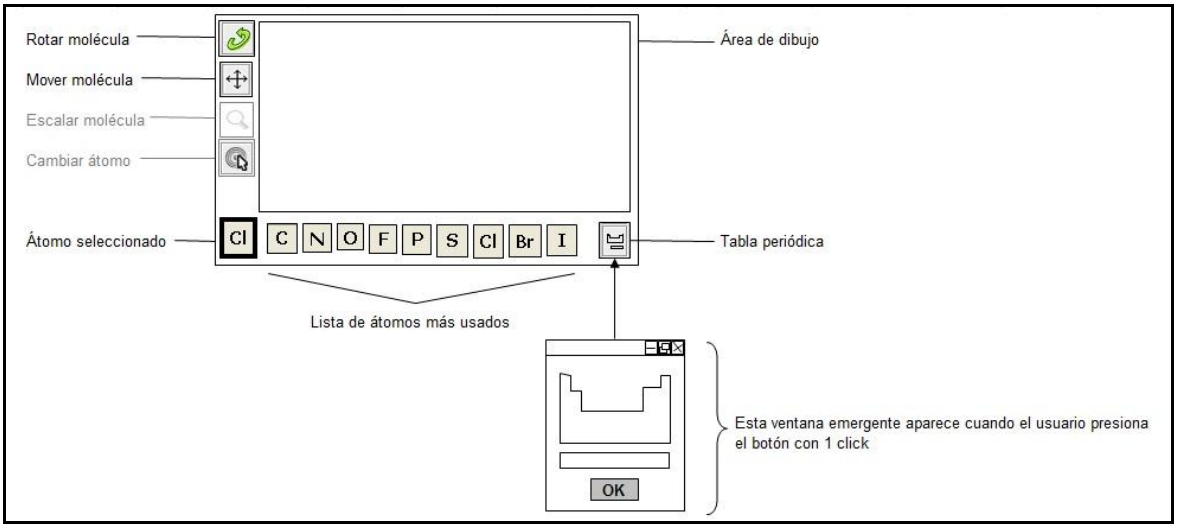


Figura 12: Primer bosquejo de *Jmol Builder*

El primer diagrama de casos de uso de la aplicación mostraba que con *Jmol Builder* el usuario debía poder: añadir un átomo seleccionándolo desde una tabla periódica o una lista de elementos más usados, cambiar entre un modo de construcción y uno de visualización, y mover o rotar la cámara (ver Fig. 13).

Un análisis más detallado de la aplicación permitió identificar que la aplicación ya tenía implementado un constructor molecular básico llamado "*Model Kit*", aunque con una interfaz poco usable basada en un menú emergente. Se decidió entonces trabajar en mejorar el constructor de *Jmol* agilizando el proceso de modelado con una herramienta que agregaba fragmentos al modelo, un control de átomos seleccionados, y nuevas funcionalidades asociadas al teclado. Para evitar sobrecargar la interfaz de usuario, los átomos más usados fueron reducidos a tres: C, O y H. En función de estas modificaciones, el diagrama de casos de uso original fue alterado al que se presenta en la Fig. 14.

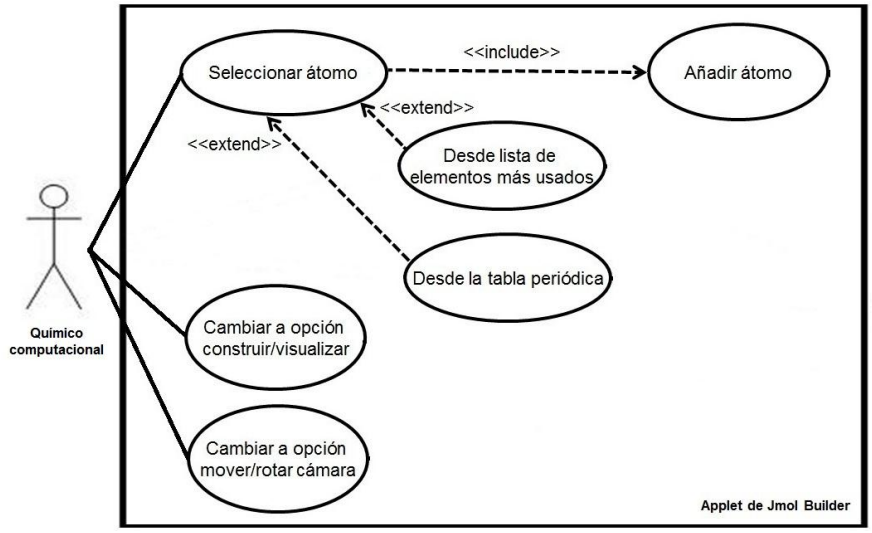


Figura 13: Primer diagrama de casos de uso de *Jmol Builder*

3.3. Diseño de Jmol Builder

Jmol Builder consta de las siguientes clases:

- *JmolBuilderApplet*, dentro del paquete por omisión.
- *JmolBuilderViewer*, dentro del paquete *org.jmol.api*.
- *JmolPeriodicTableInterface*, dentro del paquete *org.jmol.api*.
- *JmolBuilder*, dentro del paquete *org.jmol.applet*.
- *PeriodicTable*, dentro del paquete *org.jmol.applet*.
- *BuilderAppletWrapper*, dentro del paquete *org.jmol.appletwrapper*.
- *BuilderPanelComponent*, dentro del paquete *org.jmol.appletwrapper*.
- *BuilderActionManagerMT*, dentro del paquete *org.jmol.multitouch*.
- *BuilderActionManager*, dentro del paquete *org.jmol.viewer*.
- *BuilderMouseManager*, dentro del paquete *org.jmol.viewer*.
- *BuilderViewer*, dentro del paquete *org.jmol.viewer*.

JmolBuilderApplet es la clase con la que inicia *Jmol Builder*. Las clases *JmolBuilderViewer* y *BuilderViewer* manejan una gran variedad de funcionalidades, entre ellas: la representación, el almacenamiento y el control del modelo o los modelos sobre la pantalla de trabajo.

JmolBuilder es el corazón del sistema de tal forma que es la clase que contiene los datos de la página web a la que el *applet* se haya integrado y genera la instancia principal de la clase *JmolBuilderViewer*.

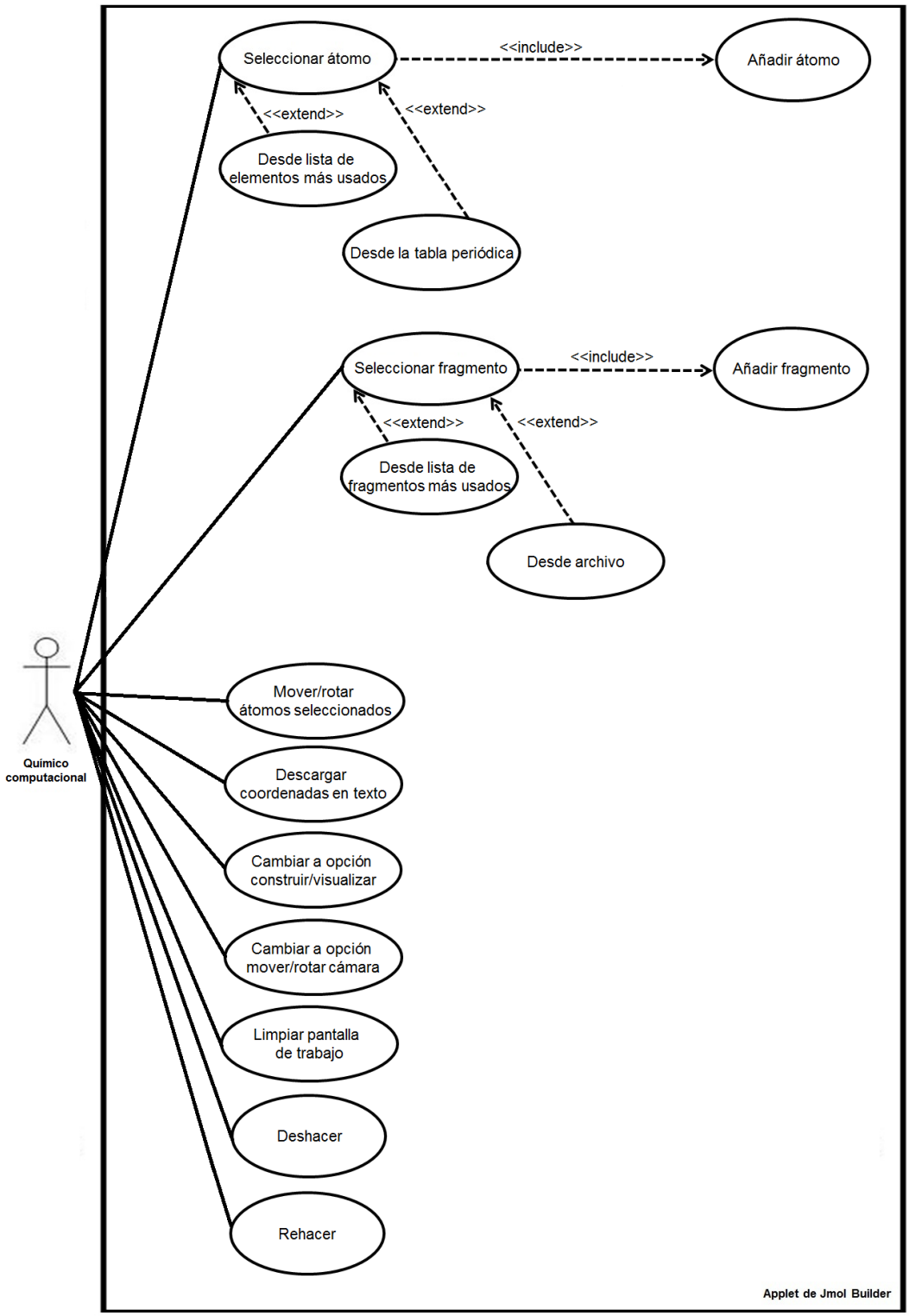


Figura 14: Último diagrama de casos de uso de *Jmol Builder*

Las clases que permitieron desplegar y manipular la barra de herramientas y el control de direcciones fueron: *BuilderAppletWrapper* y *BuilderPanelComponent*. Con las clases *JmolPeriodicTableInterface* y *PeriodicTable* se comunica una tabla periódica interactiva con *Jmol*. Para controlar el funcionamiento del ratón se crearon las clases: *BuilderActionManagerMT*, *BuilderActionManager* y *BuilderMouseManager*.

Se crearon versiones derivadas de las clases de *Jmol* tomando en cuenta cuatro elementos importantes: los constructores, los modificadores de acceso, los métodos y los atributos.

- *Constructores:*

  Como *Jmol* está escrito en *Java*, cuando una clase deriva de otra, su constructor debe ejecutar como primera instrucción el constructor de la clase base.

  La única forma que el constructor de una clase base pueda ser ejecutado por el constructor de una clase derivada es cuando su modificador de acceso no es privado. Sin embargo, todas las clases de *Jmol* de las se crearon clases derivadas, excepto *ActionManager* y *Jmol*, tenían un constructor privado. Para no afectar el funcionamiento del sistema original cambiando su modificador de acceso, se llamó a una nueva versión del constructor que se diferencia de la original por ser protegida y por recibir un parámetro booleano adicional con un valor de verdadero.

- *Modificadores de acceso*:

  Cada vez que se necesitó cambiar el modificador de acceso de un método de una clase base, especialmente si era privado, para no afectar el funcionamiento del sistema original cambiando su modificador de acceso, se creó un nuevo método en la nueva versión de la clase que recibiera los mismos parámetros, devolviera los mismos resultados y que ejecutara las mismas instrucciones que el método que se necesitó modificar, pero con su modificador de acceso público o protegido.

  Luego, se implementaron otros métodos en la clase base y en otras clases e interfaces que permitieran ejecutar el nuevo método de la clase derivada.

  Por ejemplo, el método *setModelKitMode* de la clase *Viewer* es privado pero se necesitó llamarlo desde la clase *BuilderAppletWrapper* con la instancia de la clase *WrappedApplet* inicializada con la clase *JmolBuilder*. Entonces se crearon los siguientes métodos:

  - *setBuilderModelKitMode*, dentro de la interfaz *WrappedApplet*.
  - *setBuilderModelKitMode*, dentro de la clase *Jmol*.

- *setBuilderModelKitMode*, dentro de la clase *JmolBuilder*.
- *setBuilderModelKitMode*, dentro de la clase *JmolViewer*.
- *setBuilderModelKitMode*, dentro de la clase *Viewer*.
- *setBuilderModelKitMode*, dentro de la clase *BuilderViewer*.

La versión del método *setBuilderModelKitMode* dentro de la clase *BuilderViewer* recibe los mismos parámetros, devuelve los mismos resultados y ejecuta las mismas instrucciones que el método *setModelKitMode* de la clase *Viewer*. La diferencia entre ellos es que *setBuilderModelKitMode* es público.

Los demás métodos fueron los que se crearon para ejecutar el método *setBuilderModelKitMode* de la clase *BuilderViewer* desde la clase *BuilderAppletWrapper*.

- *Métodos*:

La mayoría de los métodos de las clases exclusivas de *Jmol Builder* provienen de las clases de *Jmol* de las que derivan. El acceso a esos métodos desde las clases exclusivas de *Jmol Builder* permitió modificarlos sin originar conflictos con el sistema original, ejecutar nuevos métodos dentro de ellos y agregar varias funcionalidades nuevas, entre ellas: la recarga del modelo al agregar un fragmento utilizando la clase *BuilderViewer* y la creación de la barra de herramientas utilizando la clase *BuilderAppletWrapper*.

Algunos métodos de las clases bases eran privados por lo que no podían ser accedidos por clases derivadas. Sin embargo, para separar *Jmol Builder* de *Jmol*, las clases exclusivas de *Jmol Builder* debían tener acceso a todas las funcionalidades de las clases del sistema original. Por esa razón, para incorporar nuevas funcionalidades al sistema, se crearon nuevos métodos en las clases exclusivas de *Jmol Builder* que ejecutaran las mismas operaciones que los métodos privados. En algunos casos, como con los constructores, se les pasó un parámetro adicional y, en otros casos, se les asignó un nombre diferente a cualquier otro método de esa clase.

- *Atributos*:

Al igual que los métodos, la mayoría de los atributos de las clases exclusivas de *Jmol Builder* provienen de las clases de *Jmol* de las que derivan. El acceso a esos atributos desde las clases exclusivas de *Jmol Builder* permitió modificarlos sin originar conflictos con el sistema original y agregar varias funcionalidades nuevas, entre ellas la creación y actualización del control de direcciones, según las dimensiones del *applet,* utilizando la clase *BuilderAppletWrapper*.

Algunos atributos de las clases bases eran privados por lo que no podían ser accedidos por clases derivadas. Sin embargo, para separar *Jmol Builder* de *Jmol*, las clases exclusivas de *Jmol Builder* debían tener acceso a todas las funcionalidades de las clases del sistema original. Por esa razón, para incorporar nuevas funcionalidades al sistema, se crearon nuevos atributos en las clases exclusivas de *Jmol Builder* con el mismo nombre y las mismas características que los atributos privados.

La única interfaz exclusiva de *Jmol Builder* es *JmolPeriodicTableInterface* y es bastante sencilla. No deriva de otra interfaz y no contiene métodos con cuerpo sino sólo su declaración. Además, las declaraciones de los métodos utilizan tipos de datos básicos como booleanos en lugar de instancias de otras clases o interfaces. Por lo tanto la creación de *JmolPeriodicTableInterface* no afecta a *Jmol*. Sin embargo, se tuvieron que hacer varios cambios al sistema original para usarla:

- *Nuevo método en la interfaz JmolAppConsoleInterface*:

  Se creó el método *getPeriodicTable* en la interfaz *JmolAppConsoleInterface* y se le indicó que retornara una instancia de la interfaz *JmolPeriodicTableInterface* para luego implementar su cuerpo en la clase *JmolConsole*.

- *Nueva importación en la clase JmolConsole*:

  Se importó la interfaz *JmolPeriodicTableInterface* dentro de la clase *JmolConsole* para implementar el cuerpo del método *getPeriodicTable* declarado en la interfaz *JmolAppConsoleInterface*.

- *Nuevo método en la clase JmolConsole*:

  Se implementó un nuevo método llamado *getPeriodicTable* según su declaración en la interfaz *JmolAppConsoleInterface*, para crear una instancia de la interfaz *JmolPeriodicTableInterface*, retornarla a la clase *BuilderViewer* y luego mostrar la ventana de la tabla periódica interactiva desde la clase *BuilderAppletWrapper*.

El resto de los cambios en los cuales se usó la interfaz *JmolPeriodicTableInterface* se realizaron a una de las clases exclusivas de *Jmol Builder*, *BuilderViewer*, por lo tanto, no entraron en conflicto con el sistema original.

La Fig. 15 muestra el diagrama de clases *UML* de las nuevas clases e interfaces que se implementaron.

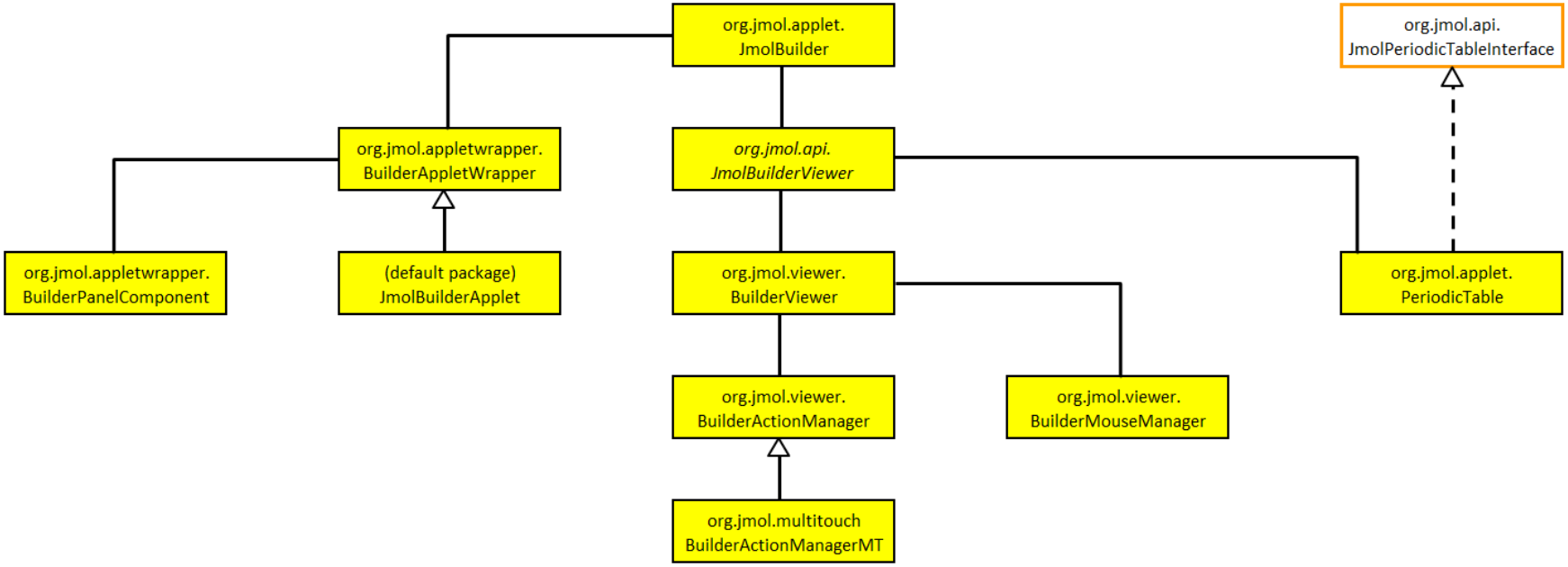


Figura 15: Diagrama de las nuevas clases e interfaces. *JmolPeriodicTableInterface* es una interfaz. Los demás objetos representan las clases.

3.3.1. JmolBuilderApplet

Se creó un nuevo archivo *Java* llamado *JmolBuilderApplet.java* dentro del paquete por omisión. En el archivo se declaró la clase *JmolBuilderApplet* y se utilizó la información de la clase *JmolApplet* para implementarla.

Luego se colocaron instrucciones en la clase *JmolBuilderApplet* para que derivara de la clase *BuilderAppletWrapper* e implementara las mismas interfaces que la clase *JmolApplet*.

El diagrama de clases *UML* de la clase *JmolBuilderApplet* se muestra en la Fig. 16.

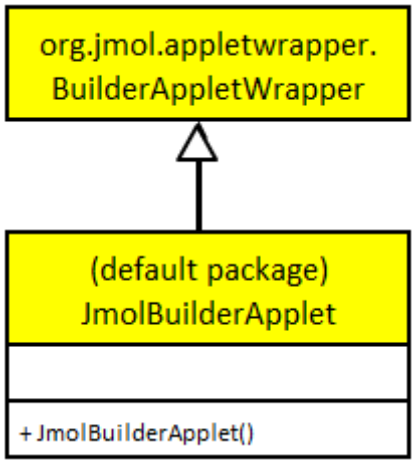


Figura 16: Diagrama de clases *UML* de la clase *JmolBuilderApplet*

JmolBuilderApplet fue creada para ser la primera clase en ejecutarse en lugar de la clase *JmolApplet*. El constructor de *JmolBuilderApplet* manda a ejecutar la clase *JmolBuilder* y no la clase *Jmol*. De esa forma se ejecuta el *applet* de *Jmol Builder* en lugar del de *Jmol*.

3.3.2. BuilderAppletWrapper

Se creó un nuevo archivo *Java* llamado *BuilderAppletWrapper.java* dentro del paquete *org.jmol.appletwrapper*. En el archivo se declaró la clase *BuilderAppletWrapper* y se utilizó la información de la clase *AppletWrapper* para implementarla. Luego se convirtió la clase *BuilderAppletWrapper* en derivada de la clase *AppletWrapper*.

El diagrama de clases *UML* de la clase *BuilderAppletWrapper* se muestra en la Fig. 17.

Desde el constructor de la clase *BuilderAppletWrapper*, se llamó a una nueva versión del constructor de la clase *AppletWrapper*, que recibe un parámetro booleano adicional. Se pasó el nuevo parámetro con un valor de verdad de *true*. Los demás parámetros que recibió el constructor fueron los mismos del constructor de *BuilderAppletWrapper*.

Se importaron las siguientes clases dentro de la clase *BuilderAppletWrapper*:

- *Elements*, dentro del paquete *org.jmol.util*.
- *ActionListener*, dentro del paquete *java.awt.event*.
- *ActionEvent*, dentro del paquete *java.awt.event*.
- *KeyEvent*, dentro del paquete *java.awt.event*.
- *JButton*, dentro del paquete *javax.swing*.
- *JOptionPane*, dentro del paquete *javax.swing*.
- *BorderFactory*, dentro del paquete *javax.swing*.
- *JPanel*, dentro del paquete *javax.swing*.

Se cambió la funcionalidad de los siguientes métodos de la clase *AppletWrapper* a través de la clase *BuilderAppletWrapper*:

- *init*:

  Comienza la ejecución del *applet*. No retorna valor alguno y no recibe parámetros. Fue modificado de forma que generara la barra de herramientas y el control de direcciones, tomando en cuenta las dimensiones del *applet*.

  Se fijó un rango de valores para que los paneles no fueran creados con botones con dimensiones demasiado grandes o demasiado pequeñas. El *applet* no puede tener dimensiones menores que 480 píxeles de ancho y 312 píxeles de alto.

  El tamaño de los botones de la barra de herramientas depende del botón más pequeño, cuyo valor es igual a la quinceava parte de la altura del *applet*, tomando en cuenta que el tamaño mínimo es 21 píxeles. La fuente de los botones de la barra de herramientas es entre 11 y 16. El valor de la fuente de esos botones según las dimensiones del *applet* se obtiene con la fórmula $F_b = (B_m * F_m)/F_M$, donde: B_m es el tamaño del botón más pequeño, F_m es la fuente mínima y F_M es la fuente máxima. El tamaño del botón más grande de la barra de herramientas es igual al doble del más pequeño. El borde entre los botones es igual a la quinta parte del botón más pequeño de la barra de herramientas.

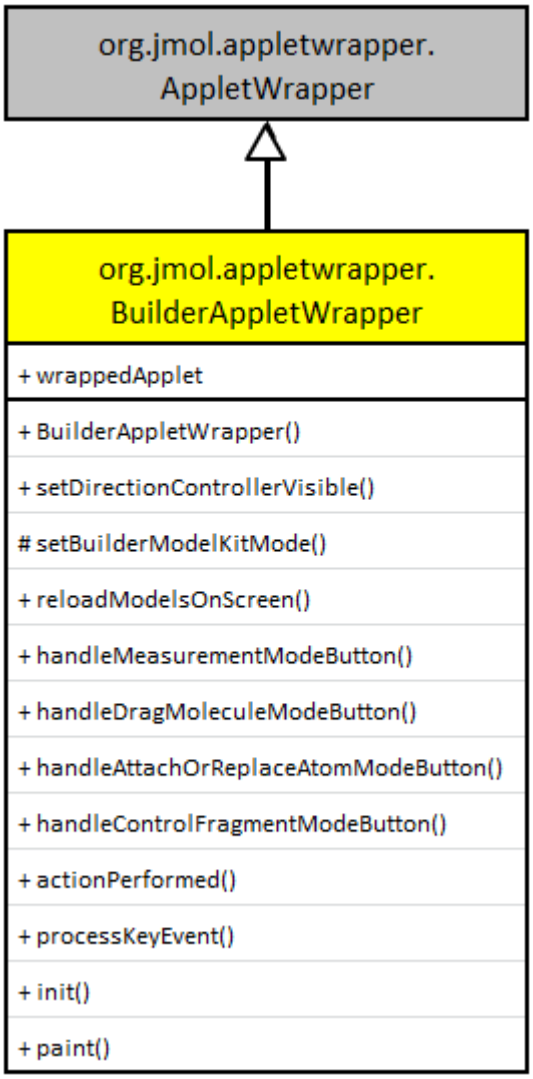


Figura 17: Diagrama de clases *UML* de la clase *BuilderAppletWrapper*

No se definió una estructura específica para el arreglo de los botones. De esa forma se pudieron insertar los paneles y los botones en posiciones arbitrarias de la pantalla de trabajo. La barra de herramientas se insertó en el sector izquierdo del *applet* mientras que el control de direcciones se insertó en el sector inferior derecho del *applet*. Cada botón fue inicializado con el constructor de la clase *BuilderPanelComponent* y fue asociado a un *tooltip*.

El tamaño mínimo de los botones del control de direcciones debe ser 18 píxeles y el máximo debe ser 26 píxeles. Los botones se generan según las dimensiones del *applet* y el tamaño del botón más pequeño, cuyo valor es igual a la décima parte de la altura requerida del panel. Esa altura fue calculada con la fórmula $A_r = A_a/2 - A_a/10$, donde: A_a es la altura del *applet*. La fuente de los botones del control de direcciones es entre 8 y 12. El valor de la fuente de esos

botones según las dimensiones del *applet* se obtiene con la fórmula $F_b = F_m + I_f$, donde: F_m es la fuente mínima e I_f es el incremento de la fuente. El incremento fue calculado con el redondeo por exceso del resultado de la fórmula $I_f = f * (F_M - F_m)$, donde: f es un factor con valores entre 0 y 1, F_M es la fuente máxima y F_m es la fuente mínima. El factor se obtuvo con la fórmula $f = (B_m – C_m)/(C_M - C_m)$, donde: B_m es el tamaño del botón más pequeño, C_m es el tamaño mínimo para un botón del control de direcciones y C_M es el tamaño máximo para un botón del control de direcciones. El tamaño del botón más grande del control de direcciones es igual al doble del más pequeño. El borde entre los botones es igual a la quinta parte del botón más pequeño del control de direcciones.

Se crearon variables para modificar el formato de los botones de la barra de herramientas y el control de direcciones al ser activados o desactivados, según las dimensiones de los botones.

El control de direcciones se oculta y se registran las dimensiones del *applet* antes de ejecutar las últimas instrucciones del método, que comienzan la ejecución del *applet*.

Hay más información sobre la estructura de la barra de herramientas y el control de direcciones en el Capítulo IV de este trabajo, titulado “*Pruebas y Resultados*”.

- *paint*:

  No retorna valor alguno y recibe como parámetro una instancia de la clase *Graphics*.

  El método *paint* de la clase *AppletWrapper* primero verifica si la referencia de la clase *WrappedApplet* fue inicializada. Si fue así, dibuja el *applet* con el método *paint* de la clase *Jmol*. Luego ejecuta el método *update* de la clase *AppletWrapper* cada cierto tiempo como última instrucción del método.

  Fue modificado de forma que primero verificara si la referencia de la clase *WrappedApplet* fue inicializada. Si fuera así, se le indicó que, sólo si las dimensiones del *applet* fueran cambiadas en tiempo de ejecución, registrara las dimensiones del *applet*, generara la barra de herramientas y el control de direcciones utilizando las mismas fórmulas que el método *init* de esta clase, y dibujara el *applet* con el método *paint* de la clase *JmolBuilder*. Luego se le indicó que ejecutara el método *update* de la clase *BuilderAppletWrapper* como última instrucción del método.

También se agregaron nuevas funcionalidades a la clase *AppletWrapper* a través de los siguientes métodos que se implementaron en la clase *BuilderAppletWrapper*:

- *setDirectionControllerVisible:*

  No retorna valor alguno y recibe un parámetro booleano que determina si el control de direcciones se mostrará o no la próxima vez que se refresque la pantalla. Sólo si el valor del parámetro es falso, se posiciona la vista de frente al modelo y se reinician los botones del control de direcciones. Luego, si el control de direcciones se encuentra sobre la pantalla, lo muestra o lo oculta según el valor de verdad del parámetro que reciba el método.

- *setBuilderModelKitMode*:

  No retorna valor alguno y recibe un parámetro booleano que determina si se activa o no el *Model Kit*. Primero desactiva el modo de control de átomos seleccionados, el botón "*Drag molecule mode*" y el botón "*Delete atom mode*" si alguno fue activado. Luego, sólo si el valor del parámetro es cierto, se desactiva el modo de centrado de cámara sobre un átomo y el modo de medición de distancias y ángulos entre los átomos si alguno fue activado, se activa el modo de inserción o reemplazo de átomos; y se muestran u ocultan etiquetas que indican el nombre de los átomos seleccionados del modelo, dependiendo si el botón "*Show/hide labels over selected atoms*" está activado o no.

  Si el valor del parámetro es falso, se activa el modo de visualización, desactivando el resto de los botones de la barra de herramientas, ejecutando *scripts* que desactivan el *Model Kit* y regresan al sistema a su estado original; e indicando con el botón "*Currently selected atom to add*" que no hay átomos para unir al modelo.

  Hay más información sobre los botones de la barra de herramientas en el Capítulo IV de este trabajo, titulado "*Pruebas y Resultados*".

- *reloadModelsOnScreen*:

  Recarga los modelos sobre la pantalla de trabajo. No retorna valor alguno y recibe como parámetros: la ubicación del modelo a cargar o unir al modelo actual, un booleano que determina si se une o no el modelo como fragmento al modelo que se esté mostrando sobre la pantalla de trabajo, y un booleano que determina si se muestra o no una ventana de diálogo para recibir la ubicación del modelo a cargar o unir al modelo actual.

  Primero se ejecuta uno de 3 *scripts* donde un *script* agrega el modelo pasado por parámetro al modelo actual, otro une el modelo pasado a través de una ventana de diálogo y el tercero vuelve a cargar el modelo actual. Luego se activa el *Model Kit*, se indica con el botón "*Currently selected atom to add*" que el átomo para unir al modelo es "C", y se posiciona el modelo actual en el centro de la pantalla de trabajo.

- *handleMeasurementModeButton*:

  Ejecuta las operaciones relacionadas con el botón "*Place/hide measurements over selected atoms*". No retorna valor alguno y no recibe parámetros.

  Primero se desactiva el modo de centrado de cámara sobre un átomo si fue activado. Luego se verifica si la referencia de la clase *WrappedApplet* fue inicializada. Si fue así, se desactiva el *Model Kit* con el método *setBuilderModelKitMode* y, sólo si hay un modelo sobre la pantalla, se procede a verificar el estado del botón "*Place/hide measurements over selected atoms*". Si el botón no está activo, se activa el botón y el modo de medición de distancias y ángulos entre los átomos. En caso contrario, se desactiva el botón y el modo antes mencionado.

- *handleDragMoleculeModeButton*:

  Ejecuta las operaciones relacionadas con el botón "*Drag molecule mode*". No retorna valor alguno y no recibe parámetros.

  Primero se desactiva el modo de centrado de cámara sobre un átomo si fue activado. Luego se verifica si la referencia de la clase *WrappedApplet* fue inicializada. Si fue así, se desactiva el modo de control de átomos seleccionados y el botón "*Delete atom mode*", si alguno fue activado. Luego se procede a verificar el estado del botón "*Drag molecule mode*". Si el botón no está activo, se activa el botón y el modo de arrastre de moléculas. En caso contrario, se desactiva el botón y el modo antes mencionado.

  Cuando el modo de arrastre de moléculas se activa o desactiva, sucede lo mismo con el modo de medición de distancias y ángulos entre los átomos.

- *handleAttachOrReplaceAtomModeButton*:

  Ejecuta las operaciones relacionadas con el botón "*Attach/replace atom mode*". No retorna valor alguno y no recibe parámetros.

  Primero se verifica si la referencia de la clase *WrappedApplet* fue inicializada. Si fue así, se procede a verificar el estado del botón "*Attach/replace atom mode*". Si el botón no está activo, se activa el botón y el modo de inserción o reemplazo de átomos. En caso contrario, se desactiva el botón y el modo antes mencionado.

- *handleControlFragmentModeButton*:

  Ejecuta las operaciones relacionadas con el botón "*Control set of atoms mode*". No retorna valor alguno y no recibe parámetros.

Primero se desactiva el modo de centrado de cámara sobre un átomo si fue activado. Luego se verifica si la referencia de la clase *WrappedApplet* fue inicializada. Si fue así, se desactiva el modo de inserción o reemplazo de átomos, el modo de arrastre de moléculas, el botón "*Place/hide measurements over selected atoms*" y el botón "*Delete atom mode*", si alguno fue activado. Luego se procede a verificar el estado del botón "*Control set of atoms mode*". Si el botón no está activo, se activa el botón y el modo de control de átomos seleccionados, y se muestra el control de direcciones. En caso contrario, se desactiva el botón y el modo antes mencionado, y se oculta el control de direcciones.

- *actionPerformed*:

Le da funcionalidad a todos los botones de la barra de herramientas y los del control de direcciones, ejecutándose cada vez que uno de ellos es presionado. No retorna valor alguno y recibe como parámetro una instancia de una clase derivada de *Java* llamada *ActionEvent*, que contiene una referencia al botón que fue presionado.

Los botones de la barra de herramientas son:

- *Clear*.
- *Load*.
- *Save as MDL Molfile data*.
- *Save current view as image*.
- *Place/hide measurements over selected atoms*.
- *Show/hide labels over selected atoms*.
- *Attach/replace atom mode*.
- *Delete atom mode*.
- *Drag molecule mode*.
- *Control set of atoms mode*.
- *Add CH_4 fragment*.
- *Add triangle fragment*.
- *Add benzene fragment*.
- *Add fragment from file*.
- *Most used atom C*.
- *Most used atom H*.
- *Most used atom O*.

- *Show periodic table of elements*.
- *Currently selected atom to add*.

El control de direcciones tiene los siguientes botones:

- *Translation/rotation north-wise*.
- *Translation/rotation west-wise*.
- *Translation/rotation east-wise*.
- *Translation/rotation south-wise*.
- *Rotation clock-wise*.
- *Rotation counterclock-wise*.
- *Show/hide cartesian coordinate system*.
- *Increase precision*.
- *Reset the current view*.
- *Translate selected atoms*.
- *Rotate selected atoms*.
- *Set front view*.
- *Set back view*.
- *Set left view*.
- *Set right view*.
- *Set top view*.
- *Set bottom view*.

- *processKeyEvent*:

Este método accede a funcionalidades de *Jmol* y *Jmol Builder* a través del teclado, ejecutándose cada vez que alguna tecla es utilizada. No retorna valor alguno y recibe como parámetro una instancia de una clase derivada de *Java* llamada *KeyEvent*, que contiene información sobre la tecla que fue utilizada y la forma en que fue utilizada. En el caso de *Jmol Builder*, las funcionalidades sólo son accedidas cuando una de un grupo selecto de teclas deja de ser presionada.

Las teclas que pueden acceder a funcionalidades de *Jmol* y *Jmol Builder* son:

- F3.
- F4.

- F5.
- F6.
- F7.
- F8.
- F9.
- F10.
- F11.
- F12.
- U.
- R.

Como la clase *BuilderAppletWrapper* es derivada de la clase *AppletWrapper*, se eliminó la declaración de sus atributos públicos y protegidos. Después se agregaron nuevos atributos para la construcción, actualización y manejo de la barra de herramientas y el control de direcciones. La mayoría de ellos se inicializaron en el constructor de *BuilderAppletWrapper*.

La creación de la clase *BuilderAppletWrapper* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo del control de direcciones.
- Creación y manejo de la tabla periódica interactiva.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del teclado.

3.3.3. BuilderPanelComponent

Se creó un nuevo archivo *Java* llamado *BuilderPanelComponent.java* dentro del paquete *org.jmol.appletwrapper*. En el archivo se declaró una clase derivada de la clase *JButton*, dentro del paquete *javax.swing*, que se llamó *BuilderPanelComponent*.

Se agregaron nuevos atributos para instanciar un botón con una imagen dada y para modificar su formato. La mayoría de los atributos se inicializaron en el constructor de *BuilderPanelComponent*.

El diagrama de clases *UML* de la clase *BuilderPanelComponent* se muestra en la Fig. 18.

Se implementaron los siguientes métodos:

- *setEnabled*:

  Este método activa o desactiva el botón que esté representando una instancia de esta clase. No retorna valor alguno y recibe como parámetro un valor booleano que determina si se activa o desactiva el botón. Primero se ejecuta el método *setEnabled* de la clase base *JButton* pasándole por parámetro el valor booleano dado y se procede a verificar si el botón fue inicializado con una imagen. Si fue así, se modifica el color de fondo del botón dependiendo del valor booleano dado. En caso contrario, se modifica el color de fondo, la fuente y el color de las letras del botón dependiendo del valor booleano dado.

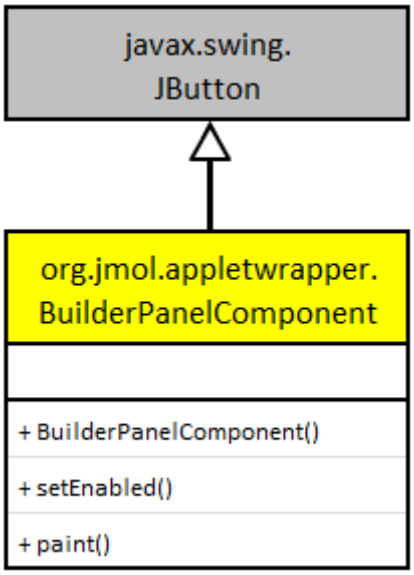


Figura 18: Diagrama de clases *UML* de la clase *BuilderPanelComponent*

- *BuilderPanelComponent*:

  Es un constructor de esta clase. Como todo constructor, retorna una nueva instancia de la clase a la que pertenece. Recibe como parámetro un texto que puede indicar el texto mostrado por el botón o puede indicar el nombre del archivo que contiene la imagen que será mostrada por el botón. Primero se verifica si el texto dado es un nombre de archivo. Si lo es, se indica con el botón un texto vacío, se busca el archivo en el sistema, se obtiene la imagen asociada al archivo y se registra la imagen en un atributo de esta clase.

- *paint*:

  Este método dibuja el botón sobre la pantalla de trabajo. No retorna valor alguno y recibe como parámetro una instancia de una clase derivada de *Java* llamada *Graphics*. Primero se ejecuta el método *paint* de la clase base *JButton* pasándole por parámetro el valor dado y se procede a verificar si el botón fue inicializado con una imagen. Sólo si fue así, se intenta modificar la nitidez de la imagen que el botón deba mostrar. Si es posible modificar la nitidez, se dibuja el botón con la imagen modificada pero, si no es posible, se dibuja el botón con la imagen original.

También se crearon métodos para modificar y obtener los valores de los atributos de esta clase, para modificar la nitidez de las imágenes de los botones que las mostraran (*hasAlpha*, *toBufferedImage*, *createCompatibleImage*, *resize*, *blurImage* y *resizeTrick*), para establecer las dimensiones del botón (*getPreferredSize*, *getMinimumSize* y *getMaximumSize*) y para depurar el código de la clase (*isDebugging* y *setIsDebugging*).

La creación de la clase *BuilderPanelComponent* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo del control de direcciones.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del teclado.

3.3.4. JmolBuilder

Se creó un nuevo archivo *Java* llamado *JmolBuilder.java* dentro del paquete *org.jmol.applet*. En el archivo se declaró la clase *JmolBuilder* y se utilizó la información de la clase *Jmol* para implementarla. Luego se convirtió la clase *JmolBuilder* en derivada de la clase *Jmol*.

El diagrama de clases *UML* de la clase *JmolBuilder* se muestra en la Fig. 19. Para esta clase se importó la clase JButton, que se encuentra dentro del paquete *javax.swing*, y se crearon los siguientes métodos dentro de la clase:

- *setBuilderModelKitMode:*

  No retorna valor alguno y recibe un parámetro booleano que determina si se activa o no el *Model Kit*. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se activa o no el *Model Kit*, con el método *setBuilderModelKitMode* de la clase *BuilderViewer*, según el valor de verdad del parámetro que reciba el método.

- *isBuilderModelKitMode:*

  Retorna un valor booleano que determina si el *Model Kit* fue activado o no y no recibe ningún parámetro. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se retorna la respuesta con el método *isBuilderModelKitMode* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método retorna como respuesta que el *Model Kit* no fue activado.

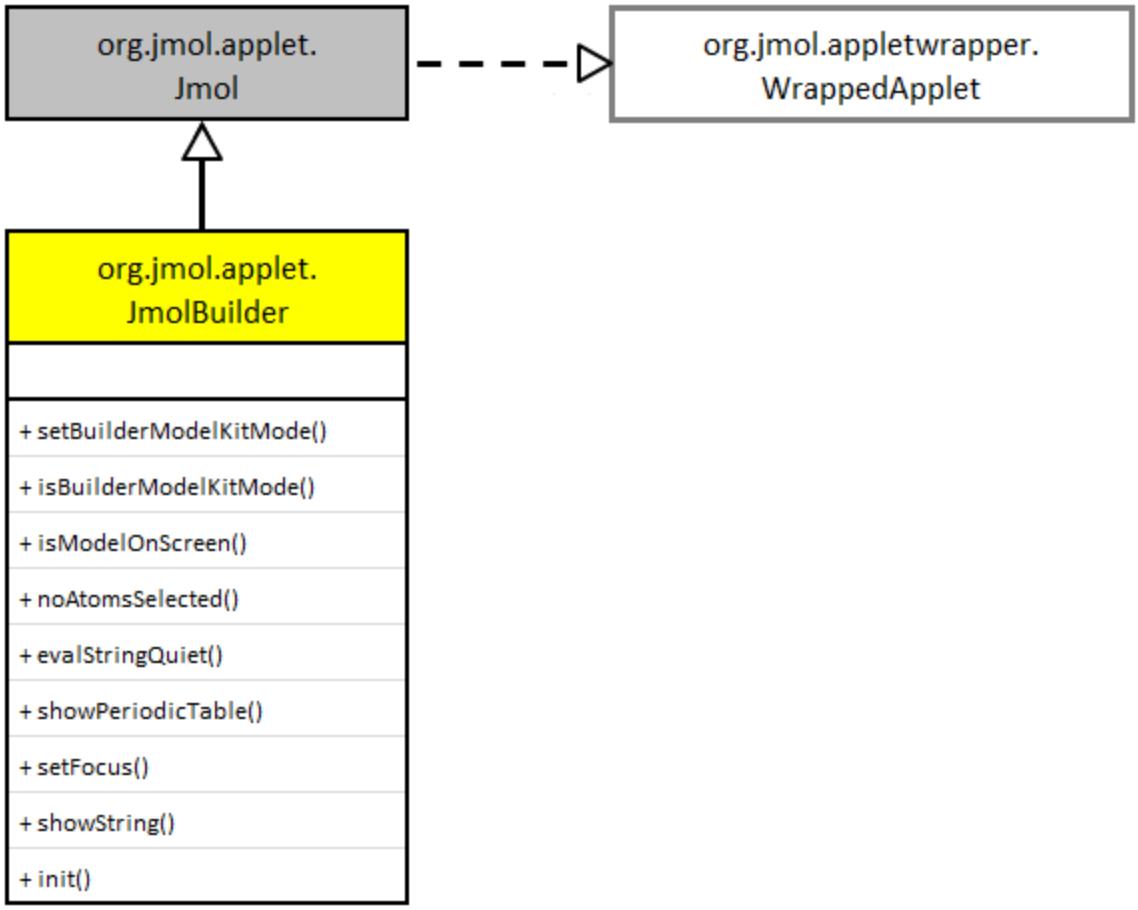


Figura 19: Diagrama de clases *UML* de la clase *JmolBuilder*

- *isModelOnScreen*:

  Retorna un valor booleano que determina si se está mostrando un modelo sobre la pantalla de trabajo o no y no recibe ningún parámetro. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se retorna la respuesta con el método *isModelOnScreen* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método retorna como respuesta que no se está mostrando un modelo sobre la pantalla de trabajo.

- *noAtomsSelected:*

  Retorna un valor booleano que determina si se han seleccionado átomos del modelo sobre la pantalla de trabajo o no y no recibe ningún parámetro. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se retorna la respuesta con el método *getNoneSelected* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método retorna como respuesta que se han seleccionado átomos del modelo sobre la pantalla de trabajo.

- *evalStringQuiet:*

  Este método manda a ejecutar un *script*, retorna un valor de texto que determina si la ejecución del *script* fue exitosa o no y recibe un parámetro de texto que indica el *script* a ejecutar. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se retorna la respuesta con el método *evalStringQuiet* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método retorna como respuesta que la ejecución del *script* no fue exitosa.

- *showPeriodicTable:*

  Este método manda a mostrar la ventana de la tabla periódica, no retorna valor alguno y recibe como parámetros: referencias a algunos botones de la barra de herramientas y un parámetro booleano que determina si se muestra o no. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se muestra o no la ventana con el método *showPeriodicTable* de la clase *BuilderViewer,* según el valor de verdad del parámetro booleano que reciba el método.

- *setFocus*:

  Este método manda a posicionar el foco de atención sobre la pantalla de trabajo, no retorna valor alguno y no recibe parámetros. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se posiciona el foco o no con el método *setFocus* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método no ejecuta ninguna instrucción.

- *showString:*

  Este método fue utilizado para depurar el código fuente. Manda a mostrar un texto sobre la consola o la salida estándar de datos, no retorna valor alguno y recibe como parámetros el texto a mostrar y un parámetro booleano cuyo valor no es relevante para este trabajo de investigación, aunque se notó que el valor pasado siempre es falso. Sólo si la referencia de la clase *JmolViewer* fue inicializada, se muestra el texto dado sobre la consola o la salida estándar de datos con el método *showString* de la clase *BuilderViewer*. Si la referencia no ha sido inicializada, el método no ejecuta ninguna instrucción.

También se cambió la funcionalidad del método *init* de la clase *Jmol* a través de la clase *JmolBuilder*, indicándole que iniciara el *applet* con el foco de atención sobre la pantalla de trabajo para la captura de eventos del teclado después de generar la barra de herramientas y el control de direcciones.

Ya que derivaba de la clase *Jmol*, se eliminaron de *JmolBuilder* los atributos públicos, los atributos protegidos y la clase interna *MyStatusListener*. Después se instanció la referencia a la clase *JmolViewer* con el método estático *allocateViewer* de la clase *JmolBuilderViewer* en vez del de la clase *JmolViewer*.

La creación de la clase *JmolBuilder* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo del control de direcciones.
- Creación y manejo de la tabla periódica interactiva.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del teclado.
- Interacción entre *JME* y *Jmol Builder*.

3.3.5. JmolBuilderViewer

Se creó un nuevo archivo *Java* llamado *JmolBuilderViewer.java* dentro del paquete *org.jmol.api*. En el archivo se declaró la clase *JmolBuilderViewer* y se utilizó la información de la clase *JmolViewer* para implementarla. Luego se convirtió la clase *JmolBuilderViewer* en derivada de la clase *JmolViewer*. Se le indicó a la clase que importara la clase *BuilderViewer* en vez de la clase *Viewer* y que importara la clase *JButton*, dentro del paquete *javax.swing*.

El diagrama de clases *UML* de la clase *JmolBuilderViewer* se muestra en la Fig. 20.

Se eliminaron todos los métodos que no fueron declarados con el nombre *allocateViewer* y se modificaron los métodos restantes indicándoles que retornaran una referencia de la clase *JmolViewer* con el método estático *allocateViewer* de la clase *BuilderViewer* en vez de la clase *Viewer*.

La creación de la clase *JmolBuilderViewer* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo del control de direcciones.

- Creación y manejo de la tabla periódica interactiva.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del ratón.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del teclado.
- Interacción entre *JME* y *Jmol Builder*.

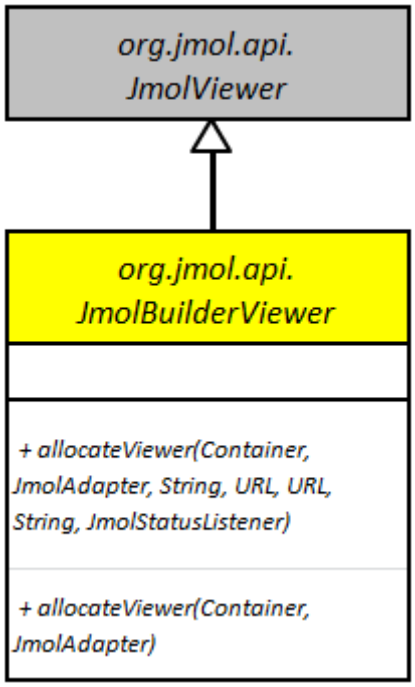


Figura 20: Diagrama de clases *UML* de la clase *JmolBuilderViewer*

3.3.6. BuilderViewer

Se creó un nuevo archivo *Java* llamado *BuilderViewer.java* dentro del paquete *org.jmol.viewer*. En el archivo se declaró la clase *BuilderViewer* y se utilizó la información de la clase *Viewer* para implementarla. Luego se convirtió la clase *BuilderViewer* en derivada de la clase *Viewer* y se declaró su constructor como protegido.

El diagrama de clases *UML* de la clase *BuilderViewer* se muestra en la Fig. 21.

Se le indicó a *BuilderViewer* que importara las siguientes clases:

- *BuilderPanelComponent*, dentro del paquete *org.jmol.appletwrapper*.
- *Panel*, dentro del paquete *java.awt*.

- *Component*, dentro del paquete *java.awt*.
- *JButton*, dentro del paquete *javax.swing*.
- *JOptionPane*, dentro del paquete *javax.swing*.

Se cambió la funcionalidad del método *allocateViewer* de la clase *Viewer* a través de la clase *BuilderViewer*, indicándole que retornara una referencia de la clase *Viewer* instanciada con el constructor de la clase *BuilderViewer* en vez del de la clase *Viewer*.

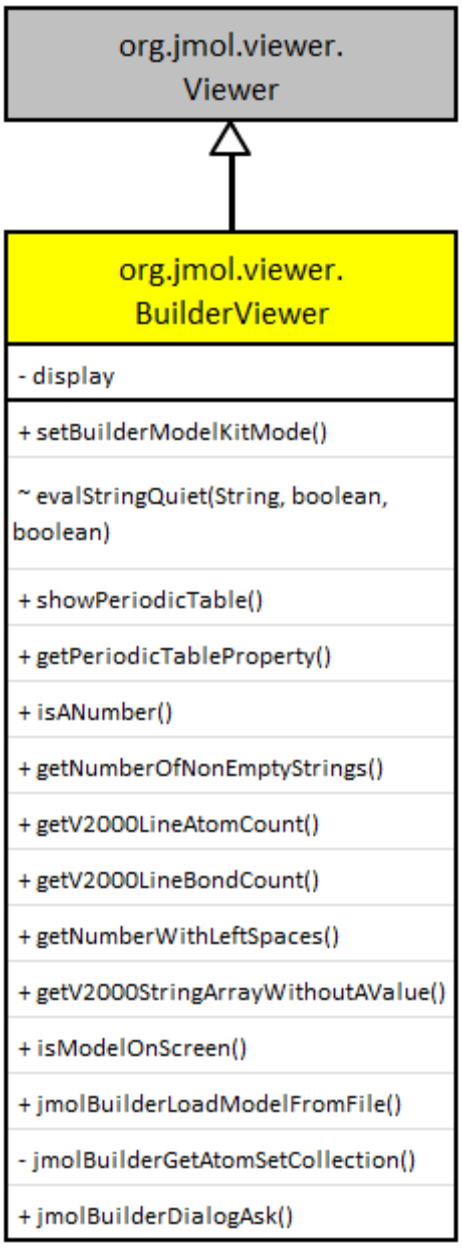


Figura 21: Diagrama de clases *UML* de la clase *BuilderViewer*

Se realizaron sobre el constructor de la clase *BuilderViewer* los cambios que se describen a continuación:

- La instancia de la clase *ActionManager* era inicializada con el constructor de la clase *ActionManager* o con la interfaz *ActionManagerMT*. Se reemplazó el constructor por el de la clase *BuilderActionManager* y en el otro caso se utilizó la interfaz *BuilderActionManagerMT*.
- El constructor de la clase *BuilderMouseManager* inicializa la referencia de la clase *MouseManager14* en lugar del constructor de la clase base.
- Como primera instrucción se llamó a una nueva versión del constructor de la clase *Viewer*, que recibe un parámetro booleano adicional. Se pasó el nuevo parámetro con un valor de verdad de *true*. Los demás parámetros que recibió el constructor fueron los mismos del constructor de *BuilderViewer*.

Como la clase *BuilderViewer* es derivada de la clase *Viewer*, se eliminó la declaración de sus atributos públicos y protegidos. Luego, se creó una instancia de la interfaz *JmolPeriodicTableInterface*, inicializada con el constructor de la clase *PeriodicTable*, que permitió mostrar la tabla periódica interactiva con un botón de la barra de herramientas. También se crearon atributos que nos permitieron cargar un modelo contenido en un archivo, reemplazando el modelo que se esté mostrando sobre la pantalla de trabajo; recargar el modelo actual y, si el modelo está compuesto por distintos modelos secundarios, agruparlos en uno solo antes de recargarlo.

Se cambió la funcionalidad del método *evalStringQuiet* de la clase *Viewer* a través de la clase *BuilderViewer*. Ese método manda a ejecutar un *script*, retorna un valor de texto que determina si la ejecución del *script* fue exitosa o no y recibe como parámetros un *script* y dos booleanos cuyos valores no son relevantes para este trabajo de investigación, aunque se notó que ambos valores siempre son pasados con el valor de verdadero, porque *evalStringQuiet* es llamado de esa forma como única instrucción de otro método llamado *evalStringQuiet* de esta clase, que sólo recibe como parámetro un *script*. Ese método es el que llama *Jmol Builder* desde la clase *BuilderAppletWrapper* y la clase *PeriodicTable*.

Se agregaron nuevas funcionalidades al principio del cuerpo del método que permitieron trabajar con cuatro *scripts* nuevos. El cambio consiste en la verificación de cada *script*, que le índica a *Jmol Builder* que ejecute operaciones distintas. Las cuatro operaciones son las siguientes:

- *Reinicio de Jmol Builder*:

  No recibe argumentos. Primero se verifica si la referencia de la clase *Container* llamada *display* fue inicializada, porque ella contiene información sobre los componentes creados desde la clase *BuilderAppletWrapper*. Si fue así, se utiliza la referencia para recorrer cada elemento de la barra de

herramientas y el control de direcciones. Se activan los componentes con los nombres "*Controller rotation clock-wise*", "*Controller rotation counterclock-wise*" y "*Controller translate selected atoms*", se indica con el componente con el nombre "*Toolbar currently selected atom to add*" que no hay átomos para unir al modelo, el resto de los componentes se desactivan y se genera un *script* que regresa *Jmol Builder* a su estado original, que es interpretado y ejecutado por las instrucciones de la versión original del método *evalStringQuiet*.

- *Carga de un modelo desde un archivo de manera secuencial*:

  Recibe un argumento que indica si se desea cargar un modelo desde un archivo ubicado en la computadora del usuario, a través de una ventana de diálogo, o no. Primero se captura el argumento y, si indica que se cargue un modelo desde un archivo, con el método *jmolBuilderLoadModelFromFile* de esta clase se muestra una ventana de diálogo que captura la ubicación del modelo, dada por el usuario. El sistema no continúa hasta que se obtenga una respuesta del usuario. Una vez obtenida la respuesta, se procede a verificar si es una ubicación correcta. Si es así, se genera un *script* que carga el modelo dado como el único modelo sobre la pantalla de trabajo. El *script* es interpretado y ejecutado por las instrucciones de la versión original del método *evalStringQuiet*.

- *Recarga del modelo o los modelos actuales en uno solo*:

  Recibe un argumento que indica si se desea: recargar el modelo actual o los modelos actuales en uno solo, o recargar el modelo actual junto con un fragmento recién agregado, ya sea de la lista predeterminada de *Jmol Builder* o desde un archivo ubicado en la computadora del usuario, a través de una ventana de diálogo. Primero se verifica si el modelo actual tiene entre 1 y 999 átomos y enlaces, que es un requisito para generar un archivo con el formato *MDL Molfile V2000*. Si es así, se extrae la información del modelo, de ella se registra el número de átomos y enlaces con los métodos *getV2000LineAtomCount* y *getV2000LineBondCount* de esta clase y se procede a verificar si se desea recargar el modelo actual junto con un fragmento recién agregado o no. Si es así, se obtiene la ubicación del fragmento, ya sea de la lista predeterminada de *Jmol Builder* o desde un archivo ubicado en la computadora del usuario, a través de una ventana de diálogo. Si se muestra una ventana de diálogo para obtener la ubicación, el sistema no continúa hasta que se obtenga una respuesta del usuario con el método *jmolBuilderLoadModelFromFile* de esta clase y, una vez obtenida la respuesta, si es una ubicación correcta, se continúa con el método. Luego se extrae la información del fragmento, de ella se registra el número de átomos y enlaces con los métodos *getV2000LineAtomCount* y *getV2000LineBondCount* de esta clase y se procede a unir los modelos. La unión de los modelos consiste en crear un archivo *MDL Molfile V2000* a partir de la información del modelo actual y la del fragmento. *Jmol Builder* genera el encabezado del archivo indicando que el número de átomos es igual a la suma de los átomos

de ambos modelos y que el número de enlaces es igual a la suma de los átomos de ambos modelos, pospone la lista de átomos del fragmento a la lista de átomos del modelo actual, modifica la lista de enlaces del fragmento de acuerdo al número de átomos del modelo actual, pospone la lista modificada de enlaces del fragmento a la lista de enlaces del modelo actual, genera el fin de archivo "*M END*", y se genera un *script* que carga el archivo creado como el único modelo sobre la pantalla de trabajo.

Si no se desea recargar el modelo actual junto con un fragmento recién agregado, sólo se utiliza la información extraída del modelo o los modelos actuales para generar un *script* que los recarga como el único modelo sobre la pantalla de trabajo.

Luego, el *script* generado es interpretado y ejecutado por las instrucciones de la versión original del método *evalStringQuiet*.

- *Nueva eliminación del modelo actual*:

  No recibe argumentos. Primero se vacía el contenido del registro que almacena la información del último modelo que fue cargado por la operación anterior y genera un *script* que elimina el modelo sobre la pantalla de trabajo. El *script* es interpretado y ejecutado por las instrucciones de la versión original del método *evalStringQuiet*.

También se agregaron nuevas funcionalidades a la clase *Viewer* a través de los siguientes métodos que se implementaron en la clase *BuilderViewer*:

- *setBuilderModelKitMode*:

  Activa o desactiva el *Model Kit*. No retorna valor alguno y recibe un parámetro booleano que determina si se activa o desactiva el *Model Kit*.

- *showPeriodicTable*:

  Manda a mostrar la ventana de la tabla periódica, no retorna valor alguno y recibe como parámetros: referencias a algunos botones de la barra de herramientas y un parámetro booleano que determina si se muestra o no la ventana. Primero se verifica si la referencia de la clase *Container* llamada *display* fue inicializada. Sólo si fue así, se procede a verificar si la instancia de la interfaz *JmolPeriodicTableInterface* fue inicializada. Si no fue inicializada, se inicializa con el método *getPeriodicTableProperty* de esta clase. Luego, se ejecuta el método *setVisible* de la clase *PeriodicTable*, pasándole por parámetro el parámetro booleano de este método.

- *getPeriodicTableProperty*:

  Inicializa la instancia de la interfaz *JmolPeriodicTableInterface* de esta clase. Retorna una instancia de la interfaz *JmolPeriodicTableInterface* y recibe como parámetros: referencias a algunos botones de la barra de herramientas y un parámetro booleano que determina si se muestra o no la ventana de la tabla periódica. Primero se inicializa la instancia de la interfaz *JmolAppConsoleInterface* de esta clase. Luego se ejecuta el método *getPeriodicTable* de la clase *JmolConsole*, y se retorna la instancia inicializada de la interfaz *JmolPeriodicTableInterface*.

- *isANumber*:

  Recibe como parámetro un texto y retorna un valor booleano que indica si el texto dado es un número o no.

- *getNumberOfNonEmptyStrings*:

  Recibe como parámetro una lista de líneas de texto y retorna un número que indica la cantidad de líneas de texto vacías en la lista.

- *getV2000LineAtomCount*:

  Recibe la línea de texto de un archivo *MDL Molfile V2000* que indica la cantidad de átomos y enlaces del modelo, y retorna la cantidad de átomos. Primero se toman los primeros 3 caracteres de la línea, se eliminan los espacios en blanco y se verifica si el texto extraído es un número con el método *isANumber* de esta clase. Si no lo es, se retorna un error. En caso contrario, se retorna el texto convertido en número como la cantidad de átomos.

- *getV2000LineBondCount*:

  Recibe la línea de texto de un archivo *MDL Molfile V2000* que indica la cantidad de átomos y enlaces del modelo, y retorna la cantidad de enlaces. Primero se toman los 3 caracteres después de los tres primeros caracteres de la línea, se eliminan los espacios en blanco y se verifica si el texto extraído es un número con el método *isANumber* de esta clase. Si no lo es, se retorna un error. En caso contrario, se retorna el texto convertido en número como la cantidad de enlaces.

- *getNumberWithLeftSpaces*:

  Retorna un texto y recibe como parámetros un texto que representa un número y la longitud del texto que el método retorna. Primero se verifica si el texto dado es un número con el método de esta clase. Sólo si lo es, se calcula el número de espacios en blanco a insertar a la izquierda del texto dado para

que el texto resultante tenga la longitud dada. Luego, se retorna el texto modificado.

- *getV2000StringArrayWithoutAValue*:

  Recibe como parámetros una lista de textos y un texto que representa el elemento a eliminar de la lista dada. Retorna la lista sin el elemento especificado.

- *isModelOnScreen*:

  Retorna un valor booleano que determina si se está mostrando un modelo sobre la pantalla de trabajo o no y no recibe ningún parámetro. Primero se extrae la información del modelo actual con el método *getModelExtract* de la versión original de esta clase y se verifica si la información extraída es vacía. Si lo es, se retorna como respuesta que no se está mostrando un modelo sobre la pantalla de trabajo. En caso contrario, se retorna el valor de verdad opuesto.

- *jmolBuilderLoadModelFromFile*:

  Retorna un texto y recibe como parámetros: una ubicación absoluta, un nombre de archivo, una lista de nombres de archivos, una instancia de la clase llamada *reader* que leerá el archivo o los archivos dados, un valor booleano que indica si se unirá el archivo o los archivos al modelo, una instancia llamada *htParams* que almacena una lista de parámetros, un *script* y un número que indica si sólo se debe leer el archivo dado. *Jmol Builder* ejecuta las mismas instrucciones que el método *loadModelFromFile* de la versión original de esta clase, excepto en los siguientes casos:

  - *Parámetro reader vacío*:

  Durante la ejecución del método, se verifica si el parámetro *reader* no fue instanciado. En ese caso se inicializa una referencia de la clase *Object*. La versión original de este método la inicializa con el método *getAtomSetCollection* de la versión original de esta clase. *Jmol Builder* la inicializa con el método *jmolBuilderGetAtomSetCollection* de esta clase. La diferencia entre ambas versiones es que la de *Jmol Builder* no continúa hasta que el método llamado termine de ejecutarse. Luego la ejecución del método continúa como la versión original hasta que se cumple el caso que se describirá a continuación.

  - *Nuevo valor de retorno*:

  Cerca del final del método se verifican los parámetros de la variable *htParams*. A diferencia de la versión original de este método, se verifica la existencia de un parámetro de *Jmol Builder* que, si existe, le indica al método que retorne la ubicación absoluta del archivo, capturada durante la ejecución

del método *jmolBuilderGetAtomSetCollection* de esta clase. La ubicación pertenece a un archivo seleccionado por el usuario a través de una ventana de diálogo.

- *jmolBuilderGetAtomSetCollection*:

  Retorna una instancia de la clase *Object* y recibe como parámetros: un nombre de archivo, un valor booleano que indica si se unirá el archivo al modelo, una instancia llamada *htParams* que almacena una lista de parámetros y un *script*. *Jmol Builder* ejecuta las mismas instrucciones que el método *getAtomSetCollection* de la versión original de esta clase, excepto en el caso en que el nombre de archivo dado es "*?*". Cuando eso sucede, se inicializa una instancia de la clase *Object* con el método *jmolBuilderCreateAtomSetCollectionFromFile* de la clase *FileManager* en vez de la versión original de ese método, llamada *createAtomSetCollectionFromFile*, también de la clase *FileManager*. El método *jmolBuilderCreateAtomSetCollectionFromFile* muestra una ventana de diálogo que solicita la ubicación de un archivo en la computadora del usuario y, a diferencia del método *createAtomSetCollectionFromFile*, sólo le regresa el control al sistema después de recibir una respuesta del usuario.

- *jmolBuilderDialogAsk*:

  Este método muestra una ventana de diálogo, retorna un texto y recibe como parámetros un texto que indica el tipo de ventana y un nombre de archivo. *Jmol Builder* ejecuta el método *jmolBuilderDialogAsk*, en lugar del método *dialogAsk*, ambos de la clase *StatusManager*.

También se creó el método *setBuilderModelKitMode* a partir del método *setModelKitMode* de la clase *Viewer*, para activar y desactivar el *Model Kit*; y el método *getStringArrayExtract* para depurar el código fuente.

La creación de la clase *BuilderViewer* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo del control de direcciones.
- Creación y manejo de la tabla periódica interactiva.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del ratón.
- Ejecución de funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del teclado.
- Interacción entre *JME* y *Jmol Builder*.

3.3.7. BuilderMouseManager

Se creó un nuevo archivo *Java* llamado *MouseManager14.java* dentro del paquete *org.jmol.viewer*. En el archivo se declaró la clase *BuilderMouseManager* y se utilizó la información de la clase *MouseManager14* para implementarla. Luego se convirtió la clase *BuilderMouseManager* en derivada de la clase *MouseManager14*.

El diagrama de clases *UML* de la clase *BuilderMouseManager* se muestra en la Fig. 22.

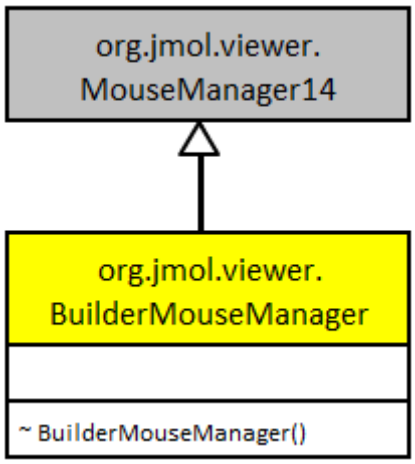


Figura 22: Diagrama de clases *UML* de la clase *BuilderMouseManager*

Desde el constructor de la clase *BuilderMouseManager*, se llamó a una nueva versión del constructor de la clase *MouseManager14*, que recibe un parámetro booleano adicional. Se pasó el nuevo parámetro con un valor de verdad de *true*. Los demás parámetros que recibió el constructor fueron los mismos del constructor de *BuilderMouseManager*.

Como la clase *BuilderMouseManager* es derivada de la clase *MouseManager14*, se pensó en eliminar la declaración de sus atributos públicos y protegidos, pero la clase *MouseManager14* no tenía atributos con esos modificadores de acceso.

La creación de la clase *BuilderMouseManager* permitió ejecutar funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del ratón.

3.3.8. BuilderActionManager

Se creó un nuevo archivo *Java* llamado *BuilderActionManager.java* dentro del paquete *org.jmol.viewer*. En el archivo se declaró la clase *BuilderActionManager* y

se utilizó la información de la clase *ActionManager* para implementarla. Luego se convirtió la clase *BuilderActionManager* en derivada de la clase *ActionManager*.

El diagrama de clases *UML* de la clase *BuilderActionManager* se muestra en la Fig. 23.

Como la clase *BuilderActionManager* es derivada de la clase *ActionManager*, se eliminó la declaración de sus atributos públicos y protegidos, y los métodos finales.

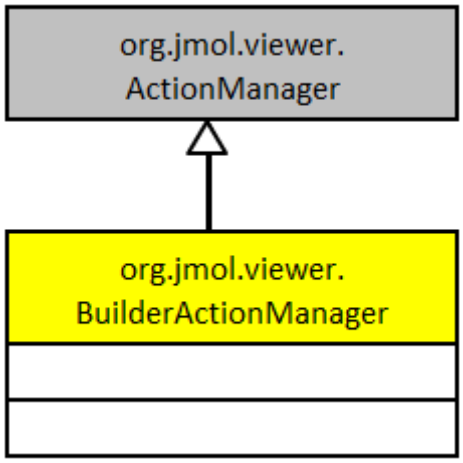


Figura 23: Diagrama de clases *UML* de la clase *BuilderActionManager*

La creación de la clase *BuilderActionManager* permitió ejecutar funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del ratón.

3.3.9. BuilderActionManagerMT

Se creó un nuevo archivo *Java* llamado *BuilderActionManagerMT.java* dentro del paquete *org.jmol.viewer*. En el archivo se declaró la clase *BuilderActionManagerMT* y se utilizó la información de la clase *ActionManagerMT* para implementarla. Luego se convirtió la clase *BuilderActionManagerMT* en derivada de la clase *BuilderActionManager* y se le indicó a la clase que importara la clase *BuilderActionManager* en lugar de la clase *ActionManager.*

El diagrama de clases *UML* de la clase *BuilderActionManagerMT* se muestra en la Fig. 24.

La creación de la clase *BuilderActionManagerMT* permitió ejecutar funcionalidades exclusivas de *Jmol Builder* luego de capturar eventos del ratón.

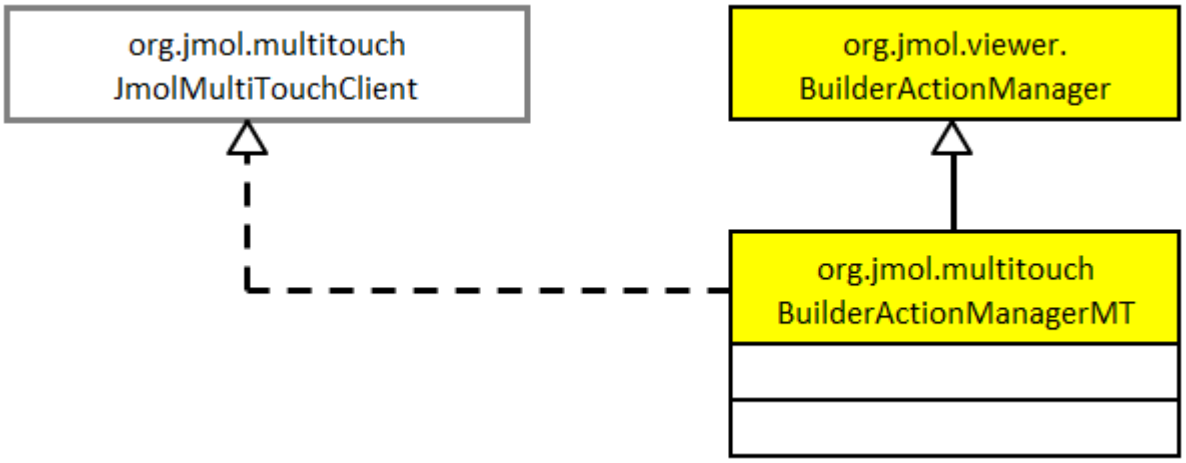


Figura 24: Diagrama de clases *UML* de la clase *BuilderActionManagerMT*

3.3.10. JmolPeriodicTableInterface

Se creó un nuevo archivo *Java* llamado *JmolPeriodicTableInterface.java* dentro del paquete *org.jmol.api*. En el archivo se declaró la interfaz *JmolPeriodicTableInterface* y se utilizó la información de la interfaz *JmolScriptEditorInterface*, dentro del mismo paquete, para implementarla.

El diagrama de clases *UML* de la interfaz *JmolPeriodicTableInterface* se muestra en la Fig. 25.

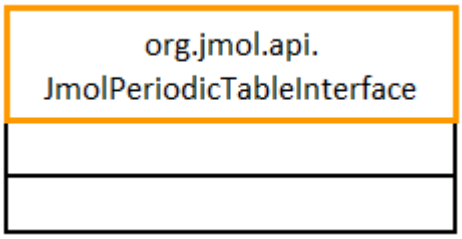


Figura 25: Diagrama *UML* de la interfaz *JmolPeriodicTableInterface*

En la interfaz se declararon métodos para la visibilidad de la tabla periódica.

La creación de la interfaz *JmolPeriodicTableInterface* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.
- Creación y manejo de la tabla periódica interactiva.

3.3.11. PeriodicTable

Se creó un nuevo archivo *Java* llamado *PeriodicTable.java* dentro del paquete *org.jmol.console*. En el archivo se declaró la clase *PeriodicTable* y se utilizó la información de la clase *ScriptEditor*, dentro del mismo paquete, para implementarla. Se modificó de forma que fuera una clase derivada de la clase *JDialog*, que se encuentra dentro del paquete de *Java* llamado *javax.swing*.

Se agregaron nuevos atributos para la construcción y manejo de la tabla periódica. También se crearon nuevos atributos para la comunicación de la tabla con la barra de herramientas. La mayoría de los atributos se inicializaron en el constructor de *PeriodicTable*.

El diagrama de clases *UML* de la clase *PeriodicTable* se muestra en la Fig. 26.

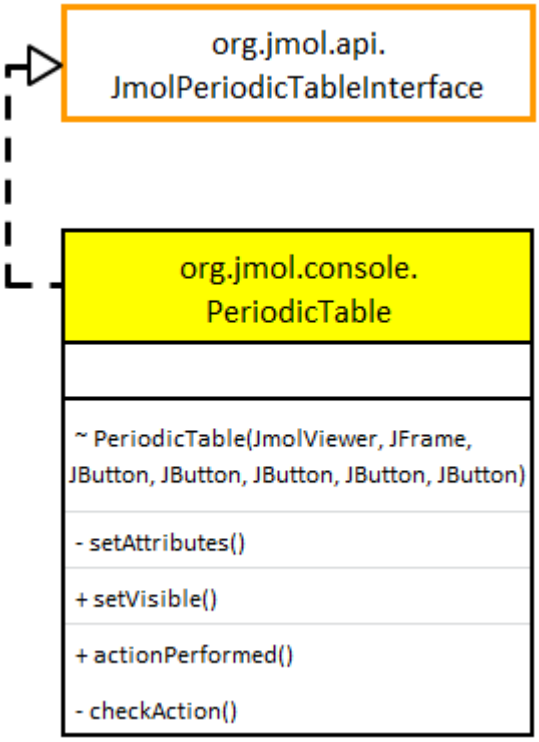


Figura 26: Diagrama de clases *UML* de la clase *PeriodicTable*

Se implementaron los siguientes métodos:

- *PeriodicTable*:

  Es un constructor de esta clase. Como todo constructor, retorna una nueva instancia de la clase a la que pertenece. Recibe como parámetros: una instancia de la clase *JmolViewer*, una instancia de la clase *JFrame*, que es

derivada de *Java*; y referencias a los botones "*Most used atom C*", "*Most used atom H*", "*Most used atom O*", y "*Currently selected atom to add*" de la barra de herramientas. Antes de retornar una instancia de *PeriodicTable*, se ejecuta el método *setAttributes* de esta clase, se inicializan varios atributos de *PeriodicTable* relacionados con el formato de los botones al ser activados o desactivados, se desactivan los botones de la barra de herramientas pasados por parámetro, se construye la ventana con el método *layoutWindow* de esta clase, estableciendo una relación entre todos los botones con el método *actionPerformed* de esta clase, y se establece el tamaño, la posición y las restricciones de la ventana.

- *setAttributes*:

  Inicializa atributos de esta clase relacionados con el formato estándar de la ventana y con la captura de los eventos de los botones de la ventana. No retorna valor alguno y no recibe parámetros.

- *setVisible*:

  Muestra u oculta la ventana de la tabla periódica. No retorna valor alguno y recibe como parámetro un valor booleano que determina si se muestra u oculta la ventana. En el cuerpo de este método se ejecuta el método *setVisible* de la clase base *JDialog* pasándole por parámetro el valor booleano dado.

- *actionPerformed*:

  Da funcionalidad a todos los botones de la ventana de la tabla periódica, ejecutándose cada vez que uno de ellos es presionado. No retorna valor alguno y recibe como parámetro una instancia de una clase derivada de *Java* llamada *ActionEvent*, que contiene una referencia al botón que fue presionado. En el cuerpo de este método se ejecuta el método *checkAction* de esta clase pasándole por parámetro la instancia dada.

- *checkAction*:

  Es ininterrumpido y les da funcionalidad a todos los botones de la ventana de la tabla periódica. Es llamado por el método *actionPerformed* de esta clase cada vez que uno de ellos es presionado. No retorna valor alguno y recibe como parámetro una instancia de una clase derivada de *Java* llamada *ActionEvent*, que contiene una referencia al botón que fue presionado.

  Los botones de la ventana de la tabla periódica son:

  - *Elemento i-ésimo*.
  - *Accept*.
  - *Cancel*.

Cada vez que se presiona un botón de la ventana de la tabla periódica se verifica cuál fue presionado. Las operaciones que se ejecutan después dependen del que fue presionado:

- *Elemento i-ésimo*:

  Estos botones capturan el *i*-ésimo elemento de la tabla periódica, donde: $1 \geq i \geq 109$, y liberan el que se haya capturado previamente. Cada valor de *i* representa un símbolo de un átomo en particular, por lo tanto cada botón "*Elemento i*" indica el símbolo *i*-ésimo. Cada vez que se presiona el botón "Elemento i", se activa ese botón, se registra el átomo como seleccionado, y se desactiva el botón previamente seleccionado.

- *Accept*:

  Cambia el valor indicado por el botón "*Currently selected atom to add*" y activa o desactiva los botones "*Most used atom C*", "*Most used atom H*" y "*Most used atom O*" según el elemento que haya sido seleccionado. Primero se verifica si un elemento de la tabla periódica fue seleccionado. Sólo si fue así, se ejecuta el método *evalStringQuiet* de la clase *JmolViewer* pasándole un *script* que le indica a *Jmol* que el elemento seleccionado es el que podrá ser unido al modelo, se indica al usuario con el botón "*Currently selected atom to add*" que el átomo para unir al modelo es el átomo seleccionado y se activan o se desactivan los botones "*Most used atom C*", "*Most used atom H*" y "*Most used atom O*" según el elemento que haya sido seleccionado.

  Luego de verificar si un elemento fue seleccionado, se desactiva el botón "*Show periodic table of elements*", se regresa la tabla periódica a su estado original y se oculta la ventana.

- *Cancel*:

  Cancela cualquier cambio realizado a la ventana de la tabla periódica y la oculta. Primero se desactiva el botón "*Show periodic table of elements*", se oculta la ventana y se desactiva el botón que haya sido seleccionado de la ventana.

Hay más información sobre la ventana de la tabla periódica en el Capítulo IV de este trabajo, titulado "*Pruebas y Resultados*".

La creación de la clase *PeriodicTable* permitió agregar las siguientes funcionalidades a *Jmol*:

- Creación y manejo de la barra de herramientas.

- Creación y manejo de la tabla periódica interactiva.

3.4. Relación de las clases nuevas con Jmol

Las clases que se agregaron a *Jmol* se dividen en dos grupos: clases derivadas de *Jmol* y clases derivadas de *Java*. Ambos grupos junto con las modificaciones realizadas a unas cuantas clases y un par de interfaces de *Jmol* permitieron implementar *Jmol Builder* separado del sistema original.

Como se muestra en la Fig. 27, la relación entre las nuevas clases e interfaces con las de *Jmol* es la siguiente:

3.4.1. JmolBuilderApplet

Hereda de la clase *BuilderAppletWrapper* y sustituye a la clase *JmolApplet*, que es la primera clase que se ejecuta del sistema original.

3.4.2. BuilderAppletWrapper

Hereda de la clase *AppletWrapper* de *Jmol*, usa instancias de la clase *BuilderPanelComponent* y maneja todos los eventos relacionados con la barra de herramientas, el teclado y el control de direcciones.

3.4.3. BuilderPanelComponent

Tiene una relación de asociación con *BuilderAppletWrapper*, siendo uno de los componentes más importantes de la barra de herramientas.

3.4.4. JmolBuilder

Hereda de la clase *Jmol* del sistema original, crea una instancia de la clase abstracta *JmolViewer*, utiliza un método de la clase abstracta *JmolBuilderViewer* y determina el modo en el que inicia *Jmol Builder*.

3.4.5. JmolBuilderViewer

Hereda de la clase *JmolViewer* de *Jmol*, tiene una relación de asociación con *JmolBuilder* porque se utiliza para crear la pantalla de trabajo, utiliza métodos de la clase *BuilderViewer* y se encarga de instanciar la nueva versión de la clase *JmolViewer* de *Jmol*.

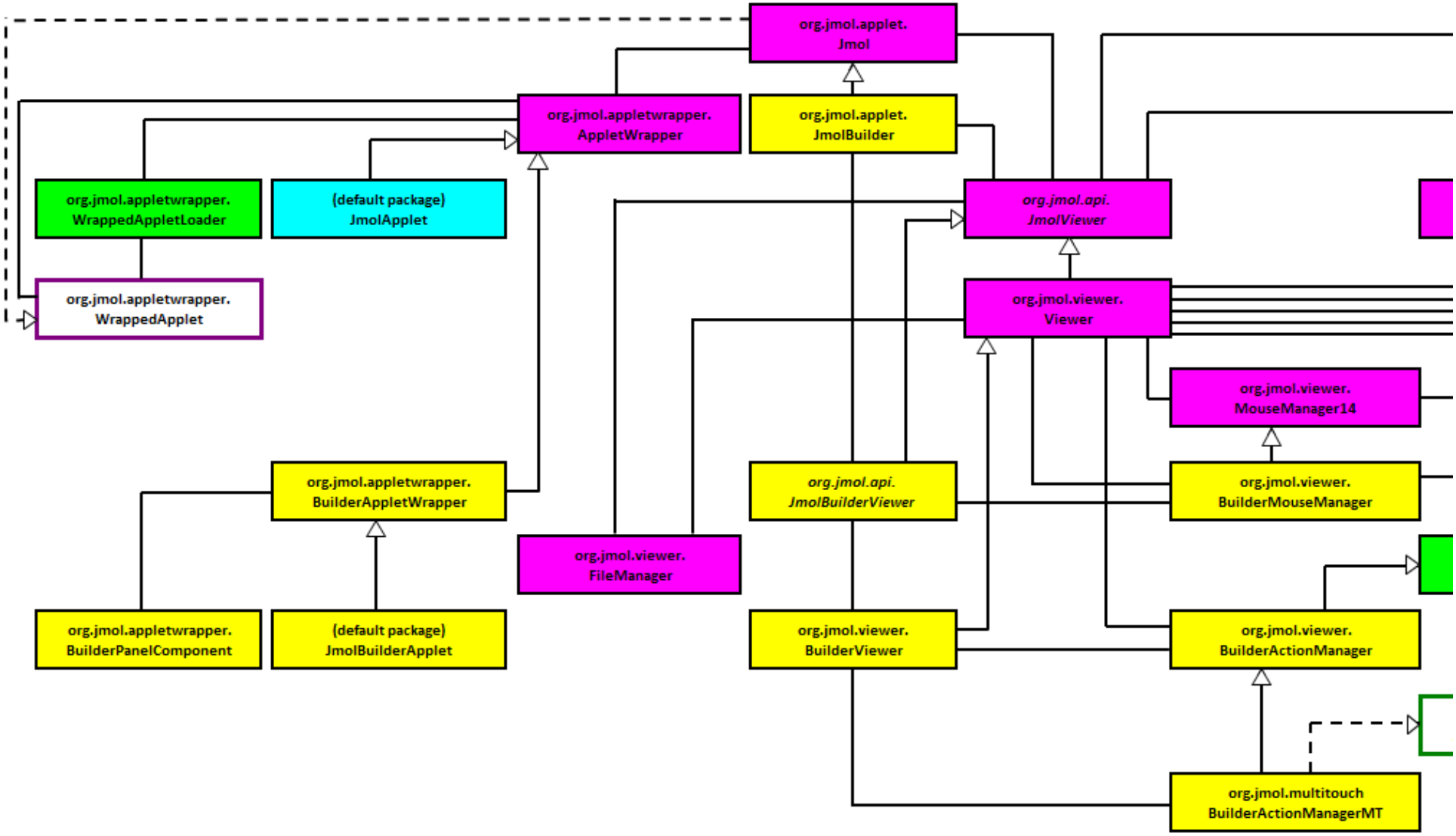


Figura 27a: Diagrama de las clases e interfaces de *Jmol* relacionadas con las nuevas clases e interfaces. Los objetos de color de relleno amarillo representan las nuevas clases, mientras que los que tienen bordes anaranjados son las nuevas interfaces. Las clases de *Jmol* que fueron reemplazadas por nuevas clases tienen un color de relleno azul celeste. El color de relleno verde claro representa las clases de *Jmol* que tienen una relación directa con las nuevas clases y no fueron modificadas; y las interfaces que cumplen con las mismas características tienen bordes de color verde oscuro. Las clases de *Jmol* que fueron modificadas tienen un color de relleno rosado, mientras que las interfaces de *Jmol* que fueron modificadas tienen bordes de color morado. El formato de cursiva representa las clases abstractas.

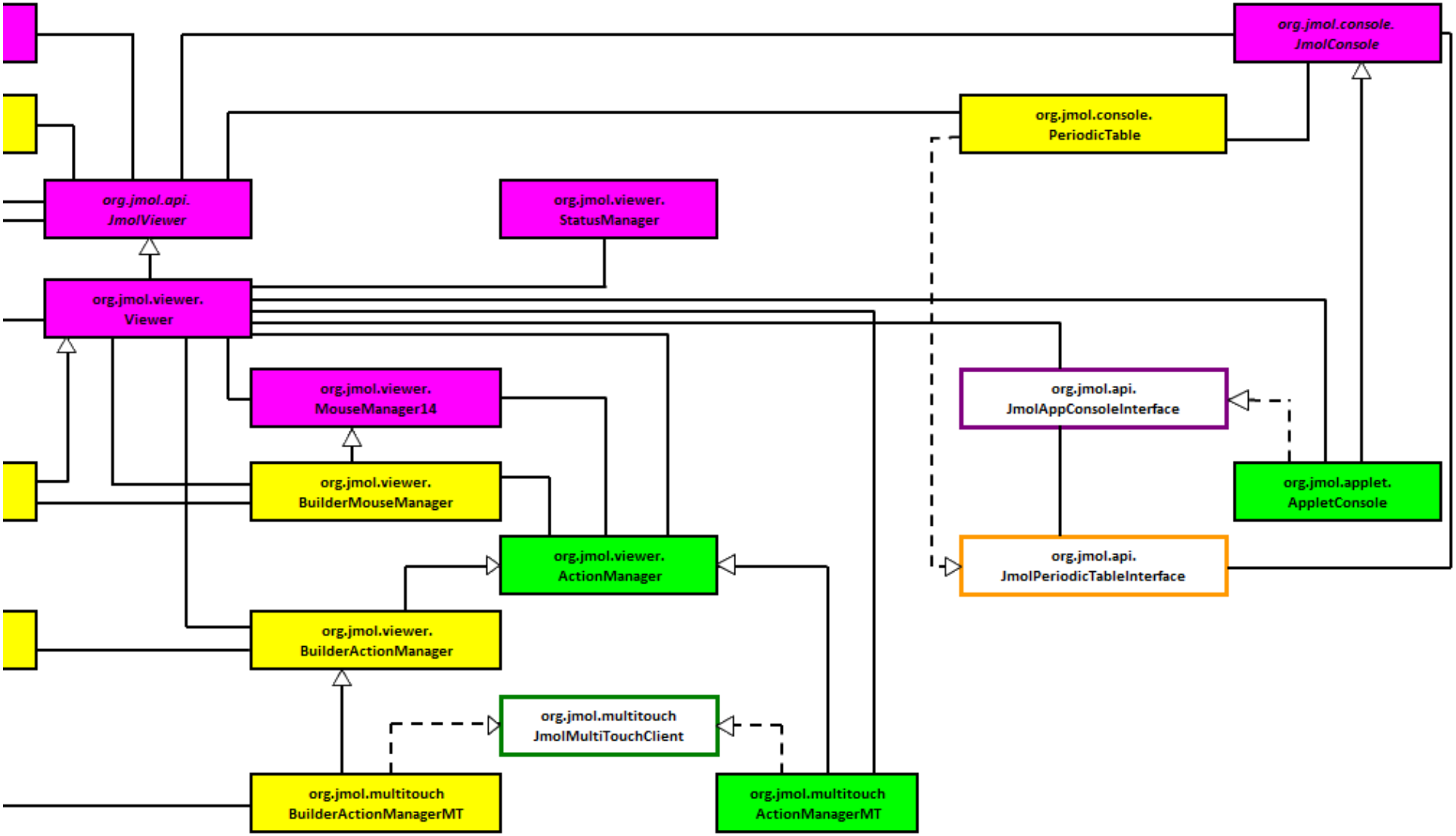


Figura 27b: Diagrama de las clases e interfaces de *Jmol* relacionadas con las nuevas clases e interfaces. Los objetos de color de relleno amarillo representan las nuevas clases, mientras que los que tienen bordes anaranjados son las nuevas interfaces. Las clases de *Jmol* que fueron reemplazadas por nuevas clases tienen un color de relleno azul celeste. El color de relleno verde claro representa las clases de *Jmol* que tienen una relación directa con las nuevas clases y no fueron modificadas; y las interfaces que cumplen con las mismas características tienen bordes de color verde oscuro. Las clases de *Jmol* que fueron modificadas tienen un color de relleno rosado, mientras que las interfaces de *Jmol* que fueron modificadas tienen bordes de color morado. El formato de cursiva representa las clases abstractas.

3.4.6. BuilderViewer

Hereda de la clase *Viewer* del sistema original, tiene una relación de asociación con *JmolBuilderViewer* porque se utiliza para crear una instancia de la clase *JmolViewer* con nuevas funcionalidades; utiliza métodos de las clases *BuilderMouseManager* y *BuilderActionManager* y maneja, entre otros datos, información del modelo, la pantalla de trabajo y los modos del sistema.

3.4.7. BuilderMouseManager

Hereda de la clase *MouseManager14* de *Jmol*, tiene una relación de asociación con *BuilderViewer* porque es utilizada para instanciar una nueva versión de la clase *MouseManager14* y tiene una instancia de la clase *Viewer* y la clase *ActionManager*. Captura todos los eventos del ratón al usar *Jmol Builder*.

3.4.8. BuilderActionManager

Hereda de la clase *ActionManager* del sistema original, tiene una relación de asociación con *BuilderViewer* porque es utilizada para instanciar una nueva versión de la clase *ActionManager* y tiene una instancia de la clase *Viewer*. Maneja algunas de las acciones ejecutadas al usar *Jmol Builder*, luego de capturar los eventos del ratón.

3.4.9. BuilderActionManagerMT

Hereda de la clase *BuilderActionManager*, implementa la interfaz *JmolMultiTouchClient* de *Jmol*, sustituye a la clase *ActionManagerMT* y es utilizada por la clase *BuilderViewer* en una de las posibles instanciaciones de la nueva versión de la clase *ActionManager*.

3.4.10. JmolPeriodicTableInterface

Tiene una relación de asociación con la interfaz *JmolAppConsoleInterface* porque se hace referencia a ella en la declaración de un método de la interfaz que retorna la tabla periódica. También está asociada a la clase abstracta *JmolConsole* de *Jmol* porque es utilizada en la implementación del método declarado en la interfaz *JmolAppConsoleInterface*.

3.4.11. PeriodicTable

Implementa la interfaz *JmolPeriodicTableInterface*, tiene una relación de asociación con la clase *JmolConsole* del sistema original porque es un

componente de esa clase, y tiene una instancia de la clase *JmolViewer*. Construye los componentes y maneja todos los eventos relacionados con la tabla periódica interactiva.

3.5. Cambios sobre el sistema Jmol original

Se modificaron las clases de *Jmol* tomando en cuenta cuatro elementos importantes: los constructores, los modificadores de acceso, los métodos y los atributos.

- *Constructores*:

  Tomando en cuenta los cambios que se realizaron a los constructores de las clases que derivaron de clases del sistema original, se crearon nuevas versiones de los constructores de sus clases bases respectivas. Las nuevas versiones son protegidas y reciben un parámetro booleano. Las instrucciones que ejecutan son las mismas que las de los constructores originales pero sólo son ejecutadas si el valor de verdad del parámetro es falso. Los constructores de las clases derivadas no dependen de los nuevos constructores que deben llamar, porque el objetivo de todas las clases derivadas es sustituir a su clase base. Si se llaman los nuevos constructores pasándole por parámetro un valor booleano falso, se crean conflictos entre *Jmol Builder* y *Jmol*. Por ejemplo, si el constructor de *BuilderMouseManager* llama a la nueva versión del constructor de la clase *MouseManager14*, cada una captura los eventos del ratón al mismo tiempo.

- *Modificadores de acceso*:

  Para cambiar el modificador de acceso de un constructor, se creó una nueva versión de él que recibe todos los parámetros del constructor original y un parámetro booleano. Ese último parámetro indica si el constructor fue llamado por la instancia de una clase derivada o no.

  Tomando en cuenta la forma en que se implementaron los métodos de las clases exclusivas de *Jmol Builder* para cambiar el modificador de acceso de los métodos de sus clases bases, se crearon nuevos métodos declarados de la misma manera pero con cuerpos que siempre devuelven respuestas de error, es decir, respuestas que los métodos devolverían si no se cumplieran los requisitos para ejecutarlos. En el caso de los métodos que no devuelven respuesta, las nuevas versiones de los métodos son vacías. De esa forma, si se utilizan esos métodos con las clases bases, no se cambia la funcionalidad del sistema original.

  Luego, se implementaron otros métodos en otras clases e interfaces que permitieran ejecutar el nuevo método de la clase derivada.

Si se toma el ejemplo que se mencionó en el apartado "*Clases Exclusivas de Jmol Builder*" sobre los modificadores de acceso, la versión del método *setBuilderModelKitMode* dentro de la clase *Viewer*, recibe los mismos parámetros, devuelve los resultados del mismo tipo que el método *setModelKitMode* de la clase *Viewer*, pero no ejecuta instrucciones.

- *Métodos*:

Todos los métodos que se crearon en las clases de *Jmol* devuelven las respuestas de error que se mencionaron en el apartado anterior. En el caso de los métodos que no devuelven respuesta, tienen cuerpos vacíos.

La razón de la creación de los métodos en las clases del sistema original fue precisamente la separación de *Jmol Builder* y *Jmol*. Al principio no se necesitaron crearlos porque las nuevas funcionalidades que se implementaron eran ejecutadas por nuevas versiones de métodos ya declarados en las clases originales. Sin embargo, a medida que pasó el tiempo, se tuvieron que crear nuevos métodos en las clases exclusivas de *Jmol Builder* que tenían que ser accedidas por el resto de las clases de *Jmol*. Por un lado se pudo haber creado una nueva versión por cada clase e interfaz del sistema original para que utilizara los nuevos métodos sin haber conflictos pero esa no era una solución práctica para futuros cambios al sistema, sin tomar en cuenta el tiempo de desarrollo ligado a ella. Entonces se decidió crear los mismos métodos en las clases originales donde cada uno devuelve una respuesta de error si necesita devolver alguna. Luego, se implementó *Jmol Builder* de forma que las clases exclusivas de *Jmol Builder* utilizaran las instancias de las clases de *Jmol* pero las inicializaran con constructores o con métodos especiales (como es el caso de *allocateViewer*) de las nuevas versiones de esas clases. No se tuvieron que crear nuevas versiones de la mayoría de las clases originales porque para ellas la ejecución de los métodos de las clases exclusivas de *Jmol Builder* fue transparente. De esa forma *Jmol* tuvo y sigue teniendo acceso a todas las funcionalidades de *Jmol Builder* sin haber conflictos y, en caso de inicializar una instancia con el constructor original de su clase respectiva y ejecutar los nuevos métodos, devuelven respuestas como si no se hubieran cumplido los requisitos para ejecutarlos.

La solución desarrollada tiene un caso especial que tiene que ver con las clases *JmolViewer*, *Viewer* y *BuilderViewer*.

Varias clases de *Jmol* tienen atributos de la clase *Viewer*. Por lo tanto, para ejecutar nuevas funcionalidades en las clases derivadas de esas clases sin entrar en conflicto con el resto del sistema, simplemente se inicializaron esos atributos con el constructor de la clase *BuilderViewer* y se crearon los nuevos métodos tanto en la clase *BuilderViewer* como en la clase *Viewer*. Sin embargo, hay clases del sistema que tienen atributos que son instancias de *JmolViewer* con los que acceden métodos de la clase *BuilderViewer*. Como la clase *BuilderViewer* deriva de la clase *Viewer*, y la clase *Viewer* deriva a su vez

de la clase *JmolViewer*, en esos casos se crearon los nuevos métodos en las clases *JmolViewer*, *Viewer* y *BuilderViewer*.

- *Atributos:*

  Sólo en una de las clases de *Jmol* se necesitó crear un nuevo atributo para incorporar la funcionalidad de la tabla periódica interactiva. En el resto de las clases no fue necesario ni crear ni modificar atributos.

Las interfaces de *Jmol* que se modificaron fueron *WrappedApplet* y *JmolAppConsoleInterface*. Se importaron nuevas clases desde ellas y se declararon nuevos métodos. Luego se asignaron a los métodos nombres diferentes al resto de los métodos de la interfaz para evitar conflictos con el sistema original.

3.5.1. Modificación de la clase Jmol

Se importó la clase *JButton*, que se encuentra dentro del paquete *javax.swing*, y se declararon métodos para acceder a ciertos métodos de la clase *BuilderViewer* desde la clase *JmolBuilder*.

3.5.2. Modificación de la clase AppletWrapper

Se implementó la nueva versión del constructor de la clase *AppletWrapper* que se discutió en el apartado anterior. En el cuerpo del método se insertaron las instrucciones del constructor original. Luego se agruparon en un segmento de código que sólo se ejecuta si el valor de verdad del nuevo parámetro es falso.

3.5.3. Modificación de la clase JmolViewer

Se importó la clase *JButton*, que se encuentra dentro del paquete *javax.swing*, y se declararon métodos para acceder a ciertos métodos de la clase *BuilderViewer* desde la clase *JmolBuilder*.

3.5.4. Modificación de la clase Viewer

Se implementó la nueva versión del constructor de la clase *Viewer* que se discutió en el apartado anterior. En el cuerpo del método se insertaron las instrucciones del constructor original. Luego se agruparon en un segmento de código que sólo se ejecuta si el valor de verdad del nuevo parámetro es falso.

Se importó la clase *JButton*, que se encuentra dentro del paquete *javax.swing*, y se declararon métodos para acceder a ciertos métodos de la clase *BuilderViewer* desde la clase *JmolBuilder*.

3.5.5. Modificación de la clase MouseManager14

Se implementó la nueva versión del constructor de la clase *MouseManager14* que se discutió en el apartado anterior. En el cuerpo del método se insertaron las instrucciones del constructor original. Luego se agruparon en un segmento de código que sólo se ejecuta si el valor de verdad del nuevo parámetro es falso.

3.5.6. Modificación de la interfaz WrappedApplet

Se importó la clase *JButton*, que se encuentra dentro del paquete *javax.swing*, y se declararon métodos dentro de la interfaz para acceder ciertos métodos de la clase *BuilderViewer* desde la clase *JmolBuilder*.

3.5.7. Modificación de la interfaz JmolAppConsoleInterface

Se importó la clase *JButton*, que se encuentra dentro del paquete *javax.swing*, y se declaró el método *getPeriodicTable* dentro de la interfaz, que recibe un atributo booleano y referencias a algunos botones de la barra de herramientas, y devuelve una instancia a la interfaz *JmolPeriodicTableInterface*.

3.5.8. Modificación de la clase JmolConsole

Se importó la interfaz *JmolPeriodicTableInterface*, que se encuentra dentro del paquete *org.jmol.api*, y se creó una instancia de la clase *PeriodicTable*.

También se implementó el método *getPeriodicTable* dentro de la interfaz *JmolAppConsoleInterface*, que recibe referencias a algunos botones de la barra de herramientas y devuelve una referencia a la interfaz *JmolPeriodicTableInterface* instanciada con el constructor de la clase *PeriodicTable*. El método retorna una nueva instancia de la clase *PeriodicTable* si no la ha retornado antes o, en caso contrario, retorna la instancia que se devolvió la primera vez.

3.5.9. Modificación de la clase FileManager

Se agregaron nuevas funcionalidades a la clase *FileManager*, que se encuentra dentro del paquete *org.jmol.viewer*, a través de los siguientes métodos:

- *jmolBuilderClassifyName*:

  Retorna una lista de textos y recibe como parámetros un nombre a clasificar y un valor booleano que indica en algunos casos si se retorna una lista vacía o no. *Jmol Builder* ejecuta las mismas instrucciones que el método *classifyName* de esta clase, excepto en el caso en que el nombre dado es "*?*". Cuando eso sucede, se ejecuta el método *jmolBuilderDialogAsk* de la clase *BuilderViewer* en lugar del método *dialogAsk* de la clase *Viewer*. La diferencia entre ambas versiones es que la de *Jmol Builder* no continúa hasta que el método llamado termine de ejecutarse, pero la original se ejecuta en paralelo al sistema. La ejecución del método *jmolBuilderDialogAsk* retorna un valor vacío si la operación es cancelada al igual que la versión original.

- *jmolBuilderCreateAtomSetCollectionFromFile*:

  Retorna una instancia de la clase *Object* y recibe como parámetros: un nombre de archivo, una instancia llamada *htParams* que almacena una lista de parámetros y un valor booleano que indica si se unirá el archivo al modelo. *Jmol Builder* ejecuta las mismas instrucciones que el método *createAtomSetCollectionFromFile* de esta clase, excepto en los siguientes casos:

  - *Clasificación del nombre dado*:

  Cuando se clasifica el nombre del archivo dado con el método *jmolBuilderClassifyName* en lugar del método *classifyName,* ambos de esta clase. La diferencia entre ambas versiones es que la de *Jmol Builder* no continúa hasta que el método llamado termine de ejecutarse, pero la original se ejecuta en paralelo al sistema.

  - *Inicialización de instancia de la clase FileReader*:

  En las últimas tres instrucciones del método *createAtomSetCollectionFromFile* se inicializa una instancia, se ejecuta el método *run* y se retorna un atributo de la clase *FileReader*. La versión de *Jmol Builder* no ejecuta esas instrucciones sino que devuelve una respuesta vacía. Se encontró que de esa forma, si ocurría un error durante la recarga del modelo, no se cargaba un CH_4 sobre el modelo que se estuviera mostrando en la pantalla de trabajo.

3.5.10. Modificación de la clase StatusManager

Se agregaron nuevas funcionalidades a la clase *StatusManager*, que se encuentra dentro del paquete *org.jmol.viewer*, a través del nuevo método *jmolBuilderDialogAsk*. Se utilizó la información de un método de la clase llamado *dialogAsk* para implementarlo.

El método *jmolBuilderDialogAsk* recibe el tipo de diálogo y el nombre del archivo, y devuelve el nombre del archivo de salida según el caso. El tipo de diálogo indica si se desea cargar un archivo, guardar la información del modelo sobre la pantalla de trabajo o guardar una captura de pantalla del modelo. Si el tipo de diálogo no es válido, el método devuelve un nombre vacío.

La versión del sistema original consiste en la ejecución del método *invokeAndWait* de la clase *SwingUtilities*, que se encuentra dentro del paquete *javax.swing*. El método *invokeAndWait* recibe una instancia dinámica de la clase *Runnable* cuyo método *run* es el núcleo del procesamiento de *dialogAsk*. Por cada tipo de diálogo, se devuelve el control al método *dialogAsk* una vez se obtiene el nombre del archivo de salida, y luego se retorna ese nombre al método que haya invocado al método *jmolBuilderDialogAsk*.

La versión de *Jmol Builder* sólo ejecuta el conjunto de instrucciones dentro del cuerpo del método *run*, de forma que por cada tipo de diálogo, una vez que se obtiene el nombre del archivo de salida, simplemente se retorna ese nombre al método que haya invocado al método *jmolBuilderDialogAsk*.

Ambas versiones de los métodos pueden llegar a lanzar excepciones de forma que se colocaron instrucciones para poder capturarlas.

La diferencia entre la versión de *Jmol Builder* y la versión original es que *jmolBuilderDialogAsk* no le regresa el control al sistema hasta que reciba una respuesta del usuario, pero la versión original se ejecuta en paralelo al sistema.

3.6. Nuevas imágenes

Se agregaron imágenes *PNG* dentro del paquete *org.jmol.modelkit.images* para la barra de herramientas y el control de direcciones:

- *addBenzeneButton*.
- *addCH4Button*.
- *addFragmentButton*.
- *addTriangleButton*.
- *attachOrReplaceAtomModeButton*.

- *bottomCenterDirButton*.
- *bottomLeftDirButton*.
- *bottomRightDirButton*.
- *cartesianCoordinateSystemButton*.
- *centerLeftDirButton*.
- *centerRightDirButton*.
- *clearButton*.
- *controlFragmentModeButton*.
- *deleteAtomModeButton*.
- *dragMoleculeModeButton*.
- *imageButton*.
- *increasePrecisionButton*.
- *measurementModeButton*.
- *openButton*.
- *otherAtomOrFragmentButton*.
- *rotateClockwiseButton*.
- *rotateCounterclockwiseButton*.
- *saveButton*.
- *showLabelsButton*.
- *topCenterDirButton*.
- *topLeftDirButton*.
- *topRightDirButton*.

Una vez se incorporaron estas imágenes al sistema, se pudieron acceder desde la clase *BuilderAppletWrapper* durante la creación de la barra de herramientas.

3.7. Carpeta de modelos

Se creó un nuevo paquete llamado *org.jmol.model.kit.models* dentro del cual se agregaron varios modelos con extensión *mol*, incluyendo todos los que pueden ser agregados por el sistema cerca del modelo que se esté mostrando en pantalla. Actualmente, sólo los siguientes modelos pueden ser agregados como fragmentos:

- *benzene*.
- *CH4*.

- *triangle*.

Los otros modelos que podrían ser utilizados para ser agregados al modelo que se esté mostrando en pantalla son los siguientes:

- *aspirin*.
- *coronene*.
- *square*.

Una vez se incorporaron los modelos al sistema, se pudieron acceder y agregar al modelo actual desde la clase *BuilderAppletWrapper* durante la ejecución del método *actionPerformed*.

3.8. Cambios sobre la configuración de Eclipse

Se realizaron modificaciones sobre la configuración de la ejecución del *applet* por parte de *Eclipse*, que utiliza una simulación de un navegador llamada *AppletViewer*. Se utilizó como guía la biblioteca escrita en *JavaScript* de *Jmol.* Le fueron pasados los siguientes parámetros:

- *progressbar* con el valor "*true*".
- *boxfgcolor* con el valor "*white*".
- *boxmessage* con el valor "*Downloading JmolBuilderApplet ...*".
- *progresscolor* con el valor "*blue*".
- *syncId* con el valor "*0*".
- *signed* con el valor "*true*".

Si no se realizan estos cambios, el sistema, interpretado por *AppletViewer*, muestra la advertencia que se muestra en la Fig. 28 y no permite ni que se utilice el *Model Kit* ni que se active el *applet* como cifrado.

Figura 28: Mensaje de advertencia sobre el parámetro *progressbar*.

3.9. Archivos de compilación

Se crearon nuevos archivos para la generación de los ejecutables de *Jmol Builder*.

- En la carpeta *Jmol/manifiest* se agregó un nuevo archivo de texto llamado *appletBuilder0.txt* con la información del archivo *applet0.txt*, que se encuentra en esa misma carpeta. Luego se modificó reemplazando el texto *JmolApplet* por *JmolBuilderApplet*. En esa misma carpeta se agregó un nuevo archivo de texto llamado *appletBuilderMonolithic.txt* con la información del archivo *appletMonolithic.txt*, que también se encuentra en esa misma carpeta. De igual forma se modificó ese archivo reemplazando el texto *JmolApplet* por *JmolBuilderApplet*.
- En la carpeta *Jmol* se agregó un nuevo archivo *.classes* llamado *appletBuilder.classes* al que se le agregó la información del archivo *applet.classes*, que se encuentra en esa misma carpeta. Luego se modificó el archivo reemplazando el texto *JmolApplet.java* por *JmolBuilderApplet.java*.
- En la carpeta *Jmol/appletweb* se agregó un nuevo archivo *JavaScript* llamado *JmolBuilder.js* con la información del archivo *Jmol.js*, que se encuentra en esa misma carpeta. Después se modificó el archivo reemplazando el texto '*JmolApplet*' por el texto '*JmolBuilderApplet*'.
- En la carpeta *Jmol* se agregó un nuevo archivo *XML* llamado *buildBuilder.xml* con la información del archivo *build.xml*. Dentro de ese archivo hizo referencia a la clase *JmolBuilderApplet* en vez de la clase *JmolApplet*, y a los archivos de texto y *JavaScript* que se agregaron en lugar de los archivos a partir de los

cuales fueron creados, con excepción del archivo *JavaScript Jmol.js*. También se incluyeron referencias al nuevo paquete *org.jmol.modelkit.models*.

El archivo de compilación *build.xml* se encarga de generar los ejecutables del sistema original. La información de los archivos *applet0.txt, appletMonolithic.txt, applet.classes* indican los paquetes, clases e interfaces involucradas en el proceso y los paquetes del sistema. El archivo *applet0.txt* indica en especial la clase con la que inicia el sistema original, *JmolApplet*.

En el caso de *Jmol Builder*, el archivo de compilación *buildBuilder.xml* se encarga de generar los ejecutables del sistema. De la misma forma, la información de los archivos *appletBuilder0.txt, appletBuilderMonolithic.txt, appletBuilder.classes* indican los paquetes, clases e interfaces involucradas en el proceso. El archivo *appletBuilder0.txt* también indica la clase con la que inicia el sistema, que es *JmolBuilderApplet*.

La biblioteca *JavaScript* del sistema original, *Jmol.js* fue desarrollada para acceder la clase *JmolApplet*, una vez dada su ubicación desde la página web, y establecer la comunicación entre *Jmol* y la página web. Por esa razón se modificó el archivo *JmolBuilder.js* para que accediera la clase *JmolBuilderApplet* en lugar de su versión original.

Los ejecutables generados por los archivos de compilación que se crearon deben ser incorporados en las páginas web para utilizar *Jmol Builder*.

3.10. Instrucciones sobre las páginas web que ejecuten Jmol Builder

Se implementó una página web que utiliza tanto *Jmol Builder* como *JME*. Se agregaron las siguientes instrucciones a esa página:

- En el encabezado de la página se insertó una etiqueta *script* de tipo "*text/javascript*" en la que se especificó la ubicación del archivo *JmolBuilder.js* y otra etiqueta del mismo tipo en la que se agregó la ubicación un nuevo archivo *JavaScript* que se llamó *JmolBuilderWithJme.js*.

- Se insertó una etiqueta *applet*, en el cuerpo de la página, en la que se especificó la ubicación del archivo ejecutable de *JME*. También se insertaron etiquetas *input* para agregar dos botones que se llamaron "*Switch to Jmol*" y "*Switch to JME*".

- En el cuerpo de la página se ejecutaron las siguientes funciones de la biblioteca *JavaScript* de *Jmol Builder*, dentro de otra etiqueta *script*:

- *jmolInitialize*

  Esta función se mandó a ejecutar pasándole la ubicación de la clase *JmolBuilderApplet* y un valor de verdad de *true*.

- *jmolSetParameter*

  Esta función se mandó a ejecutar pasándole los textos "*UseCommandThread*" y "*true*".

- *jmolApplet*

  Esta función se mandó a ejecutar pasándole las dimensiones del *applet* con el valor "*100%*".

- Se ejecutaron las siguientes funciones, en el cuerpo de la página, dentro del archivo *JmolBuilderWithJme.js*, al presionar botones en la interfaz:

  - *useJmolBuilder*

    Esta función se mandó a ejecutar sin pasarle parámetros al presionar el botón "*Switch to Jmol*". Dentro del cuerpo de esta función se ejecuta la función *jmolScript* de la biblioteca *JavaScript* de *Jmol Builder* pasándole por parámetro el *script* "*#JMOLBUILDER_RESTART_EDITOR*", se extrae la información del modelo mostrado por *JME* y se procede a verificar si la información es vacía. Sólo si no lo es, se ejecuta la función *jmolScriptWait* de la biblioteca *JavaScript* de *Jmol Builder* pasándole por parámetro un *script* que carga el modelo representado por la información en el *applet* de *Jmol Builder* y se ejecuta la función *jmolScript*, que hace una optimización preliminar del modelo cargado. Luego se oculta el *applet JME* y se muestra el *applet Jmol Builder*.

  - *useJme*

    Esta función se mandó a ejecutar sin pasarle parámetros al presionar el botón "*Switch to JME*". Dentro de esta función se extrae una instancia de la clase *BuilderViewer* con la función *jmolGetPropertyAsJavaObject* de la biblioteca *JavaScript* de *Jmol Builder* pasándole por parámetro el texto "*jmolViewer*", se registra la instancia en una variable que se llamó *viewer* y se procede a verificar si la información es vacía. Sólo si no lo es, se extrae la información del modelo mostrado por *Jmol Builder* con el método *getModelExtract* de la clase *BuilderViewer*, llamado a través de *viewer*, y se procede a verificar si la información es vacía. Si y sólo si no lo es, se utiliza para cargar el modelo representado por la información en el *applet JME*. Luego se oculta el *applet Jmol Builder* y se muestra el *applet JME*.

Hay más información sobre las instrucciones y los botones que se agregaron a la página web en el Capítulo IV de este trabajo, titulado "*Pruebas y Resultados*".

3.11. Página web

Jmol Builder es un sistema que sólo puede ejecutarse en el contexto de otro programa como, por ejemplo, el programa *AppletViewer* del *IDE Eclipse* o un navegador web. A continuación se explicará por qué se utilizaron las instrucciones mencionadas en el apartado "*Instrucciones sobre las páginas web que ejecuten Jmol Builder*":

- *Acceso a la biblioteca JavaScript*:

  Se insertó la ubicación del archivo *JmolBuilder.js* en la etiqueta *script* desde el encabezado de la página web para importar la biblioteca *JavaScript* de *Jmol Builder*.

- *Funciones dentro del cuerpo de la página web*:

  - *jmolInitialize*

    Esta función recibe la ubicación de la clase con la que inicia *Jmol Builder* y un parámetro booleano que indica si se activa el *applet* como cifrado o no; y no retorna valor alguno. La biblioteca *JavaScript* la utiliza para iniciar la comunicación entre *Jmol Builder* y la página web según la clase dada.

    Si se activa el *applet* como cifrado, el sistema es capaz de cargar un archivo ubicado en la computadora del usuario y de guardar un archivo en la misma computadora; ambas funcionalidades son de gran importancia para el buen funcionamiento del sistema. Sin embargo, una vez activado, el *applet* cifrado requiere el permiso del usuario final para ser ejecutado, permiso que solicita con un mensaje de advertencia. Resulta que la autoridad comercial, *jmol.org*, y el certificado de seguridad de *Jmol* son considerados como no confiables porque los creadores del sistema no pudieron firmar cada versión de *Jmol* con una autoridad comercial confiable, debido a su alto costo. Dicho eso, nosotros y los investigadores del IVIC consideramos que el *applet* cifrado es confiable.

  - *jmolSetParameter*

    Esta función recibe dos parámetros: el nombre de un parámetro y su valor. Le fue pasado el parámetro "*UseCommandThread*" porque determina si algunos comandos provenientes de la interfaz de la página (como un botón desplegado por la función *jmolButton* de la biblioteca) usan o no otro hilo. Como consecuencia, los comandos en forma de *script* de *Jmol*

provenientes de la interfaz de la página web pueden ser negados de acceso a ciertos archivos, por la política de seguridad de *Java*. Cuando se le asigna al parámetro el valor de "*true*", se evitan algunas restricciones de *Java* como la carga de un archivo ubicado en una carpeta diferente a la carpeta en la que se encuentre el archivo *HTML* que ejecute el método. De esa forma se pudo ejecutar una página web de prueba con el *applet* de *Jmol Builder* incrustado en ella, desde la computadora de un cliente o un servidor, usando la estructura de archivos y carpetas que se muestra en la Fig. 29.

- *jmolApplet*

  Esta función define e inserta en la página el *applet* de *Jmol Builder*. Recibe el tamaño del *applet* y dos parámetros opcionales: un guión o conjunto de instrucciones y un sufijo para el nombre y otros atributos relacionados con el *applet*, asignados por la etiqueta *HTML* que contenga el *applet*. No se entrará en detalle con el uso de los parámetros opcionales porque no fueron necesarios para este trabajo de investigación. El tamaño pasado por parámetro puede indicarse como:

  - Un solo valor en caso que se desee que el *applet* sea cuadrado.
  - Una matriz de dos elementos en caso que se desee que el *applet* sea rectangular.

  Los valores de las dimensiones pueden indicarse como:

  - Un número entero de píxeles (entre 25 y 2000).
  - Un valor de porcentaje, terminado en % y rodeado por comillas, que es interpretado por el navegador como un porcentaje de las dimensiones de la capa que engloba el *applet* (cualquier etiqueta que defina una división o una sección de la página web). Dependiendo de cuál sea esa capa y de sus propiedades, eso permite tener un *applet* que se redimensione dinámicamente cuando cambie el tamaño de la ventana del navegador.
  - Un número entre 0 y 1 que, multiplicado por 100, se interpreta como porcentaje.

Figura 29: Estructura de archivos y carpetas para utilizar *Jmol Builder*

Se especificaron las dimensiones del *applet* con el valor "*100%*" para que abarcaran todo el contenido de la capa que englobara el *applet*.

Es importante tomar en cuenta que el orden en que se ejecutaron estas funciones no debe variar.

Hay más información sobre las instrucciones que se agregaron a la página web y la estructura de la página web en el Capítulo IV de este trabajo, titulado "*Pruebas y Resultados*".

3.12. Integración entre Jmol Builder y JME

Mientras se modificaba *Jmol* se descubrió que se podría mejorar aún más su funcionalidad implementando una página que estableciera una comunicación entre nuestra aplicación y el constructor y visualizador de moléculas en dos dimensiones llamado *JME*. Se trabajó junto con los investigadores del IVIC en la creación de esa página. Una captura de pantalla de la página se muestra en la Fig. 30.

El sistema original puede comunicarse con el *applet JME*. Lo hace a través de la carga y descarga de modelos representados con el contenido de archivos *MDL Molfile*. Se utiliza una operación de *JME* que captura la información de su modelo, y esa información es recibida e interpretada por *Jmol* para cargar el modelo. Sucede lo mismo con el pase de información de *Jmol* a *JME*.

Se tuvieron que hacer cambios en el sistema para reiniciar la barra de herramientas junto con el resto del sistema antes de cargar el modelo recibido de

JME, de la misma forma que *JME* lo hace antes de cargar el modelo recibido de *Jmol Builder*.

Figura 30: Página web que comunica *Jmol Builder* con *JME*

Se pudo establecer una comunicación entre la página web y *Jmol Builder* a través del método *evalStringQuiet* de la clase *BuilderViewer*, que se mencionó en el apartado "Diseño de *Jmol Builder*"; y se pudo reiniciar *Jmol Builder* desde la página web con la función *useJmolBuilder*, que se describió en el apartado "*Instrucciones sobre las páginas web que ejecuten Jmol Builder*", y un nuevo *script* que se incorporó en el sistema.

Una de las ventajas de nuestra página web es que se puede utilizar para crear un modelo con *JME* para continuar su construcción con *Jmol Builder* y viceversa (hay más información al respecto en el Capítulo IV de este trabajo, titulado "*Pruebas y Resultados*").

Los métodos creados y/o modificados en el sistema que permitieron realizar operaciones entre *Jmol Builder* y *JME* fueron los métodos *evalStringQuiet* de las clases *Jmol*, *JmolBuilder*, *JmolViewer* y *BuilderViewer*; y de la interfaz *WrappedApplet*.

En éste capítulo se mencionó que dentro del método *evalStringQuiet* de la clase *BuilderViewer* se implementó la operación de reinicio de *Jmol Builder*. El *script* que debe pasarse al método *evalStringQuiet* para ejecutarla es "*#JMOLBUILDER_RESTART_EDITOR*". La función *useJmolBuilder* utiliza la operación para regresar el sistema a su estado original antes de cargar el modelo extraído del *applet JME*.

La secuencia de llamadas ejecutadas desde la página web hasta la clase *BuilderViewer* fue posible por la extracción de una instancia de la clase *BuilderViewer* con la función *jmolGetPropertyAsJavaObject* desde el archivo *JmolBuilderWithJme.js*. La función *jmolGetPropertyAsJavaObject* ejecuta el método *getProperty* de la clase *JmolBuilderApplet*, que a su vez ejecuta el método *getProperty* de la clase *JmolBuilder*. Luego, ese método utiliza el atributo *viewer* de la clase *JmolBuilder* para ejecutar el método *getProperty* de la clase *BuilderViewer*. Una vez capturada la instancia de la clase *BuilderViewer*, la comunicación entre la página web y *BuilderViewer* es directa.

La función *jmolGetPropertyAsJavaObject* y el resto de las funciones que se utilizaron para pasar información de un *applet* a otro son funciones de la biblioteca *JavaScript* de *Jmol Builder*, que fueron tomadas de la biblioteca *JavaScript* del sistema original.

CAPÍTULO IV: PRUEBAS Y RESULTADOS

4.1. Constructor desarrollado

4.1.1. Barra de herramientas del applet

4.1.1.1. Descripción y funcionamiento de los botones

La barra de herramientas está compuesta por 19 botones: 3 grandes ("*Drag molecule mode*", "*Control set of atoms mode*" y "*Currently selected atom to add*") y 16 pequeños, como se muestra en la Fig. 31. A continuación se explica brevemente la funcionalidad de cada botón.

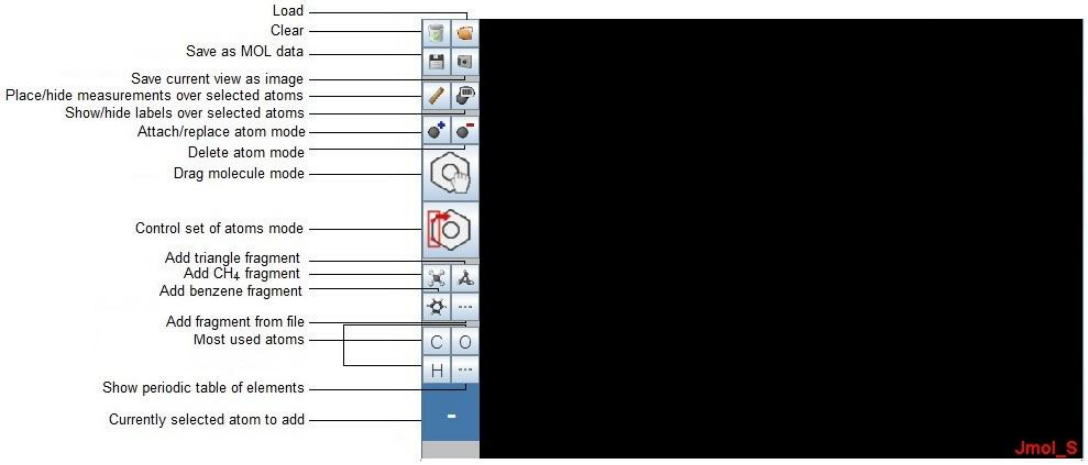


Figura 31: Botones de la barra de herramientas de *Jmol Builder*

- *Clear:* 

  Limpia la pantalla de trabajo, eliminando el modelo que se esté mostrando.

- *Load:* 

  Carga un modelo contenido en un archivo, reemplazando el modelo que se esté mostrando sobre la pantalla de trabajo. Puede ser un archivo *SPT* (*script*) *PDB* (*Protein Data Bank*) o un archivo mol *MDL* (*MDL Molfile*), creado por *MDL* (ahora conocido como *Symyx*). Por esa razón, puede cargar un archivo que haya sido grabado utilizando el botón que se explicará a continuación.

- *Save as MDL Molfile data:* 

  Guarda el modelo que se esté mostrando por pantalla como un archivo en formato *MDL Molfile*. Puede especificarse una extensión *".mol*", aunque es opcional.

- *Save current view as image:* 

  Guarda la vista actual del modelo en un archivo de imagen con el nombre que se desee, en la ubicación del equipo que se especifique, con uno de los siguientes formatos: *JPEG*, *GIF*, *PNG* o *PPM*.

- *Place/hide measurements over selected atoms:* 

  Activa y desactiva el modo de medición de distancias y ángulos entre los átomos. Una vez que el usuario lo activa, puede comenzar a medir presionando doble *click* sobre el primer átomo, y luego seleccionar hasta 3 átomos más con un *click* del ratón. Si se seleccionan 2 átomos, se calcula la distancia entre ellos. En cambio, si se seleccionan más átomos se calcula el ángulo entre esos átomos (entre 3 se calcula el ángulo plano y entre 4 el ángulo diedro). Para terminar la medición antes de seleccionar 4 átomos en total, se debe presionar doble *click* sobre el último, ya que el sistema termina la medición automáticamente al seleccionar el cuarto átomo. La Fig. 32 muestra los diferentes tipos de medición. Cuando se desactiva este modo, se eliminan las mediciones.

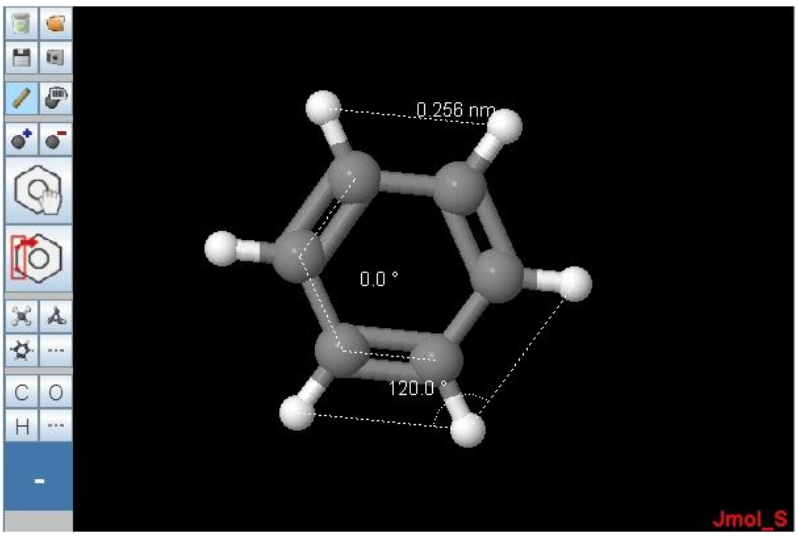


Figura 32: Mediciones entre 2, 3 y 4 átomos del modelo

- *Show/hide labels over selected atoms:* 

  Muestra u oculta etiquetas que indican el nombre de cada átomo seleccionado del modelo. *Jmol Builder* también permite mostrar u ocultar las etiquetas con la tecla F8.

- *Attach/replace atom mode:* 

  Activa y desactiva el modo de inserción o reemplazo de átomos, al que *Jmol* se refiere como "*Model Kit*" o kit para la construcción de un modelo. Una vez activado, el menú desplegable, que se puede acceder presionando *click* derecho sobre la pantalla de trabajo, permite seleccionar una serie de operaciones que facilitan la construcción del modelo, entre ellas, adjuntar un átomo al modelo o reemplazar un átomo del modelo por otro.

- *Delete atom mode:* 

  Activa y desactiva el modo de eliminación de átomos, en el que los átomos que se seleccionen son borrados del modelo uno por uno.

- *Drag molecule mode:* 

  Activa y desactiva el modo de arrastre de moléculas. Permite que la molécula que se seleccione pueda ser movida por la pantalla de trabajo sin modificar las coordenadas del resto de los átomos del modelo. Incluso, también se puede rotar la molécula si se mantiene presionada la tecla ALT. *Jmol Builder* permite activar y desactivar este modo con la tecla F4.

  Cuando se activa y desactiva el modo de arrastre de moléculas, sucede lo mismo con el modo de medición de distancias y ángulos entre los átomos.

- *Control set of atoms mode:* 

  Activa el modo de control de átomos seleccionados (pueden pertenecer a moléculas diferentes). Es un modo especial de *Jmol Builder* que muestra un panel de control de direcciones en la parte inferior derecha de la pantalla de trabajo (ver Fig. 33). Cuando el modo está activo, se puede seleccionar un grupo de átomos, ya sea presionando *click* sobre cada uno o presionando la tecla SHIFT mientras se arrastra el ratón con el botón izquierdo presionado. Esta última forma de selección hace posible dibujar un rectángulo sobre la pantalla de trabajo que indica los átomos que se desean seleccionar (ver Fig. 34). Sólo el grupo de átomos seleccionados se puede rotar o trasladar.

Este modo se puede utilizar junto con el de medición de distancias y ángulos entre los átomos. Las mediciones se mantienen sobre el modelo aún después de activarlo. No se pueden crear nuevas mediciones pero las ya desplegadas cambian a medida que se mueven los átomos. Esta característica permite a los usuarios tener un mayor control sobre la construcción del modelo y es de gran relevancia para químicos computacionales para investigar procesos químicos de nuevas moléculas en relación con otras moléculas creadas por el hombre y las que se encuentran en la naturaleza.

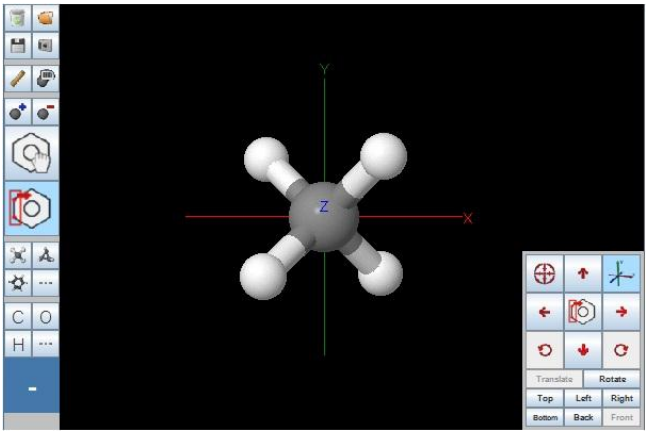


Figura 33: Estado inicial del modo de control de átomos seleccionados

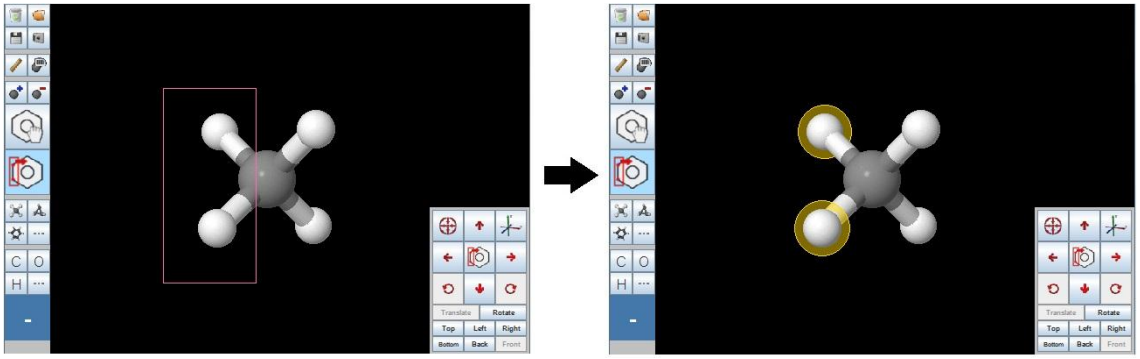


Figura 34: Selección de átomos arrastrando el ratón

(mientras se presiona el botón izquierdo y la tecla SHIFT)

El panel está compuesto por 17 botones: 9 botones grandes y 8 botones medianos, como se muestra en la Fig. 35.

- *Translation/rotation north-wise:* 

  Traslada o rota el grupo de átomos seleccionados en dirección norte o hacia arriba, mientras el modo de control de átomos seleccionados esté activo.

- *Translation/rotation west-wise:* 

  Traslada o rota el grupo de átomos seleccionados en dirección oeste o hacia la izquierda, mientras el modo de control de átomos seleccionados esté activo.

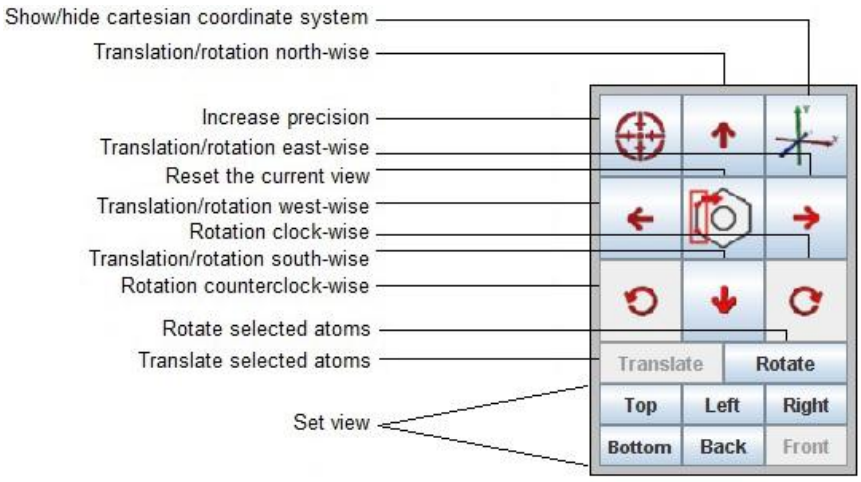


Figura 35: Botones del panel de control de direcciones de *Jmol Builder*

- *Translation/rotation east-wise:* 

  Traslada o rota el grupo de átomos seleccionados en dirección este o hacia la derecha, mientras el modo de control de átomos seleccionados esté activo.

- *Translation/rotation south-wise:* 

  Traslada o rota el grupo de átomos seleccionados en dirección sur o hacia abajo, mientras el modo de control de átomos seleccionados esté activo.

- *Rotation clock-wise:* 

  Rota el grupo de átomos seleccionados en sentido horario, mientras el modo de control de átomos seleccionados esté activo.

- *Rotation counterclock-wise:* 

  Rota el grupo de átomos seleccionados en sentido antihorario, mientras el modo de control de átomos seleccionados esté activo.

- *Show/hide cartesian coordinate system:* 

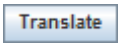
  Muestra u oculta el sistema de coordenadas cartesianas.

- *Increase precisión:* 

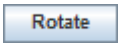
  Aumenta la precisión de la traslación y la rotación.

- *Reset the current view:* 

  Posiciona la vista de acuerdo a la vista seleccionada en el panel.

- *Translate selected atoms:* 

  Le indica a *Jmol* que traslade el grupo de átomos seleccionados en la dirección especificada.

- *Rotate selected atoms:* 

  Le indica a *Jmol* que rote el grupo de átomos seleccionados en la dirección especificada.

- *Set view:* 

  Posiciona la vista, ya sea de frente (*front*), atrás (*back*), a la izquierda (*left*), a la derecha (*right*), arriba (*top*) o debajo (*bottom*) del modelo.

- *Add CH4 fragment:* 

Agrega un CH_4 cerca del modelo que se esté mostrando en pantalla.

- *Add triangle fragment:* 

  Agrega un anillo de 3 átomos de carbono cerca del modelo que se esté mostrando en pantalla.

- *Add benzene fragment:* 

  Agrega un benceno cerca del modelo que se esté mostrando en pantalla.

- *Add fragment from file:* 

  Agrega un fragmento, desde un archivo ubicado en la computadora del usuario, cerca del modelo que se esté mostrando en pantalla.

- *Most used atoms:* 

  Consiste de 4 botones que muestran los átomos más utilizados por el usuario para ser unidos al modelo. Una vez seleccionado un botón correspondiente a uno de esos átomos, puede ser unido al modelo presionando *click* sobre alguno de los átomos del modelo y luego arrastrando el ratón hacia fuera, o también puede reemplazar un átomo del modelo sólo presionado *click* sobre el que se quiera sustituir.

- *Show periodic table of elements:* 

  Muestra una ventana especial de *Jmol Builder*, que contiene una tabla periódica de elementos (ver Fig. 36). En esta ventana el usuario puede seleccionar cualquier átomo de la tabla periódica para ser unido al modelo y aceptar o cancelar la selección con botones que ocultan la ventana. Si el usuario acepta, el átomo que se seleccionó puede ser unido al modelo o reemplazar un átomo del modelo, como se explicó al describir la funcionalidad del botón "*Most used atoms*".

- *Currently selected atom to add:* 

  Muestra el símbolo del átomo que podrá ser unido al modelo mientras el modo de inserción o reemplazo de átomos esté activo. También muestra el átomo seleccionado al utilizar los botones "*Most used atoms*" o el botón "*Show periodic table of elements*", del que tiene la misma funcionalidad.

4.1.1.2. Otros modos relacionados con la barra

Existen algunos modos especiales que no tienen asignado un botón en la barra de herramientas. Son los siguientes:

- *Modo de visualización:*

  Es el modo que se activa al iniciar tanto *Jmol* como *Jmol Builder*. Permite utilizar un menú desplegable que muestra una serie de operaciones que facilitan la visualización y análisis del modelo ya construido, como la medición de las distancias y los ángulos entre los átomos del modelo. Es el único modo que se activa cuando no hay ningún botón de la barra de herramientas presionado.

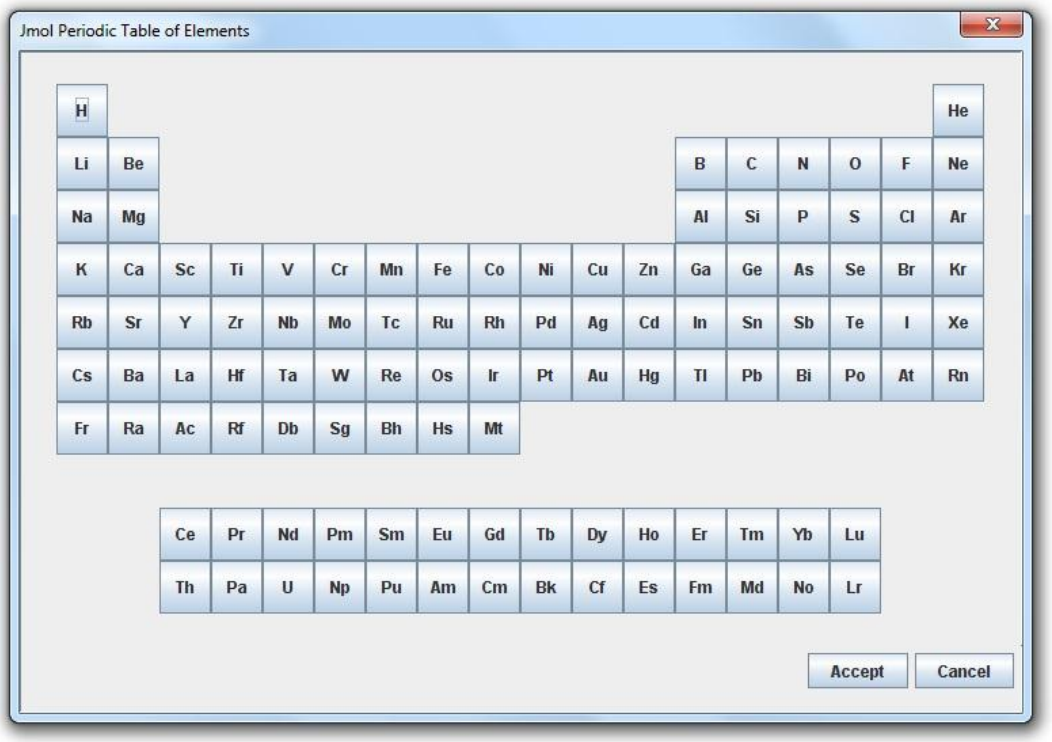


Figura 36: Ventana de tabla periódica de elementos

- *Modo de centrado de cámara sobre un átomo:*

  Es un modo que centra la pantalla de trabajo y establece el eje de rotación sobre el átomo que se seleccione con un *click* del ratón. Se activa y se desactiva con la tecla F12. Ya que no hay botón asociado a este modo y el modo de visualización se activa cuando ningún botón de la barra de herramientas está activado, se le indica al usuario que este modo se ha

activado cambiando el color predeterminado del fondo del panel de la barra de herramientas a azul.

4.1.2. Manejo de eventos del teclado

Jmol Builder permite utilizar el teclado para acceder más rápido a ciertas *operaciones de Jmol y Jmol Builder*. Las teclas que pueden acceder a esas operaciones son:

- *F3:*

  Muestra u oculta la barra de herramientas.

- *F4:*

  Activa y desactiva el modo de arrastre de moléculas y el modo de medición de distancias y ángulos entre los átomos.

- *F5:*

  Hace una optimización preliminar del modelo.

- *F6:*

  Recarga el modelo. Si está compuesto por distintos modelos secundarios, los agrupa en uno solo.

- *F7:*

  Posiciona el modelo en el centro de la pantalla de trabajo.

- *F8:*

  Muestra u oculta etiquetas que indican el nombre de cada átomo seleccionado del modelo.

- *F9:*

  Regresa el modelo a su posición de inicio.

- *F10:*

  Selecciona o deselecciona todos los átomos del modelo.

- *F11:*

  Muestra u oculta los átomos que no estén seleccionados.

- *F12:*

  Activa y desactiva el modo de centrado de cámara sobre un átomo, y el modo de medición de distancias y ángulos entre los átomos.

- *U:*

  Deshace cambios realizados al modelo, aunque no toma en cuenta los cambios realizados a la barra de herramientas.

- *R:*

  Vuelve a hacer cambios realizados al modelo previamente deshechos, aunque no toma en cuenta los cambios realizados a la barra de herramientas.

4.2. Pruebas de Compatibilidad

Jmol es compatible con: *Internet Explorer*, *Mozilla* o *Firefox*, *Safari*, *Opera*, *Konkeror* e *IceWeasel*. Como es multiplataforma, se ejecuta en *Windows*, *Mac OS X* y *Linux* o *Unix* [31].

Jmol Builder fue probado sobre: *Firefox* 4.0 hasta 10.0, *Google Chrome* 16.0.912.77, e *Internet Explorer* 7 en adelante. También se probó en diferentes sistemas operativos, aunque su funcionamiento no depende de ello. *Jmol Builder* funciona en sistemas: *Windows XP*, *Windows* 7, *Mac OS X* 10.7 en adelante y *Ubuntu* (*Linux*) 11.04 en adelante. El factor determinante para su funcionamiento es la versión de *Java*, que debe ser desde la 1.6.0__24 en adelante.

4.3. Pruebas de Usuario

La utilidad de este trabajo de investigación se midió mediante el uso de escalas de Likert, escalas psicométricas involucradas en la investigación que emplean cuestionarios. La escala consiste en una serie de ítems que se responden con base en una lista de grados simétricos de acuerdo o desacuerdo que representan la veracidad de una declaración según la persona que la conteste. De esa forma se captura el nivel de agrado hacia un cierto ítem. Una vez analizadas las respuestas de varias personas a varios ítems, se puede determinar el agrado hacia un trabajo de investigación. Se llama escala de Likert por Rensis Likert, quien publicó en 1932 un informe donde describía su uso [35].

En nuestro caso, las encuestas se enfocaron en la utilidad de cuatro componentes:

1. La barra de herramientas.
2. El panel de control de direcciones.
3. Las funcionalidades accedidas utilizando el teclado.
4. La página web que contiene a *Jmol Builder*.

Para cada componente se copiaron varias declaraciones, sobre sus funcionalidades y características, junto con una escala de acuerdo o desacuerdo de 5 valores del 1 al 5. Mientras mayor era el valor, mayor era la utilidad de ese ítem. El número 1 representaba la respuesta "*no me parece útil*" y el 5 "*me parece muy útil*".

Las encuestas fueron respondidas por 4 investigadores y 2 pasantes del laboratorio de Química Computacional del Instituto Venezolano de Investigaciones Científicas, entre ellos un investigador titular y un investigador asociado. El diseño de la encuesta incluía básicamente la información que se muestra en las Tablas 2, 3, 4 y 5.

Se tomaron las encuestas respondidas por cada investigador y se unieron en una sola tabla por componente. La Tabla 6 muestra los resultados sobre la utilidad del primer componente, la barra de herramientas. Luego se calculó el promedio de la utilidad de cada funcionalidad de cada componente. Esos promedios se calcularon con la siguiente fórmula: $P_i = (1*N_{i1} + 2*N_{i2} + 3*N_{i3} + 4*N_{i4} + 5*N_{i5}) / M_i$, donde i es la funcionalidad i-ésima, N_{ij} es el número de entrevistados que calificaron el componente con un valor de j y M_i es el puntaje máximo, que es la suma de tantos 5 como la cantidad de entrevistados que evaluaron la i-ésima funcionalidad. Estos valores P_i están en el rango [0,1]. Luego los P_i se escalan al rango [1,5] para hacerlos corresponder con los 5 valores de acuerdo o desacuerdo posibles, mediante la siguiente fórmula: $V_i = (P_i * (5 – 1)) + 1)$, donde i es la funcionalidad i-ésima, y P_i es el promedio de utilidad. La Fig. 37 presenta gráficamente los valores de V_i de las funcionalidades de la barra de herramientas. Se repitió el mismo procedimiento con los 3 componentes restantes.

No a todas las funcionalidades les fue asignado un valor de 5. Se pidió a los investigadores una justificación por cada valor asignado a cada funcionalidad de cada componente para obtener una explicación del por qué esas funcionalidades no fueron consideradas muy útiles. Sin embargo, algunos investigadores no justificaron sus respuestas. Se tomaron todas las justificaciones para interpretar las gráficas que se generaron.

Los resultados de las encuestas fueron bastante satisfactorios. Todos los investigadores consideraron que la barra de herramientas, con botones que

permiten el acceso a varias operaciones comunes de *Jmol*, además de las nuevas funcionalidades de *Jmol Builder*, es altamente útil.

La Tabla 7 muestra los resultados obtenidos sobre la utilidad del panel de control de direcciones. La Fig. 38 muestra estos resultados de forma gráfica.

Se obtuvieron resultados muy parecidos a la barra de herramientas, aunque no les pareció muy útil el hecho que el control de direcciones tiene una posición fija en la pantalla de trabajo y no se puede ocultar aunque se presione *click* sobre esa pantalla.

La Tabla 8 muestra los resultados obtenidos sobre la utilidad de las funcionalidades asignadas al teclado. La Fig. 39 muestras estos resultados de forma gráfica.

¿Qué tan útil le parece la barra de herramientas?

	No me parece útil 1	- 2	- 3	- 4	Me parece muy útil 5
La funcionalidad que elimina el modelo de la pantalla de trabajo					
La funcionalidad que carga un modelo desde un archivo					
La funcionalidad que guarda el modelo como un archivo MDL Molfile					
La funcionalidad que coloca mediciones sobre los átomos					
La funcionalidad que muestra etiquetas sobre los átomos					
La funcionalidad que activa el modo que agrega o reemplaza átomos del modelo					
La funcionalidad que activa el modo que elimina átomos del modelo					
La funcionalidad que activa el modo que arrastra moléculas del modelo					
La funcionalidad que activa el modo que controla átomos seleccionados del modelo					
La funcionalidad que agrega fragmentos predeterminados al modelo					
La funcionalidad que agrega un fragmento al modelo desde un archivo					
La funcionalidad que selecciona los átomos C, H y O para ser agregados al modelo					
La funcionalidad que muestra la tabla periódica interactiva					

Tabla 2: Sección de la encuesta sobre la barra de herramientas de *Jmol Builder*

¿Qué tan útil le parece el panel de control de direcciones?

	No me parece útil 1	- 2	- 3	- 4	Me parece muy útil 5
La funcionalidad que traslada o rota los átomos seleccionados en una dirección recta					
La funcionalidad que rota los átomos seleccionados en sentido horario					
La funcionalidad que rota los átomos seleccionados en sentido anti horario					
La funcionalidad que muestra u oculta el eje de coordenadas cartesianas					
La funcionalidad que aumenta la precisión de la traslación o la rotación					
La funcionalidad que posiciona la vista de acuerdo a la vista seleccionada en el panel					
La funcionalidad que le indica a Jmol que traslade el grupo de átomos seleccionados					
La funcionalidad que le indica a Jmol que rote el grupo de átomos seleccionados					
La funcionalidad que cambia la posición de la vista sobre el modelo					
El control de direcciones es fijo					
El control de direcciones no se oculta cuando se presiona clic sobre la pantalla de trabajo					

Tabla 3: Sección de la encuesta sobre el panel de control de direcciones de *Jmol Builder*

¿Qué tan útil le parecen las funcionalidades accedidas utilizando el teclado?

	No me parece útil 1	- 2	- 3	- 4	Me parece muy útil 5
La tecla que muestra u oculta la barra de herramientas, F3					
La tecla que activa y desactiva el modo de arrastre de moléculas, F4					
La tecla que hace una optimización preliminar del modelo, F5					
La tecla que recarga el modelo, F6					
La tecla que posiciona el modelo en el centro de la pantalla de trabajo, F7					
La tecla que muestra u oculta etiquetas sobre los átomos del modelo, F8					
La tecla que regresa el modelo a su posición de inicio, F9					
La tecla que selecciona o deselecciona todos los átomos del modelo, F10					
La tecla que muestra u oculta los átomos que no estén seleccionados, F11					
La tecla que activa el modo de centrado de cámara sobre un átomo, F12					
La tecla que deshace cambios realizados al modelo, U					
La tecla que vuelve a hacer cambios realizados al modelo previamente deshechos , R					

Tabla 4: Sección de la encuesta sobre las funcionalidades accedidas utilizando el teclado con *Jmol Builder*

¿Qué tan útil le parece la página web que contiene a Jmol Builder?

	No me parece útil 1	- 2	- 3	- 4	Me parece muy útil 5
Los compuestos creados/modificados con JME se pueden transferir a Jmol Builder					
Los compuestos creados/modificados con Jmol Builder se pueden transferir a JME					
Es posible cambiar la resolución de la página web					
Las ayudas sobre las funciones del teclado y el acceso al manual de usuario					

Tabla 5: Sección de la encuesta sobre la página *web* que contiene a *Jmol Builder*

¿Qué tan útil le parece la barra de herramientas?

	1	2	3	4	5
La funcionalidad que elimina el modelo de la pantalla de trabajo	0	0	0	3	3
La funcionalidad que carga un modelo desde un archivo	0	0	1	0	5
La funcionalidad que guarda el modelo como un archivo MDL Molfile	0	0	1	1	4
La funcionalidad que coloca mediciones sobre los átomos	0	0	0	2	4
La funcionalidad que muestra etiquetas sobre los átomos	0	0	1	2	3
La funcionalidad que activa el modo que agrega o reemplaza átomos del modelo	0	0	1	0	5
La funcionalidad que activa el modo que elimina átomos del modelo	0	0	0	1	5
La funcionalidad que activa el modo que arrastra moléculas del modelo	0	0	0	1	5
La funcionalidad que activa el modo que controla átomos seleccionados del modelo	0	0	0	1	5
La funcionalidad que agrega fragmentos predeterminados al modelo	0	0	1	0	5
La funcionalidad que agrega un fragmento al modelo desde un archivo	0	0	0	0	6
La funcionalidad que selecciona los átomos C, H y O para ser agregados al modelo	0	0	1	0	5
La funcionalidad que muestra la tabla periódica interactiva	0	0	1	2	3

Tabla 6: Respuestas sobre la utilidad de la barra de herramientas. Para cada nivel de utilidad (de 1 a 5, resaltado en azul) se muestra el número de encuestados que le asignaron ese nivel de utilidad a cada funcionalidad

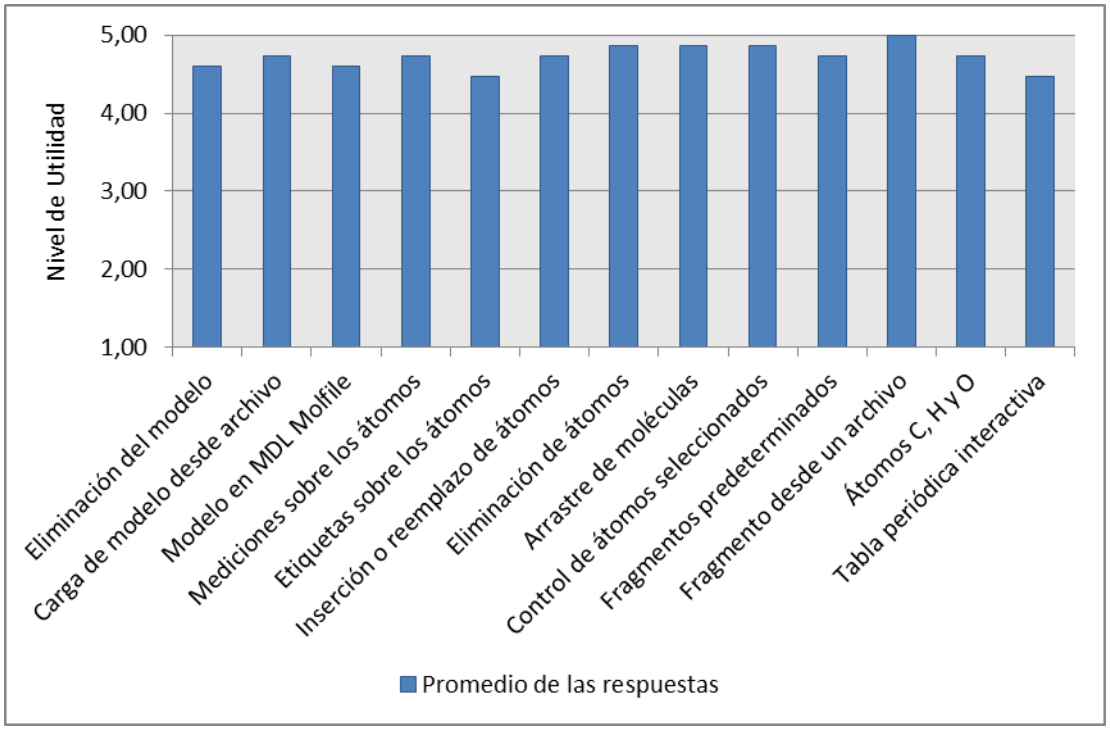


Figura 37: Gráfico de barras del promedio de las respuestas relacionadas con la barra de herramientas

¿Qué tan útil le parece el panel de control de direcciones?

	1	2	3	4	5
La funcionalidad que traslada o rota los átomos seleccionados en una dirección recta	0	0	0	1	5
La funcionalidad que rota los átomos seleccionados en sentido horario	0	0	0	2	4
La funcionalidad que rota los átomos seleccionados en sentido antihorario	0	0	0	1	5
La funcionalidad que muestra u oculta el eje de coordenadas cartesianas	0	0	1	1	4
La funcionalidad que aumenta la precisión de la traslación o la rotación	0	0	0	1	5
La funcionalidad que posiciona la vista de acuerdo a la vista seleccionada en el panel	0	0	1	2	3
La funcionalidad que le indica a Jmol que traslade el grupo de átomos seleccionados	0	0	0	1	5
La funcionalidad que le indica a Jmol que rote el grupo de átomos seleccionados	0	0	0	1	5
La funcionalidad que cambia la posición de la vista sobre el modelo	0	0	0	1	5
El control de direcciones es fijo	0	1	1	2	2
El control de direcciones no se oculta cuando se presiona click sobre la pantalla de trabajo	0	0	2	2	2

Tabla 7: Respuestas sobre la utilidad del panel de control de direcciones. Para cada nivel de utilidad (de 1 a 5, resaltado en azul) se muestra el número de encuestados que le asignaron ese nivel de utilidad a cada funcionalidad

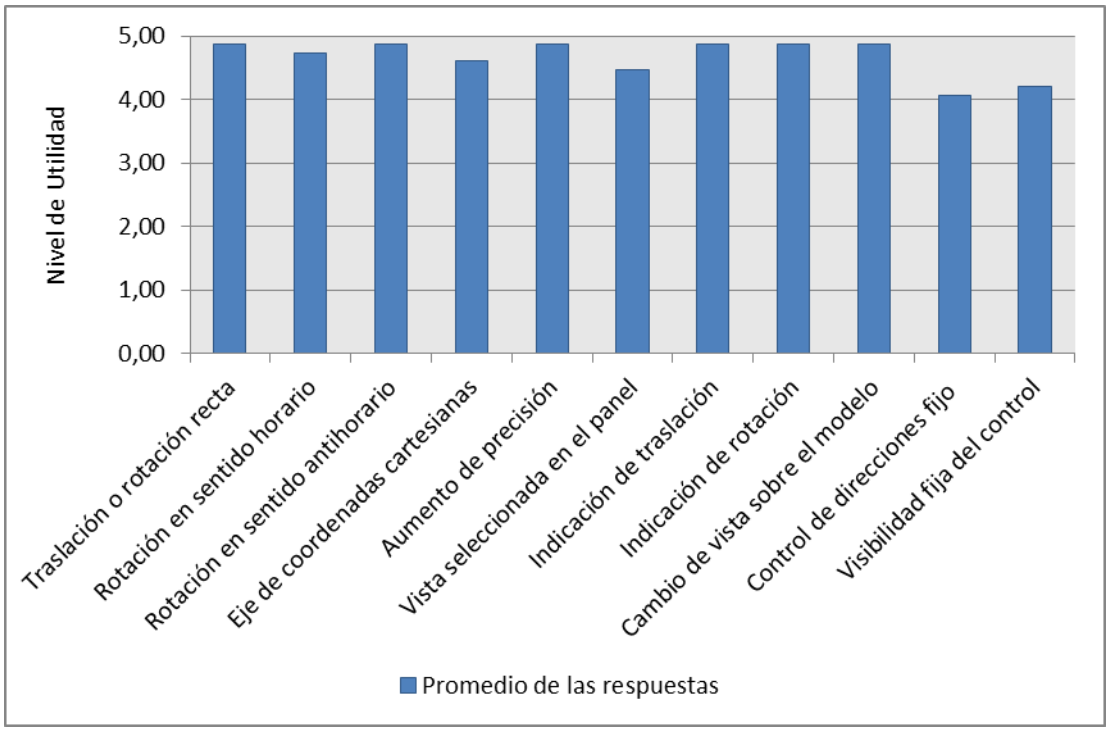


Figura 38: Gráfico de barras del promedio de las respuestas relacionadas con el panel de control de direcciones

¿Qué tan útil le parecen las funcionalidades accedidas utilizando el teclado?

	1	2	3	4	5
La tecla que muestra u oculta la barra de herramientas, F3	0	1	3	1	1
La tecla que activa y desactiva el modo de arrastre de moléculas, F4	0	0	1	3	2
La tecla que hace una optimización preliminar del modelo, F5	0	0	3	1	2
La tecla que recarga el modelo, F6	0	1	1	1	3
La tecla que posiciona el modelo en el centro de la pantalla de trabajo, F7	0	1	1	1	3
La tecla que muestra u oculta etiquetas sobre los átomos del modelo, F8	0	0	2	0	4
La tecla que regresa el modelo a su posición de inicio, F9	0	1	1	0	4
La tecla que selecciona o deselecciona todos los átomos del modelo, F10	0	0	2	1	3
La tecla que muestra u oculta los átomos que no estén seleccionados, F11	0	1	1	1	3
La tecla que activa el modo de centrado de cámara sobre un átomo, F12	0	0	0	2	4
La tecla que deshace cambios realizados al modelo, U	0	0	0	1	5
La tecla que vuelve a hacer cambios realizados al modelo previamente deshechos , R	0	0	0	1	5

Tabla 8: Respuestas sobre la utilidad de las funcionalidades usando el teclado. Para cada nivel de utilidad (de 1 a 5, resaltado en azul) se muestra el número de encuestados que le asignaron ese nivel de utilidad a cada funcionalidad

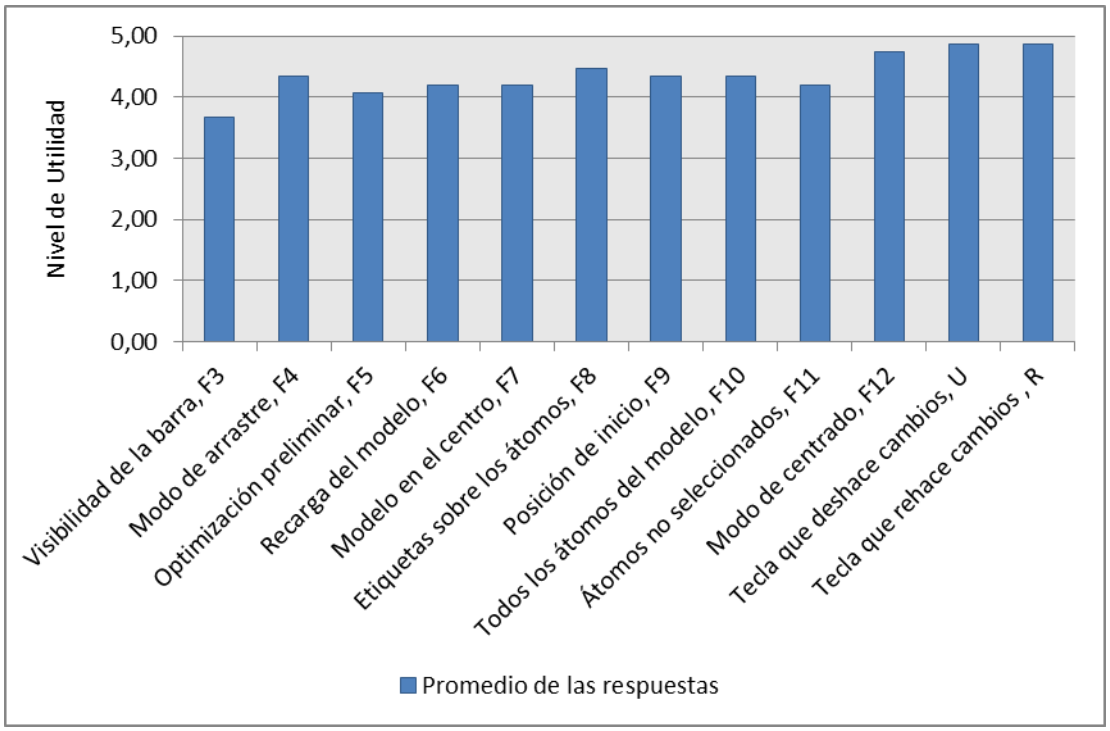


Figura 39: Gráfico de barras del promedio de las respuestas relacionadas con las funcionalidades usando el teclado

Los investigadores no le vieron una gran utilidad a las funcionalidades asignadas al teclado en comparación con la barra de herramientas y el panel de control de direcciones. Las funcionalidades que menos llamaron la atención fueron la posibilidad de ocultar la barra de herramientas y la posibilidad de realizar una optimización preliminar del modelo. Aun así, el nivel de satisfacción fue bastante alto. Las funcionalidades más útiles según los investigadores fueron: la tecla que activa el modo de centrado de cámara sobre un átomo y las que deshacen o rehacen algunos cambios al modelo.

La Tabla 9 muestra los resultados obtenidos sobre la utilidad de la página web que contiene a *Jmol Builder*. La Fig. 40 muestra estos resultados de forma gráfica.

¿Qué tan útil le parece la página web que contiene a Jmol Builder?

	1	2	3	4	5
Los compuestos creados/modificados con JME se pueden transferir a Jmol Builder	0	0	0	1	5
Los compuestos creados/modificados con Jmol Builder se pueden transferir a JME	0	0	0	1	5
Es posible cambiar la resolución de la página web	0	1	1	3	1
Las ayudas sobre las funciones del teclado y el acceso al manual de usuario	0	0	1	3	2

Tabla 9: Respuestas sobre la utilidad de la página web que contiene a *Jmol Builder.* Para cada nivel de utilidad (de 1 a 5, resaltado en azul) se muestra el número de encuestados que le asignaron ese nivel de utilidad a cada funcionalidad

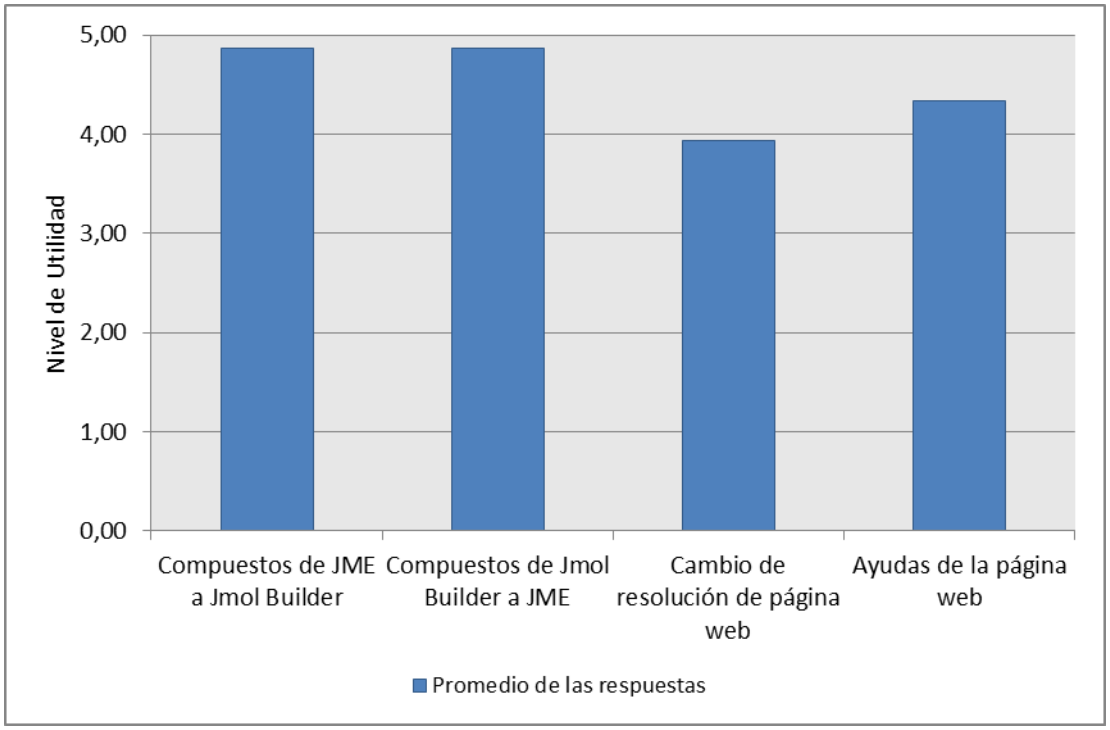


Figura 40: Gráfico de barras del promedio de las respuestas relacionadas con la página web que contiene a *Jmol Builder*

Se obtuvieron resultados muy parecidos a los obtenidos para las funcionalidades asignadas al teclado. La transferencia de los modelos de un *applet* a otro fue considerada altamente útil. Sin embargo, el cambio de resolución y las ayudas de la página web no lo fueron, porque el cambio de resolución no es utilizado con frecuencia y es costumbre en el Instituto buscar ayuda directamente en un manual de usuario.

Con los datos obtenidos de las encuestas también se calculó el porcentaje de utilidad de los nuevos componentes. Este porcentaje se calculó con la siguiente fórmula: $C_i = A_i / M_i$, donde *i* es el componente *i*-ésimo, A_i es la suma de todos los valores dados por todos los entrevistados sobre el componente *i* y M_i es el puntaje máximo, que es la suma de tantos 5 como ítems pertenecen al *i*-ésimo componente. Nótese que C_i es un indicativo de que tan cerca estuvo el componente *i* de obtener un puntaje perfecto de "*me parece muy útil*" en todas sus funcionalidades. La Fig. 41 muestra de forma gráfica los porcentajes obtenidos.

El componente de *Jmol Builder* que fue considerado más útil fue la barra de herramientas porque, entre otras razones, facilita el aprendizaje del sistema ofreciendo una serie de botones de acceso rápido en lugar de un menú emergente para el acceso a todas las operaciones de *Jmol*, permite visualizar rápidamente las herramientas disponibles y puede ser utilizado para construir modelos sofisticados agregando modelos construidos previamente. Otro componente que

fue calificado como altamente útil fue el panel de control de direcciones; porque permite posicionar los átomos de una molécula según sea requerido por un cálculo en particular. El componente relacionado con el teclado fue considerado como el menos útil, porque algunas de las funcionalidades que son accedidas por el teclado no son utilizadas con frecuencia, los investigadores prefirieron utilizar las funcionalidades a través de la barra de herramientas y no a través del teclado y algunas funcionalidades fueron consideradas tan útiles que se sugirió incluir un acceso a ellas desde la barra de herramientas en lugar de accederlas a través del teclado. Finalmente, aunque la página web en general no tuvo un porcentaje de utilidad tan alto como se esperaba, la integración entre *Jmol Builder* y *JME* sí lo tuvo.

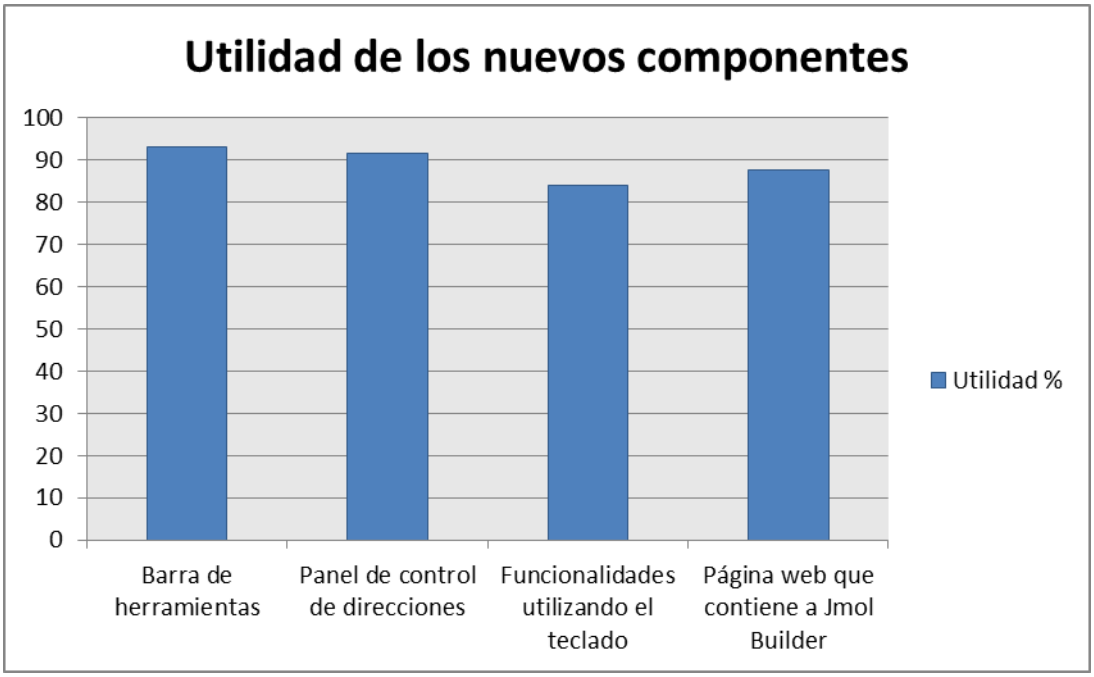


Figura 41: Gráfico de barra de los porcentajes de utilidad de los componentes de *Jmol Builder*

Por último, se promediaron los porcentajes de utilidad de todos los componentes con la siguiente fórmula: $U = (P_1 + P_2 + P_3 + P_4)/ T$, donde i es el componente i-ésimo, P_i es el porcentaje de utilidad del componente i y T es el número total de componentes, que es 4.

Con los resultados obtenidos gracias a las escalas de Likert fue posible detectar que el trabajo realizado tiene un alto porcentaje de utilidad de 89,33%. Se puede concluir que *Jmol Builder* es considerado muy útil por los investigadores del Centro de Química Computacional del IVIC.

Después de que la utilidad de *Jmol Builder* fue evaluada con las encuestas, se mostró el sistema a la comunidad de *Jmol* enviándoles un mensaje con el enlace a

la página web, a su correo electrónico, *jmol-developers-request@lists.sourceforge.net*, con el asunto "*Jmol with a toolbar*". Los siguientes fueron algunos de los mensajes que recibimos de vuelta:

- El Dr. Angel Herráez, del Departamento de Bioquímica y Biología Molecular de la Universidad de Alcalá, de España, escribió: "*That's an interesting extensión… The black box on the left is somewhat confusing. I was waiting for something to appear there. Maybe you can put a note like 'this is reserved for JME'*".

- Otro miembro de la comunidad llamado Robert M. Hanson, profesor de química de la Universidad *St. Olaf*, de Estados Unidos, escribió: "*Interesting! I notice you are using 12.1 for that -- a much better plan would be to make it sit on top of Jmol so that no matter what Jmol version there is, you are can adapt as necessary. Then as Jmol develops you can just plug it into your effort. I suggest a button to open one of the unlimited number of IUPAC-named compounds accessible. I think the days of building molecules from scratch is about over. If you can name it -- or almost name it -- you can probably get it (if it is organic) now without any work*".

- El Dr. Jonathan H. Gutow del Departamento de Química de la Universidad de *Wisconsin*, de Estados Unidos, escribió: "*I think this is pretty cool and think we should pursue replacing the little model builder menu system with this*".

Como se puede observar, se recibió como respuesta el apoyo de varios de los miembros de la comunidad junto con algunas sugerencias.

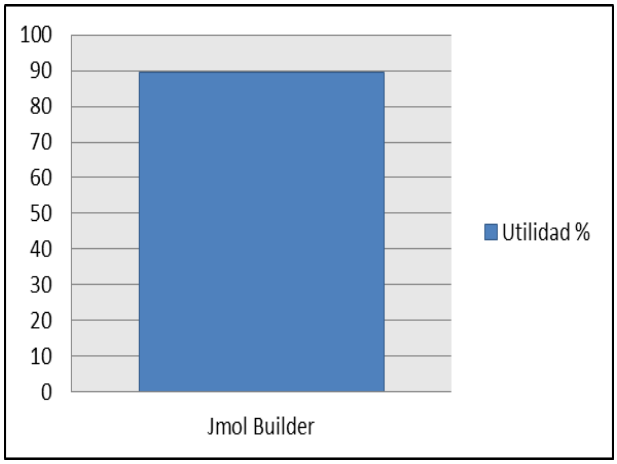


Figura 42: Porcentaje de utilidad de todos los componentes

Como resultado adicional se tiene que *Jmol Builder* actualmente se encuentra integrado al sistema *IVIChem* del Instituto Venezolano de Investigaciones

Científicas, que es un conjunto de interfaces web para química computacional (ver Fig. 43).

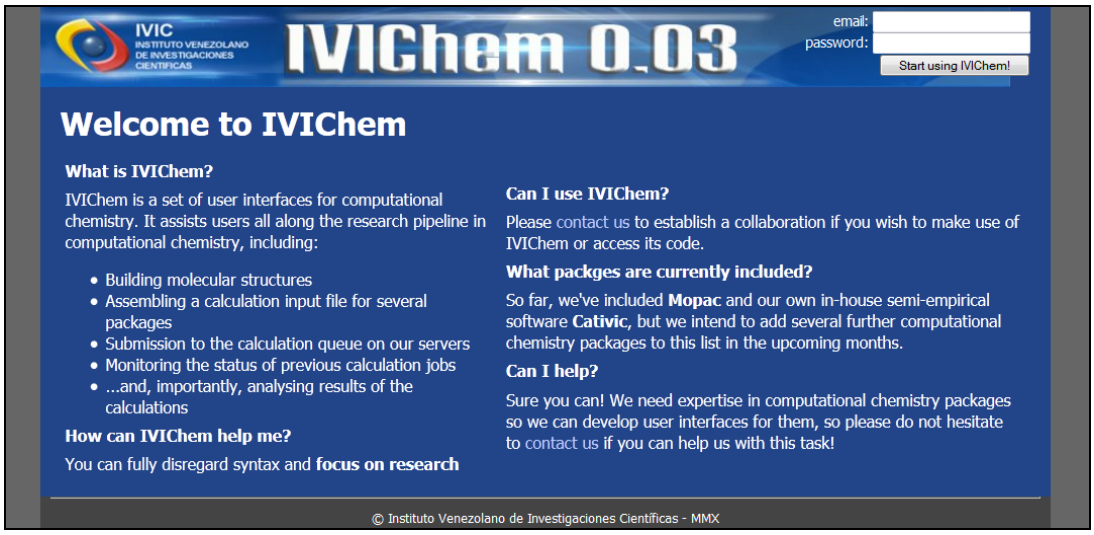


Figura 43: Página web del sistema *IVIChem*

4.4. Cambios después de la evaluación del software

Jmol Builder fue modificado después de la entrega de las encuestas. Los investigadores del IVIC sugirieron cambiar el nombre del botón "*Translate selected atoms*" por "*Move selected atoms*" y cambiar la imagen del botón "*Show periodic table of elements*" por la de una tabla periódica. La versión del sistema que se demostrará en la presentación de este trabajo no tendrá estos cambios.

CAPÍTULO V: CONCLUSIONES Y TRABAJOS FUTUROS

5.1. Conclusiones

Durante este trabajo se estudiaron una gran cantidad de constructores y visualizadores moleculares, así como la posibilidad de extenderlos para proveer un nuevo constructor molecular para el Laboratorio de Química Computacional del IVIC, que es un instituto dedicado a resolver problemas relacionados con su correspondiente rama de la Química. Se seleccionó *Jmol* para su extensión, y esto se logro satisfactoriamente, produciendo *Jmol Builder*. La nueva extensión se ajusta a las necesidades del grupo de investigación del IVIC. Se realizaron pruebas satisfactorias tanto local como remotamente, en múltiples plataformas y navegadores.

Jmol Builder ofrece una interfaz mejorada considerablemente en comparación con la interfaz de *Jmol*, ya que ofrece una barra de herramientas y un control de direcciones con imágenes iconográficas, que hacen la interacción con *Jmol* más natural y más rápida. Los iconos fueron construidos gracias a las directrices de los investigadores del IVIC, que son los usuarios principales del *Jmol Builder*.

El tiempo de dedicación que los integrantes de la comunidad de Química Computacional del IVIC dedicaron al desarrollo de *Jmol Builder* fue bastante limitado, debido a su gran número de ocupaciones, lo cuál implico un gran número de reuniones para el desarrollo de *Jmol Builder*. De acuerdo a los resultados de las encuestas y a las opiniones personales de los investigadores del IVIC, estos encuentran satisfactoria la nueva extensión y les parece que *Jmol Builder* resuelve el problema planteado en este TEG.

Se demostró que *Jmol Builder* mejora las funcionalidades de *Jmol*. De acuerdo a la comunidad de usuarios de *Jmol* en la web, las funcionalidades de *Jmol Builder* podrían ser consideradas para agregarse a *Jmol* en sus futuras versiones. Igualmente, en este trabajo se mejoro la aplicación aún más, integrando *Jmol Builder* con un constructor 2D bien conocido, como lo es JME. Esta integración también fue evaluada por los investigadores del IVIC, y les pareció bastante útil.

Jmol es un sistema compuesto de cientos de clases. El desarrollador que desee trabajar con él debe tomar en cuenta que, aunque el sistema se puede extender, se requiere bastante tiempo para reconocer la sección que se necesita modificar. Sin embargo, una vez extendido, el sistema resultante es soportado por los navegadores más comunes, ya que se basa en un applet estándar desarrollado en Java. En el caso de *Jmol Builder*, se logró la extensión conectándose solo con algunas pocas clases de la jerarquía de clases de *Jmol*, aunque se requirio bastante tiempo reconocer a cuáles clases se debía conectar la extensión.

La nueva jerarquía de clases agregada a *Jmol* para convertirse en *Jmol Builder* es separable del sistema *Jmol* original. Esto le da la posibilidad al usuario de ejecutar cualquiera de las dos versiones independientemente. Todas las funcionalidades de *Jmol Builder* se implementaron con base en esa premisa.

5.2. Trabajos futuros

Entre las modificaciones sugeridas por el IVIC, las siguientes se consideran como las más importantes:

- Sincronizar las operaciones de deshacer y rehacer cambios con los cambios realizados a través de la barra de herramientas y el control de direcciones.
- Añadir un *script* que agregue un fragmento de una lista de modelos desde la página Web, utilizando un comando de *Jmol* modificado, llamado con una función *JavaScript*.
- Implementar en una página web las encuestas que se utilizaron en el Capítulo lV de este trabajo, titulado "*Pruebas y Resultados*", para poder calcular el porcentaje de utilidad de *Jmol Builder* según la comunidad de *Jmol*.
- Implementar una funcionalidad para eliminar grupos de átomos seleccionados.

Las sugerencias más importantes de la comunidad de *Jmol* fueron:

- Convertir a *Jmol Builder* en un *plugin* que pueda ser instalado sobre cualquier versión de *Jmol* de la 12.1.50 en adelante.
- Agregar un botón que cargue cualquiera del sin fin de compuestos nombrados por *IUPAC*. De esa forma, si el modelo se puede nombrar, incluso parcialmente, se puede construir sin mucho trabajo.

GLOSARIO

1) **Alkyl (alquilo):** es una cadena lateral generalmente abreviada con el símbolo R compuesta por átomos de carbono e hidrógeno unidos solamente por enlaces simples [36].

2) **AMBER:** es un conjunto de campos de fuerza de mecánica molecular para la simulación de biomoléculas, las cuales son de dominio público y son utilizadas en una variedad de programas de simulación [37].

3) **Ángulo de torsión (ángulo diedro):** es el ángulo definido por 2 planos [38].

4) **Applet:** es un componente de una aplicación que se ejecuta en el contexto de otro programa, como por ejemplo un navegador web. Un *applet* de *Java* es un programa especial de *Java* que un navegador, habilitado con tecnología *Java*, puede descargar desde Internet y luego ejecutar. Generalmente está incrustado dentro de una página web [39].

5) **Átomo:** es una unidad básica de la materia que está conformada por un núcleo central positivo denso rodeado por una nube de electrones con carga negativa. El núcleo atómico en la mayoría de los casos contiene una mezcla de protones, cargados positivamente, y neutrones, eléctricamente neutros. Un grupo de átomos pueden permanecer unidos entre sí, formando una molécula [40].

6) **Bioinformática:** es la aplicación de la tecnología de computadores a la gestión y análisis de datos biológicos [41].

7) **C++:** es un lenguaje de programación que se creó para extender al exitoso lenguaje de programación *C* con mecanismos que permitieran la manipulación de objetos [42].

8) **Camino de reacción:** es un dibujo que representa las transformaciones que experimentan los reactantes al convertirse en productos involucrando la mínima energía posible [43].

9) **Carga eléctrica de un átomo:** el átomo cargado eléctricamente está influenciado por, y produce, campos electromagnéticos. La carga eléctrica de un átomo puede ser positiva o negativa, lo que depende del número de protones y electrones que contenga. Generalmente el número de electrones cambia, mientras que los protones suelen permanecer constantes en casi toda la química. Si el átomo contiene la misma cantidad de protones y electrones, es eléctricamente neutro. El átomo tiene una carga eléctrica positiva si contiene mayor cantidad de protones que de electrones, y tiene una carga eléctrica negativa si contiene más electrones que protones [40] [44].

10) **CHARMM:** es un conjunto de campos de fuerza para proteínas utilizados en dinámica molecular, que a su vez es utilizada en el estudio de moléculas [45].

11) **CMake:** es un sistema multiplataforma y de código abierto, conformado por una familia de herramientas diseñadas para construir, probar y empaquetar *software*. Las herramientas generan y ejecutan una gran variedad de tareas que los desarrolladores de *software* hacen en sus actividades del día a día, entre ellas: compilar código fuente de la computadora en código binario, hacer que el sistema del *software* esté disponible para ser utilizado, y crear la documentación y/o actualizaciones del *software* [46] [47].

12) **CML:** *Chemical Markup Language* o *CML* es un lenguaje que manipula información molecular usando herramientas como *XML* y *Java* [48]. Puede representar la estructura química de las moléculas, reacciones, datos de química computacional, entre otros datos [49].

13) **Daylight:** es una compañía privada que ofrece a compañías científicas *software* de química de alta calidad, cuya arquitectura se puede modificar [50].

14) **Daylight SMILES:** es una extensión del formato *SMILES* realizada por la compañía *Daylight*, que incluye nuevas funcionalidades y dos lenguajes químicos nuevos: *SMARTS* y *SMIRKS* [50].

15) **Dinámica Molecular:** es una forma de simulación por computadora en la que a los átomos y moléculas se les permite interactuar durante un período de tiempo por aproximaciones de la Física, dando una visión del movimiento de las partículas. Este tipo de simulación se utiliza con frecuencia en el estudio de proteínas y biomoléculas [51]**.**

16) **Eigen2:** es una biblioteca gratuita, liviana y multiplataforma de plantillas de *C++* que permite el manejo de conceptos de álgebra lineal como: vectores, matrices y algoritmos relacionados con ellos [52].

17) **Encapsulated PostScript (EPS):** es un formato estándar para importar y exportar archivos en lenguaje de programación *PostScript* en todos los ambientes. El propósito del archivo *EPS* es ser incluido como una ilustración en otras descripciones de páginas en *PostScript*. El archivo *EPS* puede contener cualquier combinación de texto, gráficas e imágenes [53].

18) **Enlaces dobles E/Z:** son enlaces dobles que pertenecen a un compuesto molecular orgánico, cuya configuración es definida por la notación *E-Z*. Siguiendo un conjunto de reglas definidas (las reglas de prioridad de Cahn-Ingold-Prelog), a cada sustituyente en un enlace doble se le asigna una prioridad. Si los dos grupos de mayor prioridad están en lados opuestos del doble enlace, se le asigna al enlace la configuración *E* (de "*entgegen*", la palabra alemana para "*opuesto*"). Si los dos grupos de mayor prioridad están en el mismo lado del doble enlace, se le asigna al enlace la configuración *Z* (de "*zusammen*", la palabra alemana para "*juntos*") [54].

19) **Estereoquímica:** es una subdisciplina que implica el estudio de la disposición espacial relativa de los átomos dentro de las moléculas [55].

20) **FDPB:** es un método basado en soluciones a la ecuación Poisson-Boltzmann (*PBE*) que calculan el cambio del valor de *pKa* de una cadena lateral de un aminoácido cuando esa cadena lateral se mueve de un estado hipotético completamente apartado de su posición en la proteína [56].

21) **Gaussian:** es un *software* comercial publicado inicialmente en 1970 como *Gaussian* 70. Se utiliza en química computacional para realizar cálculos sobre moléculas utilizando orbitales gaussianos, decisión que se tomó para mejorar la eficiencia del programa con las limitaciones del *hardware* de ese entonces. Ha sido actualizado constantemente a través de los años [57].

22) **Git:** es un *software* de control de versiones de sistema distribuido, gratuito y de código abierto; que fue diseñado para manejar todo tipo de proyectos, de pequeños a muy grandes, con velocidad y eficiencia. Cada clon de *Git* es un repositorio entero con la historia completa del proyecto y con la capacidad de realizar una búsqueda completa en cada versión de la aplicación, que no depende del acceso a la red o a un servidor central. El proceso de ramificación y mezcla de versiones es fácil y rápido de hacer. Es utilizado por el grupo de programación del núcleo del sistema operativo *GNU/Linux* [58].

23) **Grupo R:** también llamado cadena lateral (*side chain*), es un grupo químico unido a la cadena principal de una molécula que representa la parte desconocida o sin especificar de esa molécula. Usualmente R es usado como la representación del grupo alquilo (*alkyl*) en diagramas de estructuras químicas [20].

24) **InChI:** es un identificador no propietario, desarrollado por la *International Union of Pure and Applied Chemistry* (*IUPAC*), que representa estructuras químicas que se encuentran tanto en bases de datos públicas como privadas, sin permitir que existan 2 estructuras con la misma representación. Puede ser utilizado en fuentes de datos impresas y electrónicas, lo que permite un manejo sencillo de enlaces a diversas compilaciones de datos. Su nombre proviene de *IUPAC International Chemical Identifier* [59].

25) **Isótopos:** son cada uno de los elementos químicos que poseen el mismo número de protones y distinto número de neutrones. Todos los isótopos de un elemento ocupan el mismo lugar en la tabla periódica y poseen las mismas propiedades químicas.

26) **IUPAC:** es una organización que sirve para progresar en los aspectos de las ciencias químicas en todo el mundo y contribuir en la aplicación de la química al servicio de la humanidad. Por ser un ente científico, internacional, no gubernamental y objetivo, *IUPAC* puede encargarse de muchos problemas globales relacionados con las ciencias químicas [59].

27) **JavaScript:** antes conocido como *LiveScript*, es un lenguaje web de secuencias de comando simple y multi-plataforma de *Netscape*, sólo vagamente relacionado con *Java*, que es una marca de *Sun*. *JavaScript* está íntimamente ligado a la *World Wide Web*, y actualmente funciona sólo en tres ambientes: como un lenguaje de secuencias de comando del lado del servidor, como un lenguaje integrado en un documento *HTML* analizado por un servidor, y como un lenguaje integrado que se ejecuta en navegadores web donde es la parte más importante del *DHTML*. Tiene una sintaxis simplificada tipo *C* y está estrechamente integrado con el *DOM* (*Document Object Model*) del navegador. Es útil para la implementación de formularios mejorados, interfaces web sencillas de bases de datos y mejoras en la navegación [60].

28) **JDK:** es un ambiente de desarrollo de la corporación *Oracle* que permite la creación y prueba de aplicaciones, *applets* y componentes usando el lenguaje de programación *Java*. Su nombre proviene de *Java Development Kit* [61].

29) **Mecánica Molecular:** es un método de la química computacional que utiliza la mecánica newtoniana para modelar sistemas moleculares. La energía potencial de todos los sistemas en la mecánica molecular se calcula utilizando campos de fuerza. La mecánica molecular puede ser utilizada para estudiar moléculas pequeñas, así como sistemas biológicos grandes o conjuntos de materiales con miles a millones de átomos [62].

30) **MMFF94:** es la primera familia de campos de fuerza desarrollada por *Merck Research Laboratories*, basado en el campo de fuerza *MM3*, que no es optimizada para un único uso (como la simulación de proteínas o pequeñas moléculas), sino que trata de tener un buen rendimiento al ejecutar una gran variedad de cálculos de química orgánica. Los parámetros en el campo de fuerza se obtuvieron a partir de datos de computadora [63].

31) **Molinspiration:** es una organización independiente de investigación centrada en el desarrollo y aplicación de técnicas modernas de química computacional, especialmente en relación con Internet [64].

32) **Nucleón:** en Física, es un nombre colectivo para dos bariones: el neutrón y el protón [65].

33) **OpenGL:** es una especificación estándar que define una *API* multilenguaje y multiplataforma para escribir aplicaciones que produzcan gráficos 2D y 3D. La interfaz consiste en más de 250 funciones diferentes que pueden usarse para dibujar escenas tridimensionales complejas a partir de primitivas geométricas simples, tales como puntos, líneas y triángulos. Su nombre viene de la frase "*Open Graphics Library*", que significa "*Biblioteca Gráfica Abierta*" [66].

34) **OpenLabel:** es una caja de herramientas químicas diseñada para hablar la gran variedad de lenguajes de los datos químicos. Es un proyecto abierto y de colaboración que permite a cualquier persona buscar, convertir, analizar, o

almacenar datos de modelado molecular, química, materiales de estado sólido, bioquímica o áreas relacionadas [67].

35) **Par solitario:** es un par de electrones de valencia que no se vinculan ni comparten con otros átomos [68] [69] [70].

36) **PDB:** es un archivo que contiene información sobre estructuras de proteínas, ácidos nucleicos o ensamblajes complejos descubiertos experimentalmente por biólogos y bioquímicos de todo el mundo, utilizando Cristalografía de Rayos X o Resonancia Magnética Nuclear. Su nombre proviene de *Protein Data Bank* (Banco de Datos de Proteínas). Algunas bases de datos que almacenan la estructura tridimensional de las moléculas utilizan este tipo de archivos [71].

37) **Plugin:** es un archivo que contiene datos utilizados para modificar, mejorar o extender el funcionamiento de un programa de aplicación padre [60].

38) **Policristal:** es un agregado de pequeños cristales de cualquier sustancia, a menudo llamados cristalitas o granos cristalinos por su forma incorrecta. Muchos de los materiales del origen tanto natural como sintético (minerales, metales, aleaciones, cerámica, etc.) son policristales [72].

39) **Python:** es un lenguaje de programación interpretado de código abierto, que permite dividir el programa en módulos reutilizables desde otros programas creados en: el mismo lenguaje, *C*, *C++* o *Java* (a través de *Jython*). Tiene módulos incluidos que proporcionan operaciones de entrada y salida de archivos, llamadas al sistema, sockets e interfaces *GUI* (Interfaz Gráfica con el Usuario) [73].

40) **Psicometría:** es la disciplina que se encarga de medir fenómenos psíquicos.

41) **Qt4:** es una aplicación multiplataforma y un *framework* de interfaz de usuario. Usando *Qt4* se pueden escribir aplicaciones web una vez y hacerlas disponibles para ser utilizadas en sistemas operativos de escritorio, móviles e integrados, sin reescribir el código fuente [74].

42) **Química cuántica:** es una rama de la química teórica en la cual se aplica la mecánica cuántica y la teoría cuántica de campos en problemas de química. Describe matemáticamente el comportamiento fundamental de la materia a escala molecular. Una aplicación en esta rama es el estudio del comportamiento de átomos y moléculas, en cuanto a sus propiedades ópticas, eléctricas, magnéticas y mecánicas, así como su reactividad química, sus propiedades redox, etc.; pero también se estudian materiales, tanto sólidos extendidos como superficies [75].

43) **Química teórica:** es una disciplina que utiliza los fundamentos de la Física para explicar o predecir fenómenos químicos. Se puede dividir ampliamente en ramas como la estructura electrónica, dinámica y mecánica estadística [75].

44) **Radical:** es un agrupamiento de átomos que interviene como una unidad en un compuesto químico y pasa inalterado de unas combinaciones a otras.

45) **Sistema de tiempo de ejecución:** es una colección de *software* diseñado para apoyar la ejecución de programas de computación escrito en algún lenguaje de programación. El sistema puede proporcionar servicios de *software* como subrutinas y bibliotecas para operaciones comunes, implementación de comandos de lenguajes de programación, verificación de tipos, depuración, e incluso generación y optimización de código. Alivia a los programadores la carga de escribir código para tareas comunes como dibujar texto en la pantalla o establecer una conexión con Internet [76].

46) **SMARTS:** es un lenguaje que permite especificar patrones y propiedades de moléculas para la búsqueda detallada de subestructuras, basándose en *Daylight SMILES* [50].

47) **SMILES:** *Simplified Molecular Input Line Entry Specification* (Especificación de Ingreso Lineal Molecular Simplificado o *SMILES*), es un formato que describe sin ambigüedades la estructura de una molécula usando cadenas *ASCII* cortas. Las cadenas *SMILES* pueden ser importadas por la mayoría de los editores moleculares para su conversión en dibujos bidimensionales o modelos tridimensionales de las moléculas [77].

48) **SMIRKS:** es un lenguaje que es una versión limitada de la funcionalidad de reacciones de *SMARTS*. Se caracteriza por interpretar nuevos patrones entre átomos unidos por enlaces para definir reacciones químicas genéricas [52]. Puede ser utilizado para crear nuevas reacciones, manipular moléculas y crear nuevas moléculas [50].

49) **SDK:** es un conjunto de herramientas de desarrollo de *software* que permite la creación de programas para un cierto paquete de *software*, *framework*, plataforma de *hardware*, computadora, consola de videojuegos, sistema operativo, entre otras plataformas [61] [78] [79].

50) **Vibración molecular:** es una vibración que se produce debido al constante movimiento de los átomos en una molécula, unos con respecto a otros, mientras que a su vez toda la molécula tiene un movimiento de traslación y rotación constante. La frecuencia del movimiento periódico se conoce como frecuencia de vibración, y está relacionada con el espectro infrarrojo (*IR*) de la molécula [80].

REFERENCIAS

1. Red Escolar Nacional (RENa), "*Computadoras y ambiente*", Página web: http://www.rena.edu.ve/cuartaEtapa/Informatica/Tema16.html.

2. E. González-Manet, "*Nuevas formas de Comunicación y Cultura: el impacto de las nuevas tecnologías*", Editorial Pablo de la Torriente Brau, p. 11, 1998.

3. S. J. Smith, B. T. Sutcliffe, "*The development of Computational Chemistry in the United Kingdom*", Reviews in Computational Chemistry, pp. 271-316, John Wiley & Sons, Inc., 2007.

4. Universia, "*Cuando la computación le da otro giro a la química*", Página web: http://noticias.universia.com.ar/en-portada/noticia/2004/05/13/367807/computacion-da-otro-giro-quimica.html.

5. Herbert J. Bernstein, "*The Historical Context of RasMol Development*", Página web: http://rasmol.org/history.html.

6. *MIT Technology Review,* "*Molecular Imaging*", Biomedicina, Página web: http://www.technologyreview.com/biomedicine/.

7. Francis T. Marchese, "*Interactive Molecular Visualization at the Interface*", Capítulo 11 del libro "*Trends in Interactive Visualization*"; Robert Liere, Tony Adriaansen y Elena Zudilova-Seinstra (Eds.), 2009.

8. José Ramón Bertomeu Sánchez y Antonio García Belmar, "*La historia de la química: pequeña guía para navegantes. Parte I: viejas y nuevas tendencias*", Anales de la Real Sociedad Española de Química, ISSN 1575-3417, 104(1):56-53, 2008.

9. José Ramón Bertomeu Sánchez y Antonio García Belmar, "*La historia de la química: pequeña guía para navegantes. Parte II: libros, revistas, sociedades, centros de investigación y enseñanza*", Anales de la Real Sociedad Española de Química, ISSN 1575-3417, 104(2):146-153, 2008.

10. Stephen Wilson. "*Chemistry by Computer: an Overview of the Applications of Computers in Chemistry*". *Theorical Chemistry Department*. Universidad de Oxford. Oxford, Inglaterra. *Plenum Press*. Nueva York y Londres. 1986.

11. Amelia Baldes, Ramón Olivas, César Contreras, Marco Chávez, Alfredo Urbina, Carlos de la Vega, Francisco Pérez, Luz María Rodríguez y María Elena Fuentes, "*Laboratorio de Química Computacional*", Facultad de Ciencias Químicas, Universidad Autónoma de Chihuahua, Video promocional, Publicado el 3 de Octubre de 2006, Página web: http://www.youtube.com/watch?v=Jtmu0XoNWlo.

12. *Molinspiration*, "*Ancient (i.e. pre-Java) History*", *JME*, Página web: http://www.molinspiration.com/jme/prejme.html.

13. Química Computacional en la *AHPCC*, "*Xmol*", Página web: http://www.hpc.unm.edu/~chem/xmol/xmol.html.

14. Philip Deacon, "*Investigation of the Different MIME Types related to Molecular Biology and Chemistry Visualization Programs*", *EC-CSE* (*EdCenter on Computational Science and Ingineering*), Página web: http://www.edcenter.sdsu.edu/repository/Mbcw/chemmime.html

15. Blog de los Productos de *Metamolecular*, "*About ChemWriter*", Página web: http://products.metamolecular.com/2009/10/28/chemwriter-1-3-4.

16. Página Oficial de *ChemWriter*, "*Download ChemWriter 1.3.4*", Página web: http://chemwriter.com/evaluate.

17. Proyecto *CDK*, "*JChemPaint*", Página web: http://sourceforge.net/apps/mediawiki/cdk/index.php?title=JChemPaint.

18. *JsMolEditor*, "*jsMolEditor: Molecule Editor of JavaScript*", Página web: http://chemhack.com/jsmoleditor/.

19. *Molinspiration Cheminformatics*, "*JME Molecular Editor*", Página web: http://www.molinspiration.com/jme/index.html.

20. *ChemAxon*, "*Marvin, Calculator Plugin and Chemical Terms Demo*", Página web: http://www.chemaxon.com/marvin/sketch/index.php.

21. S. A. Rahman, M. Bashton, G. L. Holliday, R. Schrader y J. M. Thornton, "*Small Molecule Subgraph Detector* (*SMSD*) *toolkit*", *Journal of Cheminformatics*, 1:12, 2009.

22. *National Center for BioTechnology Information*, "*PubChem*", Página web: http://pubchem.ncbi.nlm.nih.gov/.

23. García Cubas Robert Spassky y Mijares Toussaint Michael Rodney, "*Guía de Química: Teoría y Práctica*", Escuela Dr. Jesús María Bianco, Facultad de Farmacia, Universidad Central de Venezuela, Venezuela, pp. 46-49. Agosto, 2003.

24. *Biomolecular Modeling*, "*Ascalaph Designer*", *Ascalaph*, Página web: http://www.biomolecular-modeling.com/Ascalaph/Ascalaph_Designer.html.

25. *Open Molecules*, "*Avogadro*", *Wiki*, Página Principal, Página web: http://avogadro.openmolecules.net/wiki/Main_Page.

26. *BALL: Biochemical Algorithms Library*, "*BALLView*", Página web: http://www.ball-project.org/Ballview.

27. Sistema Operativo *GNU*, "*GNU General Public License*", Página web: http://www.gnu.org/copyleft/gpl.html.

28. *Jmol*, *Wiki*, Página web: http://wiki.jmol.org/index.php/Main_Page.

29. *SourceForge*, *Jmol*, Páginas del proyecto *Jmol* en *SourceForge*, Listas de correo, Página web: http://jmol.sourceforge.net/project/.

30. *Jmol*: un visor *Java* de código abierto para estructuras químicas en tres dimensiones, "*Historia del desarrollo de Jmol*", Página web: http://jmol.sourceforge.net/history/index.es.html.

31. "*Jmol: an open-source Java viewer for chemical structures in 3D*", Página web: http://www.jmol.org.

32. *SourceForge*, Proyectos, "*Jmol*", Archivos, Página web: http://sourceforge.net/projects/jmol/files/Jmol/.

33. *Eclipse Foundation*, "*Eclipse*", Página web: http://www.eclipse.org/.

34. *Polarion Software*, Productos, "*Subversive SVN Team Provider Connector Discovery*", Página web: http://www.polarion.com/products/svn/subversive/connector_discovery.php.

35. Rensis Likert, "*A Technique for the Measurement of Attitudes*", *Archives of Psychology*, 22(140): 1–55, 1932.

36. A. D. McNaught y A. Wilkinson, *Compendium of Chemical Terminology*: *The Gold Book*, *Second Edition*, *Blackwell Scientific Publications*, *Oxford*, 1997.

37. "*Amber Home Page*", *Assisted Model Building with Energy Refinement*, Página web: http://ambermd.org/.

38. Eric W. Weisstein, "*Dihedral Angle*", *Mathworld*, Página web: http://mathworld.wolfram.com/DihedralAngle.html.

39. *The Java™ Tutorials*, "*Lesson: Applets*", *Sun Developer Network*, Página web: http://java.sun.com/docs/books/tutorial/deployment/applet/index.html.

40. J. Rigaudy y S. P. Klesney, "*Nomenclature of Organic Chemistry*". *International Union of Pure and Applied Chemistry*. *Commission on the Nomenclature of Inorganic Chemistry*. Oxford, 1990.

41. *European Bioinformatics Institute*, "*What is Bioinformatics?*", Página web: http://www.binf.umn.edu/about/whatsbinf.php.

42. Bjame Stroustrup, "*El Lenguaje de Programación C++*", Addison-Wesley, Madrid, 1998.

43. I.S. Butler y J.F. Harrod, "*Química Inorgánica: Principios y Aplicaciones*". Addison-Wesley Iberoamericana, 1992.

44. Roald K. Wangsness, "*Electromagnetic Fields*", 2da Edición, *Wiley*, 1986.

45. MacKerell, Jr. A.D., Banavali N., y Foloppe N., "*Development and current status of the CHARMM force field for nucleic acids*", *Biopolymers*, 56 (4): 257–265, 2001.

46. Kitware, "*CMake*", Página web: http://www.cmake.org/.

47. Mike Clark, "*Pragmatic Project Automation: How to Build, Deploy, and Monitor Java Apps*", Los Programadores Prácticos, 2004.

48. P. Murray-Rust y H. S. Rzepa, "*Chemical Markup, XML, and the Worldwide Web. 1. Basic Principles*", DOI: 10.1021/ci990052b, *Journal of Chemical Information and Computer Sciences.* 39 (6): 928–942, *American Chemical Society*, 1999.

49. G.L. Holliday, P. Murray-Rust y H. S. Rzepa, "*Chemical Markup, XML and the World Wide Web. Part 6. CMLReact; An XML Vocabulary for Chemical Reactions*", *Journal of Chemical Information and Modeling.* 46 (1): 145–157, *American Chemical Society*, 2006.

50. *Daylight*, *Chemical Information Systems Inc.*, Página web: http://www.daylight.com/.

51. B. J. Alder y T. E. Wainwright, "*Studies in Molecular Dynamics. I. General Method*", *Journal of Chemical Physics*, 31(2), 1959.

52. *Tux Family*, "*Eigen*", Página Principal, Información general, Página web: http://eigen.tuxfamily.org/.

53. Allen Braunsdorf, "*PostScript: Answers and Questions*", Página web: http://www.postscript.org/FAQs/language/FAQ.html.

54. *IUPAC* (*International Union of Pure and Applied Chemistry*), "*Recommendation R-7.1.2.*", Una Guía para la Nomenclatura de Compuestos Orgánicos de *IUPAC*, Oxford, *Blackwell Science*, 1993.

55. Jerry March, "*Advanced Organic Chemistry: Reactions, Mechanisms, and Structure*", tercera edición, *New York*: Willey, 1985.

56. A. Onufriev, D. A. Case y G. M. Ullmann, "*A Novel View of pH Titration in Biomolecules*", *Biochemistry*, 40 (12), pp 3413–3419, 2001.

57. Emilio San Fabian, "*El programa Gaussian (92)*", Página web: http://www.ua.es/cuantica/docencia/otros/g-92/g-92.html.

58. "*Git: the Fast version control system*", Página web: http://git-scm.com/.

59. "*IUPAC: International Union of Pure and Applied Chemistry*", Página web: http://www.iupac.org/.

60. "*Free On-line Dictionary of Computing* (*FOLDOC*)", Página web: http://foldoc.org/.

61. Oracle, "*Java SE Downloads*", JDK, Página web: http://www.oracle.com/technetwork/java/javase/downloads/index.html.

62. Allinger N. L. y Burkert U., "*Molecular Mechanics*", Sociedad Química Americana, 1982.

63. Thomas A. Halgren, "*Merck Molecular Force Field. I. Basis, form, scope, parametrization, and performance of MMFF94*", *Journal of Computational Chemistry*, 17(5-6):490-519, Abril, 1996.

64. "*Molinspiration*", Página web: http://www.molinspiration.com/.

65. Milton Orchin, Roger S. Macomber, Allan Pinhas y Marshall Wilson, "*The Vocabulary and Concepts of Organic Chemistry: Atomic Orbital Theory*", Segunda Edición, 2005.

66. "*OpenGL: the Industry`s Foundation for High Performance Graphics*", Página web: http://www.opengl.org/.

67. "*Open Babel: The Open Source Chemistry Toolbox*", Página web: http://openbabel.org/wiki/Main_Page.

68. Marye Anne Fox y James K. Whitesell, "*Organic Chemistry*", segunda edición, 2001.

69. John McMurry, "*Organic Chemistry*", quinta edición, 2000.

70. J.D. Lee, "*Concise Inorganic Chemistry*", cuarta edición, 1991.

71. *RCSB*, "*PDB: Protein Data Bank*", Portal de Información de Estructuras Macromoleculares Biológicas, Página web: http://www.pdb.org/.

72. O. Raymond, "*Textura Global y Propiedades Físicas de Muestras Policristalinas*", Tesis de Doctorado en Física, Universidad de la Habana, 1998.

73. Jim Knowlton, "*Python*", primera edición, Anaya Multimedia-Anaya Interactiva, 2009, p. 272.

74. Corporación Nokia, "*Qt: cross-platform application and UI framework*", Productos, Página web: http://qt.nokia.com/products.

75. Attila Szabo y Neil S. Ostlund, "*Modern Quantum Chemistry: Introduction to Advanced Electronic Structure Theory*", *Dover Publications*, 1996.

76. Tim Lindholm y Frank Yellin, "*The Java Virtual Machine specification*", 2da. edición, Addison-Wesley, 1999.

77. E. Anderson, G.D. Veith y D. Weininger, "*SMILES: A line notation and computerized interpreter for chemical structures*", Número de reporte EPA/600/M-87/021, *U.S. EPA*, *Environmental Research Laboratory-Duluth*, Duluth, Minnesota 55804, Estados Unidos, 1987.

78. ABBYY, "*Software Development Kits*", Página web: http://latam.abbyy.com/sdk/.

79. *Olympus*, "*Developer Program website*", Página web: http://developer.olympus.com/.

80. L.D. Landau y E. M. Lifshitz, "*Mechanics*", 3ra. edición, *Pergarmon Press*, 1976.