UNIVERSIDAD DE LA REPUBLICA (URUGUAY)
FACULTAD DE INGENIERIA
INSTITUTO DE COMPUTACION

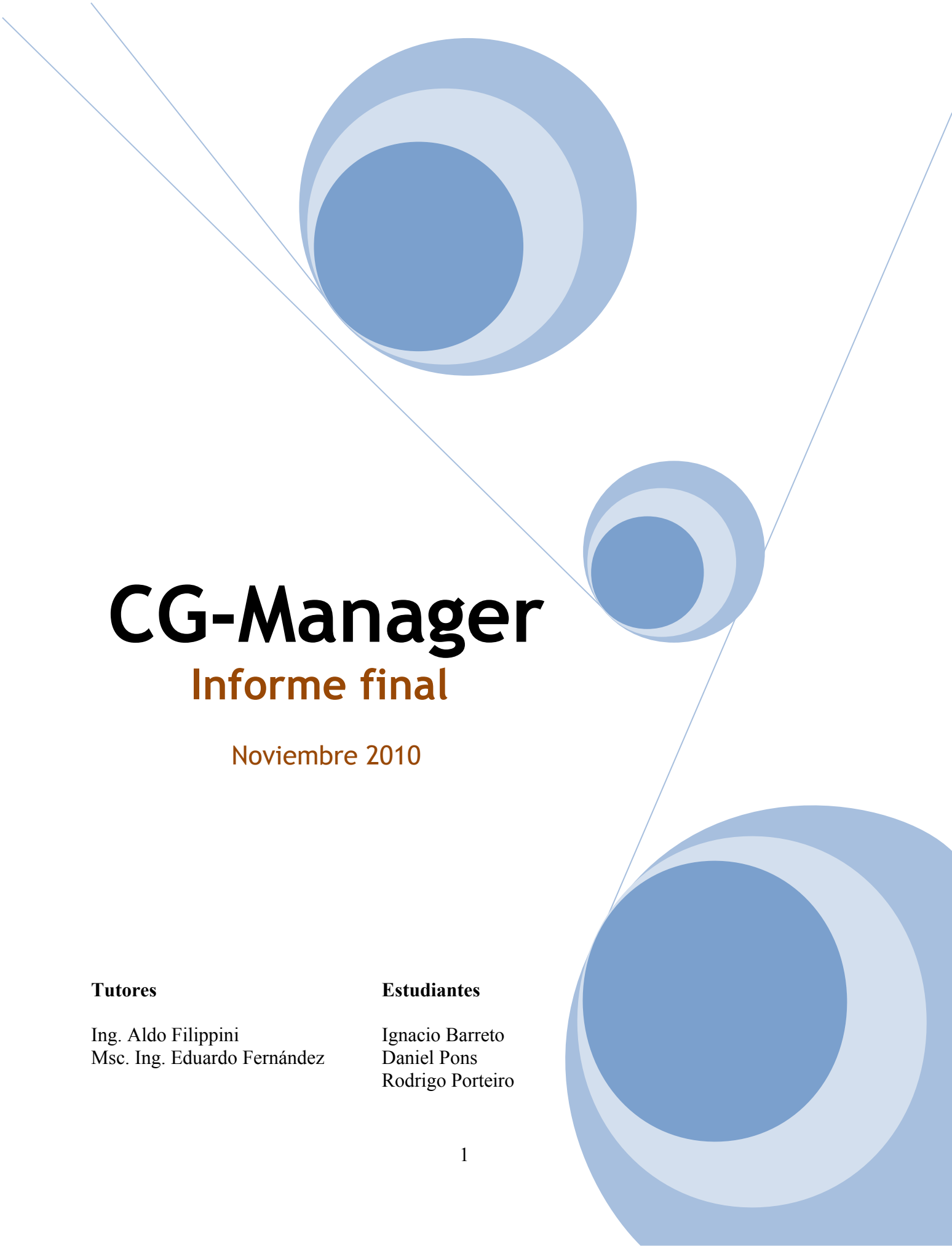
Informe del Proyecto de grado

COURSE GRAIN MANAGER
INFORME FINAL

Ignacio Barreto **Daniel Pons** **Rodrigo Porteiro**

Tutores:
Ing. Aldo Filippini
Msc. Ing. Eduardo Fernández

Montevideo, noviembre de 2010



CG-Manager

Informe final

Noviembre 2010

Tutores

Ing. Aldo Filippini
Msc. Ing. Eduardo Fernández

Estudiantes

Ignacio Barreto
Daniel Pons
Rodrigo Porteiro

Resumen

El Grupo de Simulaciones Biomoleculares del Institut Pasteur de Montevideo tiene como principal objetivo la modelación a nivel atómico de sistemas biológicos relevantes. Una de las técnicas utilizadas para el estudio de estos sistemas son las simulaciones de dinámicas moleculares. Estas son técnicas que permiten predecir y simular la estructura y el comportamiento de una molécula en forma computacional, con el objetivo de determinar la energía total del sistema molecular. Algunas de las técnicas consideran el cálculo de la energía de la molécula únicamente en función de las posiciones de los núcleos a través del conjunto de ecuaciones conocidos como Campos de fuerza. En estos métodos se calcula el movimiento de los átomos mediante las ecuaciones según las leyes del movimiento de Newton en forma iterativa a lo largo del tiempo, obteniéndose un nuevo conjunto de coordenadas espaciales en un tiempo $t + \Delta t$. Existen limitaciones en los tiempos de simulación debido a la cantidad de posiciones de átomos que deben ser calculadas en cada paso de tiempo. En general la duración que abarcan oscila entre los pocos nanosegundos llegando en algunos casos específicos hasta el microsegundo de simulación. Una forma de poder alcanzar mayores tiempos de simulación es transformar la molécula en su representación *All-Atoms* a la representación *Coarse-Grain*. Esta transformación es una condensación o reducción de la molécula en donde cada elemento representa a un conjunto de átomos, denominados Superátomo.

Actualmente no existen aplicaciones informáticas en donde el usuario pueda definir en forma controlada modelos *Coarse-Grain* a partir de moléculas *All-Atoms*, para luego poder utilizarlos en simulaciones de dinámicas moleculares.

Este proyecto de grado propone la construcción de una aplicación en la cual a partir de ciertos archivos que representan a la molécula, el usuario pueda editar y visualizar propiedades de la misma a través de una interfaz gráfica de usuario y definir modelos *Coarse-Grain* a partir de un modelo *All-Atoms*. A su vez, se implementan dos algoritmos de transformación. Un algoritmo transforma la molécula *All-Atoms* a *Coarse-Grain* a partir de las definiciones realizadas por el usuario. A su vez, se encuentra el algoritmo inverso, el cual a partir de un modelo *Coarse-Grain* y de un archivo que contiene información de la condensación realizada sobre el modelo original, obtiene en forma aproximada la molécula original *All-Atoms*. Ambos algoritmos generan como salida un conjunto de archivos en cierto formato los cuales pueden utilizarse en otros programas de simulación de dinámicas moleculares y de visualización molecular.

Palabras clave: simulación de dinámicas moleculares, modelos *All-Atoms*, modelos *Coarse-Grain*, algoritmo de transformación.

Agradecimientos

Sergio Pantano, Pablo Dans y el resto de los integrantes del Grupo de Simulaciones Biomoleculares del Institut Pasteur, a Ramiro Patiño, Sofía Castro, Claudia Guinovart.

A nuestros padres por habernos apoyado durante todos estos años de estudios en facultad, a nuestras novias y/o esposas que nos bancaron estar hasta altas horas varios días a la semana o fines de semana y a Aldo Filippini por haber sido nuestro tutor y acompañado en las numerosas reuniones con la gente del Pasteur.

Tabla de contenido

1 Introducción

La caracterización de las interacciones moleculares de interés biomédico requiere la visualización tridimensional de sus estructuras y superficies de contacto entre diversos actores biológicos (proteínas, ácidos nucleicos, membranas, etc.). Dicha representación gráfica debe permitir identificar clara y velozmente las características fisicoquímicas de los átomos/moléculas a partir de sus coordenadas cartesianas y su evolución temporal. Actualmente existen diversos paquetes informáticos comerciales y de libre distribución, que realizan estas tareas para los problemas más comunes de la bioinformática estructural. Sin embargo, una de las actividades del Grupo de Simulaciones Biomoleculares (SBM) del Institut Pasteur de Montevideo se enfoca en el tratamiento de interacciones entre grandes sistemas moleculares para los cuales no existen aún implementaciones específicas.

Estas interacciones entre sistemas moleculares se enmarcan en el área de las simulaciones de dinámica molecular. Este es un método de simulación computacional en donde átomos y moléculas interactúan durante un cierto período de tiempo bajo las leyes de la física (leyes del movimiento de Newton), con el objetivo de estudiar la relación entre la estructura molecular y el movimiento de los átomos. Existen limitaciones en los tiempos de simulación debido a la cantidad de posiciones de átomos que deben ser calculadas en cada paso de tiempo. Las simulaciones deben considerar muchos pasos discretos de unidades de tiempo, para ser comparable con los tiempos que demandan los procesos naturales que son estudiados. En general la duración que abarcan oscila entre los pocos nanosegundos llegando en algunos casos específicos hasta el microsegundo de simulación.

Una de las técnicas para simular una mayor cantidad de nanosegundos, léase centenas o miles de nanosegundos, es convertir la molécula original, la cual está en representación *All-Atom* (modelo de representación de la molécula en donde se consideran a todos los átomos que la componen), a una versión reducida que contiene una menor cantidad de átomos pero que mantiene las propiedades esenciales de la molécula original. Esta representación se conoce como *Coarse-Grain* y permite que, a través de los programas que realizan simulaciones de dinámica molecular, se tengan que calcular las posiciones resultantes de una menor cantidad de átomos en cada iteración de la simulación. Hoy en día no existen implementaciones específicas que realicen la transformación *All-Atom* a *Coarse-Grain* cuyas características sean las deseadas por el Grupo de Simulaciones Biomoleculares.

Como resultado de este proyecto de grado, se creó una aplicación que cumple con los requerimientos deseados. Entre ellos se destacan los algoritmos de transformación de representación molecular, *All-Atom* ←→ *Coarse-Grain*.

1.1 Motivación

El Grupo de Simulaciones Biomoleculares aplica una variedad de técnicas computacionales como bioinformática estructural, modelado molecular y simulaciones de dinámica molecular. Estas técnicas tienen como objetivo proveer nuevas perspectivas relacionadas a mecanismos moleculares en las que se incluyen: interacciones proteína-proteína, lípido-proteínas, simulaciones de modelos *Coarse-Grain* específicos de ADN/ARN y relaciones estructura-función ligadas a problemas particulares de relevancia biomédico y biológico.

Los modelos llamados *All-Atom* representan la estructura de la molécula a nivel atómico, considerando todos los átomos de la misma. Estos modelos son utilizados en simulaciones de dinámica molecular, en los cuales a partir de un conjunto de coordenadas espaciales y de velocidades, se calcula la fuerza resultante en todos los átomos de la molécula en cuestión. Esto se realiza a lo largo del tiempo en forma iterativa, calculando las soluciones de las ecuaciones de movimiento según las leyes de Newton y obteniéndose un nuevo conjunto de coordenadas espaciales en un tiempo $t + \Delta t$.

Por su parte, los modelos *Coarse-Grain* son una condensación o reducción del modelo *All-Atom*, definiéndose de tal forma que mantienen los rasgos esenciales de esa molécula pero con una menor cantidad de átomos. Cada elemento del modelo *Coarse-Grain* son esferas o centroides efectivos, llamados Superátomos (SAT), en el que cada uno de ellos representa la información fundamental de un subconjunto de átomos del modelo *All-Atom*. Las conectividades son redefinidas en los modelos *Coarse-Grain* considerando a los superátomos, pudiendo no tener relación con las conectividades del modelo original *All-Atom*. Estos modelos reducen la complejidad del sistema siendo compatible con la representación *All-Atom*.

El hecho de realizar simulaciones de los modelos *All-Atom* tiene como ventaja que no solo reproducen en forma aproximada computacionalmente los resultados experimentales, sino que muchas veces proveen datos que no son sencillos de obtener a través de experimentos de laboratorio. De hecho, las simulaciones de dinámicas moleculares *All-Atom* son utilizadas en el estudio de las propiedades energéticas, propiedades dinámicas y de mecánica estadística en la formación de pequeñas estructuras secundarias de cadenas de péptidos. Por más información ver [1].

Este proyecto nos representó un gran atractivo a la hora de decidir realizarlo. Para nosotros fue muy importante que el producto resultante pudiera ser utilizado asiduamente y permitiera contribuir a la investigación científica, en lugar de enmarcarse solamente como una investigación teórica. En nuestra opinión, el proyecto se sitúa en un marco conjunto entre la computación y la ciencia en donde la ciencia resulta beneficiada por el primero. Por estas razones nos generó una importante motivación y un gran compromiso a la hora de cumplir los objetivos y lograr la calidad que demanda una herramienta orientada a la investigación.

1.2 Descripción del problema

Un sistema molecular el cual es sometido a una simulación de dinámica molecular, suele estar limitado en la escala o tiempo de simulación. Esta oscila entre 10ns y 1μs y son considerados de unos miles a millones de átomos. Aquí subyacen dificultades de muestreo que son insuficientes para realizar una evaluación posterior de los mismos. Pueden instrumentarse mejoras en el muestreo, modificando diferentes propiedades que deben ser calculadas o que son tenidas en cuenta en el cálculo de la trayectoria de un átomo. También existe la posibilidad de aumentar la capacidad del hardware que ejecuta la dinámica molecular, así como de mejorar los algoritmos que efectúan los cálculos dentro de una simulación [2]. Una forma de mejorar este aspecto es mediante la utilización de técnicas de alto desempeño computacional, ya sea estudiando la naturaleza del problema o aprovechando las capacidades de cómputo del hardware, considerando un nodo o un conjunto de nodos interconectados, así como también utilizando técnicas de procesamiento en paralelo y/o distribuido.

La razón por la cual se utilizan modelos *Coarse-Grain*, se debe a que hoy en día las simulaciones existentes a nivel atómico con modelos *All-Atom* se realizan en tiempos de simulación significativamente cortos, alrededor de 100ns. Muchos de los procesos relevantes en biología no se pueden visualizar teniendo en cuenta estos tiempos de simulación, es decir que se deben poder abarcar tiempos de simulación mayores. Por otro lado, es deseable poder modelar sistemas y/o moléculas más grandes que incluyan millones de átomos, los cuales, hoy en día resulta complejo simular utilizando la representación *All-Atom*.

Existe un programa, Visual Molecular Dynamics, que permite convertir una molécula *All-Atom* a *Coarse-Grain* de manera automática mediante un algoritmo preestablecido. Si bien en algunos casos es útil, éste no cumple con los requerimientos del Grupo de Simulaciones Biomoleculares.

1.3 Objetivos

El objetivo principal de este proyecto de grado es la construcción de una aplicación enfocándose en las siguientes características:

- Definir un algoritmo para la conversión de una molécula en representación *All-Atom* a una molécula en representación *Coarse-Grain*.

- Definir un algoritmo inverso, es decir, realizar la conversión de una molécula en representación *Coarse-Grain* a *All-Atom*.

- Construir una interfaz gráfica de usuario en donde se puedan visualizar y modificar diferentes propiedades de la molécula, residuos y átomos.

- Implementar la lectura y generación de ciertos archivos que definen a la molécula de modo que estos sean compatibles con otras aplicaciones de dinámicas moleculares.

Como características adicionales a mencionar tenemos que:

- El programa desarrollado debe ser multiplataforma, es decir, sin limitaciones en cuanto al sistema operativo sobre el cual ejecutar.

- No existen restricciones en lo que refiere al lenguaje de programación a utilizar.

- La aplicación debe poder ser distribuida libremente para su uso en la comunidad científica internacional.

1.4 Organización del documento

El documento se organiza de la siguiente manera: en el capítulo 2 se realiza una introducción a algunos de los conceptos teóricos en los que se basa la aplicación desarrollada para el proyecto.

En el capítulo 3 se presenta el estado del arte en lo relativo a aplicaciones de visualización molecular.

En el capítulo 4 se explica la evolución del proyecto desde la idea original hasta la definición final de la herramienta a construida, presentándose junto con el diseño, la arquitectura de la aplicación. También se especifica parte de la implementación de la interfaz de usuario, de ciertos aspectos gráficos y del manejo de archivos.

En el capítulo 5 se profundiza en la descripción de los algoritmos desarrollados para realizar la conversión de la molécula en su representación *All-Atom* a *Coarse-Grain* y viceversa.

Por último en el capítulo 6 se presentan las conclusiones finales y el trabajo a futuro planteado con el objetivo de exponer mejoras a realizar en la aplicación desarrollada.

2 Ámbito del problema

Un problema importante que tuvimos que abordar fue el de familiarizarse con la terminología científica utilizada en bioquímica, que es una de las áreas que abarca el proyecto de grado. Dicha terminología resultó esencial para poder comunicarnos con los investigadores, que son los usuarios finales del proyecto.

Es así que quisimos realizar una introducción a los términos y conceptos de bioquímica que están detrás del producto que desarrollamos, para darle un marco teórico y explicar cómo se refleja en la aplicación. Se desarrollan algunos conceptos teóricos, específicamente en los temas de modelado y simulación molecular utilizados a lo largo de todo el informe.

La lectura del presente capítulo y del Anexo F en forma complementaria, será suficiente para que el lector pueda entender los temas que este proyecto se propone resolver.

1.5 Marco teórico

En esta sección se explican en forma resumida, aspectos teóricos de bioquímica enfocados a los temas que están involucrados dentro de este proyecto de grado.

1.5.1 Modelado Molecular

El modelado molecular es una disciplina que estudia la relación entre la estructura molecular y las propiedades químicas de la materia. Esta rama de la bioquímica se basa en el uso de diferentes métodos computacionales (la mayoría numéricos) para simular, explicar o predecir la estructura tridimensional y las propiedades fisicoquímicas de las moléculas, utilizando en forma adicional aplicaciones gráficas que facilitan su interpretación o permiten su visualización de manera más amigable para el usuario [3]. En el capítulo siguiente correspondiente al estado del arte, se presentan y enumeran algunas de las características de algunos programas computacionales que se enfocan en esta área.

1.5.2 Métodos de simulación molecular en bioquímica

Son métodos que pretenden predecir y simular la estructura y el comportamiento de las macromoléculas, empleando para ello las reglas de la física clásica y de la química general. Existen dos familias de métodos de simulación molecular: los métodos clásicos y los métodos derivados de la mecánica cuántica. Las dos técnicas buscan el mismo objetivo: describir o calcular la energía potencial del sistema a nivel atómico. Pero existen diferencias entre estas.

La mecánica cuántica se basa en el estudio de la energía de un sistema molecular a partir de una configuración nuclear fija, calculando el movimiento de los electrones. La

mecánica clásica, por su parte, considera únicamente la posición de los núcleos de los átomos para realizar dichos cálculos. Además, se basa en el uso de campos de fuerza empíricos y supone que las moléculas son partículas clásicas gobernadas por las leyes de la mecánica clásica de Newton.

En lo que respecta al proyecto de grado, el enfoque y el área de investigación de SBM se basa en los métodos clásicos.

En los métodos clásicos de simulación molecular, la energía de una molécula se expresa únicamente en función de las posiciones de los núcleos. Por lo tanto, no se consideran los electrones de modo explícito. Esto simplifica el cálculo de la energía del sistema en contraposición a la mecánica cuántica.

Este modelo de mecánica clásica considera a los átomos de las moléculas como pequeñas esferas con una determinada masa y que están unidos entre sí por "resortes", los cuales actúan restaurando los valores "naturales" de las distancias y los ángulos entre los diferentes átomos. La aproximación más elemental es usar la ley de Hooke (conocida como aproximación armónica).

El conjunto de ecuaciones y sus parámetros que expresan la energía de la molécula en función de las coordenadas nucleares, es el llamado campo de fuerza (conocido también como "*force-field*"), los cuales se determinan a partir de evidencias experimentales. Dicho de otra manera, el campo de fuerza representa la dependencia que existe entre la energía potencial de un sistema molecular con la geometría del mismo. Esta información puede ser procesada de diversas maneras dando lugar a una serie de técnicas que se explicarán más adelante.

En la figura 1 se presenta la ecuación de la energía potencial en función de los campos de fuerza del sistema. Esta energía se expresa como una suma de términos enlazantes y no enlazantes.

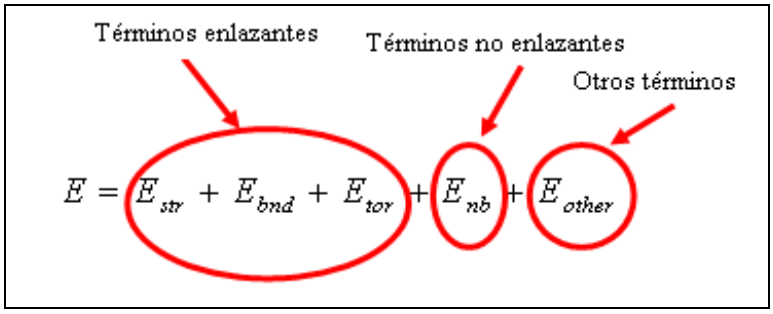


Figura 1 – Expresión de la energía potencial en mecánica clásica.

Los campos de fuerza contienen una serie de parámetros que definen los diversos tipos de interacción entre los átomos y que deben ser conocidos antes de realizar el cálculo. En la ecuación de la figura 1 se distinguen tres términos: los "*Términos enlazantes*" que se relacionan con el enlace existente entre un par de átomos, los *Términos no enlazantes* y *otros términos* que se corresponden a otras propiedades que afectan a la energía.

Dentro de las interacciones de enlace (enlazantes) o "*bondedterms*" se encuentran tres términos: *Strechting* (Str), *Bending* (bnd) y *Torsion* (tor) respectivamente.

Bond Stretching: representa la variación de la energía en relación a los cambios de longitud del enlace. El concepto de bond *stretching* involucra solamente a dos átomos unidos a través de un enlace o conectividad. La constante de fuerza (K_{str}) da la idea de la capacidad de deformación del enlace y la distancia ($l - l_0$) en la cual los dos átomos están en equilibrio. La ecuación de la figura 2 representa la energía total que aporta el bond *stretching* para todas las conectividades existentes en la macromolécula.

$$E_{str} = \sum_{bonds} K_{str}(l - l_o)^2$$

Figura 2 – Energía del Bond *stretching* en la macromolécula.

l: es la distancia entre 2 átomos conectados a través de un enlace.
l_0: es la distancia a la cual están en equilibrio esos 2 átomos.

Se puede observar que el término dentro de la sumatoria se basa en la Ley de Elasticidad de Hook [5]. Lo mismo aplica para los términos de *Angle Bend* y *Torsion*.

Angle Bending: representa la variación de la energía potencial relacionada a los cambios en el ángulo de enlace entre tres átomos unidos a través de dos conectividades. La constante de fuerza (K_{ang}) da una idea de la capacidad de deformación del ángulo y la distancia en la cual el resorte (ángulo) está en equilibrio. La ecuación de la figura 3 representa la energía total que aporta cada ángulo existente en el sistema.

$$E_{bnd} = \sum_{angles} K_{ang}(\Theta - \Theta_o)^2$$

Figura 3 – Energía del *Angle bending* en la macromolécula.

Θ: es el valor del ángulo que forman tres átomos cualesquiera al momento de calcular la energía.
Θ_0: es el valor del ángulo para el cual está en equilibrio.

Torsión: también conocido como diedro o ángulo diedro. Existen dos variedades de ángulos de torsión. Uno son los propios, los cuales representan la rotación respecto a los enlaces químicos. Por otro lado están los impropios, que son conocidos como "*out of plane*" o fuera del plano, estos términos sirven básicamente para reforzar la planaridad de algunos enlaces. Los potenciales de torsión siempre se expresan como series de Fourier como las representadas en la ecuación de la figura 4.

Φ = ángulo de torsión o ángulo diedro.
n = multiplicidad. Denota el número de mínimos cuando el enlace se rota 360º.
Vn = es la "altura de la barrera".
γ = es la fase a la cual se encuentra el ángulo.

$$E_{tor} = \sum_{tor} \sum_{n=1}^{3} \frac{V_n}{2}(1 + \cos n\Phi - \gamma)$$

Figura 4 – Energía de los ángulos diedros en la macromolécula.

El otro término relevante de la ecuación de energía potencial (ecuación de la figura 1), corresponde a las interacciones entre átomos que no están unidos a través de enlaces (también llamado "non bonded terms"). Estas interacciones son las electrostáticas y las de Van der Waals.

Interacciones electrostáticas: representan las interacciones entre las distribuciones de carga de los distintos átomos de la molécula. Suelen representarse a través de un potencial de Coulomb. Por más información sobre el potencial de Coulomb consultar [79].

Interacciones de van der Waals: representa las interacciones entre átomos que no se pueden considerar como enlaces covalentes o iónicos (ver definición de estos términos en el Anexo F). Estas interacciones pueden ser fuerzas de atracción (conocidas también como dispersión) o repulsión [6]. La ecuación de la figura 5 representa la interacción de Van der Waals, también conocido como potencial de Lennard-Jones.

$$E_{vw} = \sum_{i,j} \left(\frac{A_{ij}}{R_{ij}^{12}} - \frac{C_{ij}}{R_{ij}^{6}} \right)$$

Figura 5 – Energía proveniente de las interacciones de Van der Waals en la macromolécula.

Donde *Aij* y *Cij* son parámetros que definen la interacción de Van der Waals entre átomos y R es la distancia entre los átomos involucrados.

Dentro de la mecánica clásica existen varios métodos para obtener la energía del sistema molecular. Entre ellos se encuentran la Mecánica Molecular (MM) y las Dinámicas Moleculares (MD). La MM son los métodos más sencillos, son computacionalmente económicos y el objetivo de ellos es encontrar la conformación de átomos de mínima energía para un sistema. También son usados para preparar el sistema para realizar diferentes cálculos de dinámica molecular. Sin embargo, MM presenta dos limitaciones. La primera es que el sistema se considera en forma estática. Esto quiere decir que el sistema se visualiza como una "foto" y tiene como consecuencia que no se conoce nada acerca de su flexibilidad, ni tampoco de la movilidad del mismo a lo largo del tiempo. La otra limitación es que no se pueden introducir efectos de la temperatura en el cálculo de la energía. Estas limitaciones son las que dieron paso al desarrollo de las dinámicas moleculares.

MD tiene como objetivo determinar el movimiento interno de la molécula a lo largo del tiempo considerando la temperatura, la presión y otras propiedades macroscópicas del sistema. La información sobre la energía del sistema se procesa para obtener las fuerzas que actúan sobre cada átomo. Luego se determinan las posiciones que estos átomos van tomando a lo largo del tiempo. A esta variación de la posición de los átomos se le conoce como trayectoria [4].

3 Estado del Arte

La idea original del producto a realizar por parte del cliente fue modificándose a medida que se iba entendiendo más en profundidad el problema. Se pasó de realizar un software de visualización molecular a la realización de una herramienta gráfica que sea capaz de transformar una molécula en su representación original a una versión reducida y viceversa (*All-Atom*s ←→ *Coarse-Grain*), generando los archivos necesarios para utilizarse en programas específicos de visualización y simulación de dinámicas moleculares.

En esta sección describiremos una parte de la investigación realizada en las diferentes áreas por las cuales fue pasando el proyecto. Estas áreas abarcan el estado del arte en software de visualización molecular, en particular orientada a describir aplicaciones científicas que soporten la técnica de *Coarse-Grain* (técnica a aplicarse en este proyecto de grado). La otra parte de la investigación se encuentra en el Anexo C, en donde se describen diferentes bibliotecas para la construcción de software de visualización molecular y bibliotecas para el desarrollo de interfaces gráficas, entre otras cosas.

1.6 ¿Qué es una interfaz gráfica?

Debido a que una de las principales características de lo implementado es la interfaz gráfica de usuario, explicaremos qué es y cuál es su objetivo.

Una interfaz gráfica es un medio por el cual un usuario trabaja con una aplicación informática de una manera gráfica, interactuando principalmente con el mouse y el teclado, aunque en la actualidad existen otros dispositivos periféricos que son capaces de interactuar con un programa [7]. Naturalmente, el programa debe estar desarrollado para soportar la interacción con diferentes tipos de dispositivos. Por el contrario existen las interfaces de texto, en las que el usuario trabaja con la aplicación introduciendo texto en forma de órdenes digitadas usando el teclado. El objetivo de las interfaces gráficas es que los programas sean amigables y fáciles de usar para el usuario final, a través de una representación apropiada de los datos y formas de acceder a las diferentes funcionalidades que se intentan representar en la interfaz gráfica [7].

El diseño de una interfaz gráfica se basa en objetos (conocidos como *widgets*) como pueden ser botones, cuadros de texto, menúes, etc, los cuales permiten realizar acciones específicas y visualizar la información que provee el programa en ejecución. Estos objetos, enfocándose en la programación, tienen ciertas propiedades y permiten generar eventos (o señales) que son enviadas cuando se realiza algún tipo de acción sobre ellos (por ejemplo: se pulsa un botón, se introduce texto en un cuadro de texto, etc). Cada lenguaje de programación tiene su forma de definir cómo se ejecutan y envían los eventos o acciones que el usuario introduce.

1.7 Software de visualización molecular

La computación gráfica es una rama de las ciencias de la computación que engloba a todos los aspectos relacionados con la representación de imágenes a través de una computadora [8]. Esta área ha crecido debido a la evolución más que nada del hardware de las computadoras de escritorio y del software, de forma que hoy en día se pueden crear, almacenar y manipular modelos e imágenes de objetos más fácilmente [9].

Una de las áreas que compone la computación gráfica y que está relacionada con este proyecto es la de visualización. Según la Real Academia Española [10], visualizar significa representar mediante imágenes ópticas fenómenos de otro carácter o, siendo más específico, hacer visible una imagen, por ejemplo, en un monitor. Según [11], la visualización en su término más amplio significa "cualquier técnica para la creación de imágenes que representan datos abstractos". En general, el término visualización científica se refiere a cualquier técnica que involucre la transformación de datos en información visual.

El proceso de visualización consiste en filtrar un conjunto de datos para seleccionar un nivel de resolución para cierta área de interés, mapear esos resultados en una forma de representación gráfica y producir una imagen, animación u otro producto visual. El resultado es evaluado, se modifican los parámetros de visualización y se reinicia el proceso. Estos procesos son utilizados por los investigadores al momento de realizar una simulación de cierto modelo de la realidad, en el cual luego de la simulación, obtienen los resultados a través de una o varias imágenes, analizan ese conjunto de datos, modifican algunos parámetros de la simulación si es necesario y se vuelve a simular para obtener nuevos resultados. Así sucesivamente.

La importancia y el crecimiento de la visualización científica se debe a las mejoras en estos últimos años respecto al hardware gráfico así como el desarrollo de diversos programas de visualización o bibliotecas (API's) para el desarrollo de las mismas. La posibilidad de obtener imágenes y generar animaciones para diferentes modelos o simulaciones que se hacen de la realidad, son importantes para el entendimiento y para poder plantear diversas conclusiones de un problema de la realidad que se esté investigando.

Las moléculas, en especial las proteínas y los ácidos nucleicos (ver definición en el Anexo F) son estructuras tridimensionales formadas en algunos casos por pocos átomos, o como en el caso del ADN, por millones de átomos. Esto tiene como consecuencia que para la mayoría de las moléculas, no es posible su representación completa en papel. Antes del desarrollo de la computación, las estructuras más simples, como los aminoácidos, se representaban a través de modelos que consistían en varillas y bolas que representaban los enlaces y átomos respectivamente. Sin embargo, cuando se trataba de visualizar estructuras más complejas como las proteínas, este método resultaba poco práctico, porque se requerían diferentes visualizaciones para estudiar la relación entre sus propiedades estructurales que estos modelos con varillas y bolas no podían ofrecer [13].

Los grandes volúmenes de datos que se generan y manipulan a partir de estudios experimentales o a partir de simulaciones computacionales, obligan a la necesidad de

aplicar técnicas especiales y a la construcción y utilización de software especializado en visualización, sobre todo en visualización tridimensional (3D). Este proyecto de grado se enmarca dentro de la visualización molecular. Desde hace más de 30 años, con la aparición de las computadoras, se han venido desarrollando aplicaciones informáticas capaces de representar las estructuras complejas en 3D. Al principio eran computadoras muy caras y exclusivas para esas tareas. Con el paso de los años el desarrollo de computadoras personales permitió avanzar en la construcción de programas especializados en visualización molecular. Hoy en día estos programas permiten la representación de la estructura de una molécula conociendo, por ejemplo, las coordenadas de los átomos entre otras propiedades.

Existe una cantidad importante de aplicaciones informáticas de visualización molecular desarrolladas por diferentes laboratorios y universidades en el mundo. Algunas son bastante parecidas entre sí, pero cada una posee diferentes características como por ejemplo el lenguaje de programación en el cual se desarrolló, el tipo de aplicación (si es de escritorio o vía web), el tipo de archivos de coordenadas que utiliza y si permite la conversión de diferentes formatos de archivos. Una propiedad importante de este tipo de aplicaciones es la calidad de la representación gráfica de las moléculas y la performance o eficiencia computacional con la que se realiza la misma. En [14] y [15] se pueden encontrar una vasta lista de programas de visualización y modelación molecular.

A continuación describiremos algunas aplicaciones especializadas en visualización molecular que fueron sugeridas por SBM como lo son RASMOL, HyperChem y VMD. En este sentido nosotros analizamos solo algunas aplicaciones investigando qué funcionalidades permitían cada programa y viendo qué posibilidades existían de integrarlo con los requerimientos de nuestro cliente.

1.7.1 RASMOL: RASter MOLecule.

Rasmol fue el primer programa de visualización molecular para computadoras personales desarrollado en 1992 por Roger Sayle. Entre sus características principales están las de permitir la visualización de proteínas, ácidos nucleicos y moléculas pequeñas. Además, es muy usado en la enseñanza para los cursos de química o biología molecular. Aunque el programa se desarrolló originalmente para el sistema operativo UNIX, actualmente puede ser ejecutado en diferentes arquitecturas y sistemas operativos como Windows (RasWin) y MacIntosh (RasMac).

Esta aplicación lee diversos formatos de archivos de coordenadas de los átomos pertenecientes a una molécula, aunque el más utilizado es el de la extensión *pdb* correspondiente al formato *Protein Data Bank* [16] (ver descripción del formato en el Anexo A). La molécula puede verse en la pantalla de diferentes maneras; como una red de todos los enlaces (*wireframe*, *sticks*), como un conjunto de átomos representados como esferas (*spacefill*), como un esqueleto mostrando solamente carbonos alfa (ver descripción en el Anexo F), visualizar como cintas y hélices el esqueleto de los carbonos alfa (también llamado *ribbons*), etc. (ver Anexo B). Otra característica es que distintas partes o sectores de la molécula pueden ser representadas y coloreadas independientemente del resto de la misma, o mostrar la molécula en diferentes representaciones en forma simultánea. La molécula puede manipularse (ser rotada,

trasladada, aplicársele zoom, etc.) interactivamente mediante el mouse o a través de línea de comandos. La imagen obtenida puede ser guardada en diversos formatos de archivo [18] [19].

RasMol tiene ciertas carencias en la calidad de la imagen y de los archivos de salida generados ya que no utiliza la biblioteca OpenGL [32] u otras bibliotecas para el renderizado. Esto no permite producir imágenes que puedan ser utilizadas, por ejemplo en publicaciones científicas, en comparación con otros programas de visualización molecular [20].

1.7.2 HyperChem

HyperChem es un programa comercial desarrollado para diferentes plataformas (Windows o MAC). Permite diseñar, editar, visualizar y modelar estructuras moleculares en tres dimensiones.

En lo que respecta al diseño de estructuras moleculares, la interfaz gráfica es sencilla de usar y se pueden crear desde pequeñas moléculas hasta estructuras más complejas. Por otro lado, se pueden editar diferentes propiedades de la molécula, como enlaces individuales, ángulos, ángulos diedros y cargas atómicas. También es posible realizar una gran variedad de cálculos moleculares y cuánticos. Permite la lectura de diversos formatos estándar de archivos para importar estructuras (PDB, CHM, MOL, MOL2, etc.), incluyendo el reconocimiento de la información de estructura secundaria de los archivos PDB [23].

En esta aplicación se pueden visualizar diferentes representaciones 3D de las moléculas y de sus propiedades, entre las que se destacan la visualización de los potenciales electrostáticos, las densidades de carga, etc. [24]. A partir de la versión 7.5, la representación de las moléculas en la interfaz gráfica se realiza mediante OpenGL [27] [28] y posee además un módulo en el cual se obtienen imágenes de mayor calidad mediante la técnica *ray-tracing* utilizando POV-Ray [21] [22].

1.7.3 VMD – Visual molecular Dynamics

VMD es un programa de visualización molecular *open source* y multiplataforma, que permite visualizar sistemas biológicos como proteínas, ácidos nucleicos, bi-capas lipídicas, entre otras. Además, puede ser utilizado para realizar animaciones y analizar la trayectoria de una estructura atómica, a partir de los resultados de una simulación de dinámica molecular. Este programa fue desarrollado en conjunto por el grupo de Teoría y Biofísica computacional de la Universidad de Illinois y el instituto Beckman [25].

VMD ofrece muchas opciones para la visualización molecular para el usuario. Tiene más de 15 modos de representación molecular, diferentes estilos de coloreado, ya sea por átomo, por residuo, por nombre de molécula, etc. También permite controlar diferentes propiedades del material, como su luz ambiental, difusa, especular, para generar escenas con sombreado de la estructura molecular [25].

En la figura 6 se muestra la interfaz gráfica principal del programa VMD. Puede observarse que tiene tres ventanas: una en donde se despliegan las estructuras tridimensionales (ventana de visualización) a la izquierda de la figura, otra ventana donde se encuentran distintos menús (ventana de menú) en la parte superior hacia la derecha, y por último una tercera ventana en donde se escriben los comandos que se deseen ejecutar (ventana de comandos) que se encuentra en la parte inferior a la derecha.

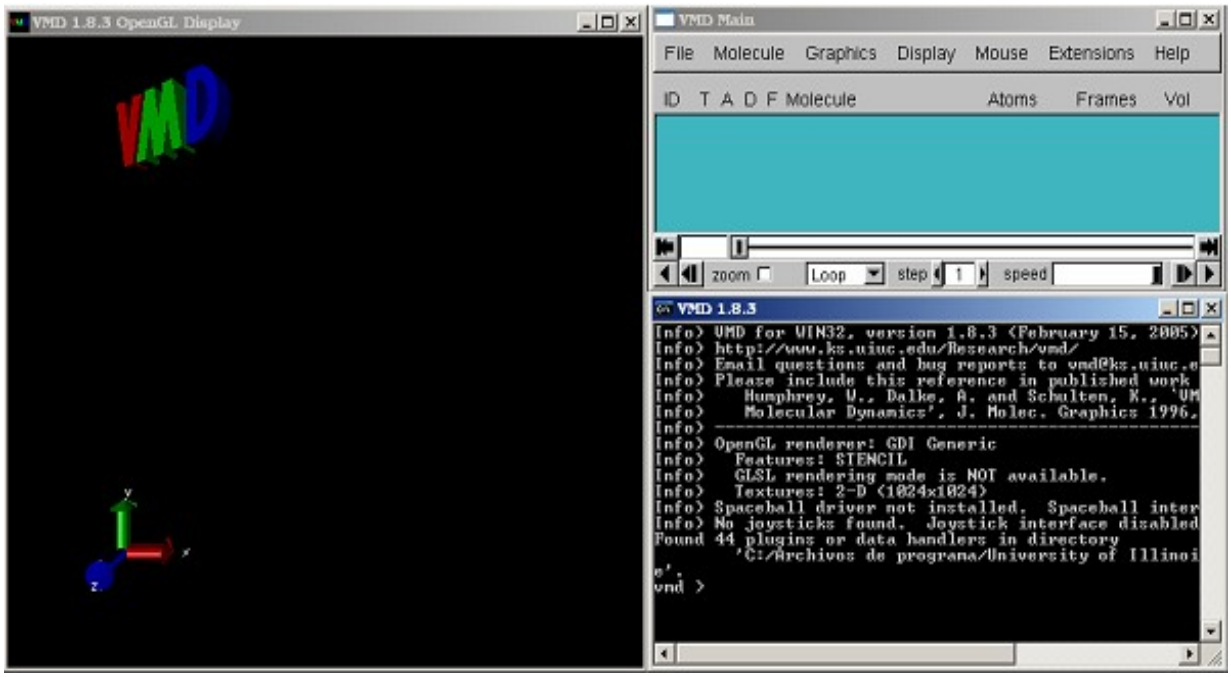


Figura 6 – Interfaz gráfica del programa VMD.

Este software está desarrollado a través del lenguaje C++ siguiendo el paradigma de orientación a objetos. La lógica está dividida en módulos en donde cada uno se encarga de resolver o modelar diferentes problemas. Las funcionalidades de VMD se pueden extender agregando plugins sin necesidad de recompilar el código. Estos pueden desarrollarse a través de los lenguajes C/C++, Python o Tcl/Tk y cada uno tiene su propia forma de integrarse a VMD [27].

Los plugins desarrollados en C/C++, llamados *molfile* [27], actúan como traductores entre los archivos de diferentes formatos que se encuentran en disco y la representación en memoria en VMD. De esta manera, este programa soporta numerosos formatos de archivos de entrada, como archivos de coordenadas y topologías moleculares, archivos de trayectorias dinámicas moleculares, etc.

Por otro lado, contiene una herramienta llamada CG-Tools que da soporte a la realización de la técnica de *Coarse-Grain* (ver Capítulo 2). CG-Tools efectúa la conversión a *Coarse-Grain* en dos modos: basado en la forma de la molécula o basado en residuos [27]. Con el método basado en la forma (*shape-based*) se realiza el pasaje *All-Atom*s a *Coarse-Grain* sin intervención del usuario. Para lograrlo, se basa en algoritmos de redes neuronales tratando de encontrar ciertos patrones en la forma dentro de un residuo. Con el método basado en residuos solamente se define el mapeo entre el superátomo y los átomos originales que se están condensando allí, generándose un archivo con el nombre del nuevo superátomo y su posición (x, y, z) en el espacio en un

formato propio. Tiene como gran desventaja que no se sabe nada de las propiedades de ese superátomo nuevo, tampoco se sabe nada sobre la topología del nuevo residuo con las propiedades físico-química que eso conlleva (conectividades, cargas, masas, etc) [29]. Además en estos dos métodos hay poca interacción con el usuario, lo cual limita la capacidad de modificar ciertas propiedades que pueden querer ser cambiadas.

Por último se puede mencionar que tanto nuestro grupo como SBM hasta el momento de la realización de este informe de proyecto de grado, desconocen la existencia de un programa de visualización molecular que aplique la técnica de *Coarse-Grain* basándose en los requerimientos de nuestro cliente. Si bien existe una herramienta dentro del programa VMD, como se explicó anteriormente, que genera una representación *Coarse-Grain* de la molécula, ésta no se adecúa a la representación que SBM desea generar.

1.8 Conclusiones

La investigación del estado del arte fue más extensa de lo presentado aquí. A medida que se fueron depurando y redondeando los requerimientos, la investigación del estado del arte se fue centrando en temas más específicos.

El primer paso que tomamos en el relevamiento de estado del arte fue la investigación de las características que ofrecían diferentes aplicaciones de visualización molecular. Así, presentamos un pequeño subconjunto de programas, enfocándonos sobretodo en el programa VMD, ya que este es utilizado asiduamente por SBM y posee algunas características que considerábamos interesantes. Esta aplicación ofrece la funcionalidad de convertir una molécula en su representación *All-Atoms* a una representación *Coarse-Grain*. Sin embargo, esta no cumplía con la totalidad de los requerimientos de SBM, por lo cual fue descartada (ver Capítulo 4).

La segunda opción a evaluar fueron las características de un conjunto de bibliotecas, las cuales son utilizadas en diversas áreas de la ciencia por diversos programas que se basan en la visualización de imágenes. Dichas bibliotecas son SDL, OpenDX, VisAd y VTK. Estas poseen un conjunto de funciones gráficas que facilitan la tarea del desarrollador, permitiendo que este se focalice en resolver problemas de más alto nivel (ver Anexo C). Sin embargo en lo que respecta a la visualización, estas funcionalidades no se ajustaban a los requerimientos relevados. Esto, sumado al tiempo necesario para aprender a utilizar estas bibliotecas, fueron motivos suficientes para investigar otras opciones.

Según los requerimientos gráficos relevados (ver Capítulo 4) no fue necesaria la implementación de visualizaciones complejas como las brindadas por las bibliotecas mencionadas anteriormente. Dichos requerimientos se basaron en renderizar los átomos con sus respectivos colores y las conectividades entre ellos. Por lo tanto, decidimos utilizar OpenGL u otra biblioteca similar y que el usuario pueda interactuar con lo dibujado a través de acciones simples, como por ejemplo, rotar, hacer zoom y realizar picking (ver definición en la sección 4.5.4).

Como resultado de lo expuesto anteriormente, nuestra atención se centró en la elección del lenguaje de programación y las bibliotecas que utilizaríamos para desarrollar interfaces gráficas de usuario. Para ello consideramos:

1- La facilidad ofrecida por las bibliotecas para ser utilizadas en el desarrollo de interfaces gráficas de usuario.

2- La performance para realizar el renderizado de moléculas.

3- Que el lenguaje y la biblioteca elegida fueran multiplataforma.

Luego de la comparación realizada, se decidió implementar la aplicación con el lenguaje C++ utilizando Qt [59] como biblioteca multiplataforma para desarrollar la interfaz gráfica de usuario. Para el renderizado de la molécula decidimos utilizar OpenGL directamente, ya que esta biblioteca puede interactuar sin problemas con Qt. En el Anexo C se encuentra una explicación más profunda de las distintas características de los lenguajes de programación seleccionados como candidatos para este proyecto.

Hubo varias alternativas exploradas respecto a cómo desarrollar la aplicación, las cuales se profundizan en el siguiente capítulo.

4 Planteo de la solución

En este capítulo se mencionan aspectos relativos al desarrollo de la aplicación. Esto incluye a la etapa de análisis, diseño e implementación del producto final.

En el apartado referido a la etapa de análisis se especifican los requerimientos iniciales y las alternativas que surgieron para el desarrollo de la aplicación. Seguidamente se presenta el diseño en donde se describe la estructura general de la molécula, incluyendo sus propiedades y la relación entre los diferentes conceptos que la componen, el modelo de dominio generado para la molécula, la arquitectura definida para el sistema y otras decisiones de diseño. Estas decisiones contemplan 2 de los 3 aspectos fundamentales de la aplicación, como lo son el diseño de la interfaz de usuario y el manejo de archivos. Los otros aspectos son: el algoritmo de generación de una molécula *Coarse Grain* a partir de una molécula *All-Atoms* y su inverso (reconstrucción de una molécula *All-Atoms* a partir de una molécula *Coarse Grain*), que se explicarán en el capítulo 5.

Respecto a la etapa de implementación se enfatiza en los aspectos gráficos desarrollados, las características de la interfaz de usuario y el manejo de los archivos que describen a la molécula.

1.9 Evolución del proyecto

Desde el inicio del proyecto se realizaron numerosas reuniones junto a SBM, en donde se planteó el problema a resolver tratando de entender la realidad en la que se enmarcaba el proyecto. También fueron definidos los requerimientos necesarios para la construcción de la aplicación en forma detallada debido a la complejidad que involucra trabajar para otra disciplina científica.

Partiendo de los requerimientos y de la investigación del estado del arte en lo que se refiere a software de visualización, surgieron diversas estrategias para el desarrollo de la aplicación, las cuales se describen en la sección 4.1.2.

1.9.1 Requerimientos funcionales

A continuación se lista el conjunto de requerimientos funcionales y no funcionales que fueron establecidos originalmente para la herramienta a desarrollada.

1- Generar archivos de topología. Esto significa que se generan archivos que incluyen diferentes propiedades relacionadas con la molécula como lo son: conectividad, carga, masa, ángulos de equilibrio, distancia de equilibrio y ángulos de torsión (diedros), entre otras. Estos conceptos se aclararán más adelante.

2- Dada una molécula, definir el pasaje *All-Atom* → *Coarse-Grain*. Con la posibilidad de crear un archivo que permita establecer la relación entre los

átomos originales y los átomos *Coarse-Grain* (también llamados superátomos), así como también las relaciones de las conectividades entre superátomos. El objetivo del archivo a definir radica en permitir el proceso inverso (*Coarse-Grain* → *All-Atom*), es decir, volver a armar la representación *All-Atom* a partir de representaciones *Coarse-Grain*.

3- La interfaz de usuario debe permitir una elección arbitraria de agrupamiento de átomos para generar una molécula *Coarse-Grain* de un modo flexible. Alternativamente pueden seleccionarse los átomos directamente en el área donde están dibujados con el objetivo de consultar sus propiedades.

4- Las posiciones definidas sobre los SATs se deben poder definir en términos de los átomos que contiene. Se puede elegir la posición del SAT en la posición de uno de los átomos que lo conforman, en el centro de masa o en el centro geométrico de un conjunto de los átomos que lo conforman.

5- Representar visualmente en la interfaz gráfica a los átomos y a los residuos.

6- La aplicación debe trabajar con archivos en el formato de Amber y Gromacs [35] tanto en su lectura como escritura, con la posibilidad de editar todas sus propiedades.

7- Incluir diferentes visualizaciones extras para un post-procesamiento de las simulaciones, obteniendo datos o información sobre diferentes propiedades de los residuos.

8- Las propiedades de la molécula, residuos y átomos deben poder ser modificados a través de la interfaz gráfica.

9- La visualización debe ser de la mejor calidad gráfica posible.

10- La interfaz debe ser intuitiva para el usuario final.

11- No existen limitantes sobre el lenguaje de programación a utilizar.

12- La aplicación debe poder ejecutarse por lo menos en la plataforma Linux con posibilidad de extenderla al sistema operativo Microsoft Windows.

1.9.2 Alternativas exploradas

Basándose en los requerimientos y en la investigación del estado del arte en software de visualización molecular, se profundizó en el estudio de las posibilidades que brinda la aplicación VMD (Visual Molecular Dynamics), sugerido y utilizado asiduamente por SBM. VMD es una aplicación multiplataforma que tiene más de 10 años de evolución en lo que se refiere al desarrollo gráfico. Esta provee más de 15 representaciones visuales en 3D para una molécula y muchas visualizaciones en 2D que son utilizadas para un post-procesamiento de la información. Además posee otras funcionalidades específicas en bioquímica.

Como resultado de estas investigaciones y considerando que este programa es de código abierto, existe soporte y documentación online y además permite agregar plugins desarrollados por el usuario, surgieron tres posibles alternativas a desarrollar. Estas alternativas fueron: la modificación del código de VMD, el desarrollo de una herramienta independiente en C++ que se comunicara con el VMD y, por último, dibujar en la propia ventana del VMD a través de comandos implementados en el lenguaje Tcl.

A continuación se describe la investigación realizada en cada una de estas alternativas, investigación que a pesar de la gran cantidad de tiempo que demandó debió llevarse a cabo con profundidad por ser determinante para el éxito del proyecto.

1.9.2.1 Modificación del código de VMD

El objetivo de esta alternativa fue el de extender las posibilidades del VMD, modificando directamente su código fuente. Se estudió con dicho propósito la implementación existente basándose en tres aspectos principales:

1- Cómo interactuar con la interfaz de visualización de moléculas.
2- Cómo incluir algoritmos de lectura/escritura de archivos.
3- Cómo acceder a la representación interna de las moléculas en memoria.

Esta alternativa nos planteó un gran problema, que fue el estudio de código ajeno extremadamente complejo siendo inadecuada (por no decir inexistente) la documentación disponible sobre el mismo. Luego de un período de exhausta investigación vimos que esta alternativa insumiría demasiado tiempo y sería muy compleja de llevar a cabo.

1.9.2.2 Comunicación con C++

La estrategia presentada a continuación relaciona un plugin de VMD que actúa como intermediario con una herramienta independiente desarrollada en C++.

Esta herramienta posee una interfaz gráfica de usuario la cual, a partir de un residuo de una molécula, permite crear y establecer las diferentes propiedades de un SAT. A partir de estos SATs se generan nuevos residuos *Coarse-Grain*. Este proceso se repite para todos los residuos que se desee transformar en la molécula.

Finalmente se generan un conjunto de archivos, los cuales contienen la información de la nueva molécula *Coarse-Grain*. Por otro lado existe un plugin de VMD desarrollado en Tcl, el cual recibe la ubicación de dichos archivos, con el objetivo de poder visualizar en VMD la molécula *Coarse-Grain* en forma global y no parcial, como en el caso de la herramienta independiente de C++. Para llevar a cabo este proceso, el plugin invoca a ciertas primitivas de VMD para cargar la molécula en el *Canvas* del mismo.

Esta estrategia tiene problemas al interactuar simultáneamente el *Canvas* de VMD, con el *Canvas* del programa desarrollado en C++. En ninguna de las dos ventanas era

posible visualizar los átomos de un residuo. Nos pudimos contactar con dos desarrolladores, Axel Kohlmeyer y John Stone, los cuales nos sugirieron que no era trivial dibujar en el *Canvas* de VMD en simultáneo con otra ventana que utilizara también OpenGL. Ellos explicaron que al tener dos programas en ejecución que utilizan OpenGL bajo un mismo thread (VMD y el programa en C++), se deben salvar los "contextos" de cada uno de los programas. En este sentido, contexto se le denomina a cada uno de los estados de la máquina de estados de OpenGL.

En las aplicaciones que dibujan en múltiples ventanas ejecutándose bajo un mismo thread, el contexto de OpenGL debe salvarse y restaurarse para cada ventana al momento de dibujar en la misma. Muchas bibliotecas realizan este manejo como GTK, GLUT y SDL, entre otras. En cambio VMD utiliza su propio manejo de contexto de OpenGL y por lo tanto, ninguna de estas bibliotecas son capaces de guardar y restaurar el estado interno usado por VMD. Debe ser realizado explícitamente por el programador. Según John Stone, el sólo hecho de implementar un manejador de contexto que independice el contexto utilizado por el VMD del utilizado por el plugin, constituye un complejo proyecto en sí mismo.

John Stone [32] es uno de los principales desarrolladores de VMD desde el año 1998 hasta la fecha. Fue el creador de la nueva GUI de VMD y de mejorar aspectos visuales desarrollándolo con primitivas y shaders de OpenGL aprovechándose de la arquitectura paralela que brinda GPU [31]. Hoy en día es el programador encargado del mantenimiento y de realizar mejoras a VMD. Por su parte, Axel Kohlmeyer [33] es director asociado y profesor investigador asociado en el Centro de Modelación Molecular perteneciente al Departamento de Química de la Universidad de Pennsylvania, Estados Unidos.

1.9.2.3 Dibujo en la propia ventana de VMD a través de comandos Tcl

Esta alternativa considera la realización de la interfaz de creación de átomos *Coarse-Grain* en la propia ventana principal de VMD, aprovechando el desarrollo gráfico de este programa. Para ello, se utilizan las primitivas que provee VMD para desarrollar plugins en el lenguaje de scripting Tcl/Tk. De esta forma, se dibuja solamente el residuo *All-Atom* a ser transformado. El plugin muestra una GUI para que el usuario pueda editar el residuo que se encuentra cargado en la propia ventana del VMD y al mismo tiempo definir SATs y residuos *Coarse-Grain*. Este proceso se repite para todos los residuos que pertenecen a la molécula.

El algoritmo de conversión *Coarse-Grain* → *All-Atom* se lleva a cabo dentro del plugin para luego de determinada la molécula resultante, renderizarla en el *Canvas* de VMD.

Si bien es posible la implementación de esta estrategia, existen ciertos problemas complejos a destacar. El principal de ellos es que el algoritmo *All-Atom* → *Coarse-Grain* debería ser implementado con Tcl/Tk, lenguaje poco adecuado cuando uno de los requerimientos es la performance, como en el caso del algoritmo en cuestión. Por otra parte, la estrategia requiere mucha más investigación del VMD y más conocimiento del lenguaje Tcl/Tk, hecho no viable debido a no tener la certeza plena de poder conseguir los objetivos.

1.9.3 Estrategia seleccionada

La estrategia de modificar el código de VMD fue descartada por los motivos expuestos allí. Entre las alternativas restantes, la opción de dibujar en el *Canvas* de VMD interactuando con un plugin desarrollado en Tcl y la aplicación independiente desarrollada en C++, optamos por la segunda ya que todos los integrantes del grupo poseemos los conocimientos sobre las tecnologías necesaria para desarrollar la aplicación. Los dos pilares de esta estrategia consisten en poseer conocimientos del lenguaje C++ y de la biblioteca OpenGL. En nuestro grupo existen conocimientos suficientes en estas dos áreas hecho que fue determinante al decidirnos por esta estrategia. La dificultad presentada en esta estrategia es el dibujado en ambos *Canvas* (el de VMD y el de la aplicación C++) simultáneamente. Sin embargo, este problema no impide poder iniciar el desarrollo de la aplicación (en C++), ya que se puede implementar este programa y posteriormente resolver el problema del dibujado en ambos *Canvas*.

Por su parte, la alternativa del plugin Tcl/Tk requería una investigación más profunda, existiendo la posibilidad de encontrarse con nuevos problemas en su implementación. Además todos los integrantes deberíamos aprender ambos lenguajes Tcl y Tk, esto hubiese generado una curva de aprendizaje mucho mayor en comparación con la otra alternativa.

A la herramienta desarrollada la denominamos "*Coarse Grain Manager*" (CG Manager). A continuación, en la figura 7, se muestra un diagrama en donde participan varios programas del framework AMBER (ver Anexo D), el cual SBM utiliza para la generación de trayectorias moleculares como consecuencia de la aplicación de métodos de dinámicas moleculares. Dentro de esta secuencia se considera el uso de la aplicación *CG Manager*.

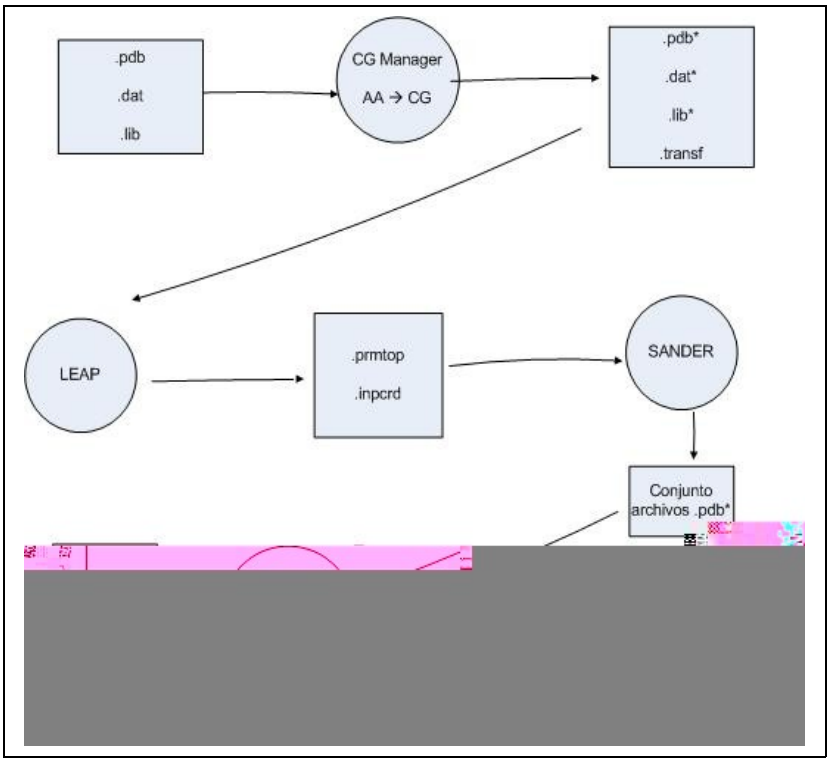


Figura 7 – Secuencia de uso de los diferentes programas de AMBER junto con la herramienta desarrollada en el proyecto de grado.

Como se observa en la figura 7, la herramienta *CG Manager* es utilizada al principio y al final de esa secuencia. En el primer caso, se define una molécula en la representación *Coarse-Grain* y al final se realiza la acción inversa para todas las instancias de la molécula generadas en la simulación, es decir, se convierte cada instancia en su representación *All-Atom*.

A continuación se detallan los pasos que se realizan en la generación de archivos de dinámicas moleculares utilizando AMBER:

1- Se eligen tres archivos que definen a una molécula en particular. Estos deben tener el formato *pdb*, *dat* y *lib* (por más información sobre los formatos ver Anexo A). Estos archivos representan la molécula y sus propiedades en su forma *All-Atom*.

2- Se utiliza la herramienta *CG Manager* a partir de los tres archivos indicados en el paso 1, para generar un modelo *Coarse-Grain* a partir de un modelo *All-Atom*.

3- Se generan los tres archivos con la información de la molécula "condensada" y con un archivo conteniendo información de las transformaciones definidas en la molécula "condensada", con el objetivo de que la simulación del paso 6 sea computacionalmente más eficiente.

4- Estos archivos (*pdb**, *dat** y *lib**) los utiliza un programa llamado LEAP para luego convertirlos a un formato de archivo que acepte el programa SANDER.

5- A partir de la ejecución de LEAP se generan dos archivos (*.prmtop* y *.inpcrd*).

6- Luego del paso 5, los archivos contienen la información de topología de la molécula en otro formato, los cuales son utilizados como parámetro de entrada a otro programa llamado SANDER, que realiza la simulación de dinámica molecular, calculando la trayectoria individual de cada átomo (en este caso Súper Átomos).

7- A cada uno de los archivos *pdb** se les aplica el algoritmo de conversión inversa considerando el archivo de formato *transf* convirtiendo la molécula que se encuentra en la representación *Coarse-Grain* hacia la representación *All-Atom*, obteniendo nuevamente un conjunto de archivos en formato *pdb*. De esta manera se genera la molécula con las posiciones de los átomos originales, considerando un margen de error.

8- Fin del ciclo funcional.

El conjunto de archivos finales en formato *pdb* pueden utilizarse directamente en el programa VMD. Este posee la funcionalidad de agrupar estos archivos desplegándolos en secuencia a razón de un archivo *pdb* por frame, formando finalmente una animación de algunos segundos.

1.9.4 Conclusiones

De acuerdo a las razones explicadas anteriormente, el grupo decidió desarrollar la herramienta *CG Manager* en el lenguaje de programación C++ ya que este permite integrar en forma directa el uso de la biblioteca de renderizado OpenGL. Además, el lenguaje y la biblioteca son conocidos y utilizados por los integrantes del grupo. En lo que respecta a la biblioteca para el desarrollo de la *GUI*, se optó por Qt. Esta biblioteca es multiplataforma y el manejo de eventos de los diferentes *widgets* es bastante directo y de uso sencillo (mecanismo Signal/Slot [64]) (por más información acerca del estudio de las diferentes bibliotecas de interfaces gráficas ver Anexo C).

1.10 Diseño

En este capítulo se presenta el diseño en el cual se basa la implementación de la aplicación. Se define la estructura básica de una molécula con sus propiedades y las relaciones existentes entre ellas, el modelo de dominio generado a partir de la realidad del problema, la arquitectura del sistema con los módulos definidos en ella y por último se menciona parte de la implementación, enfocándose en el diseño de la interfaz gráfica de usuario, el manejo de archivos y una descripción en alto nivel de la visualización gráfica que posee la aplicación desarrollada.

1.10.1 Estructura de una molécula

A continuación se explica la estructura de una molécula y sus propiedades, enfocándose en la etapa de análisis del desarrollo de la aplicación. Esta descripción es la base para la realización del modelo de dominio que se presenta más adelante. Las fuentes de información para poder llevar a cabo esto fueron los archivos con el formato *pdb*, *lib*, *dat*, ver Anexo A, y el conocimiento transmitido por parte de los investigadores de SBM. La comprensión de la realidad a modelar, en este caso, la estructura de la molécula junto a sus propiedades físico-químicas tomando como base los tipos de archivos que la aplicación debía soportar, fue una tarea que requirió de numerosas reuniones con el grupo SBM y de mucha investigación en la web. Ya que la documentación existente, respecto de los distintos formatos de archivos, es muy escasa.

Una molécula está compuesta por un conjunto de átomos genéricos (llamados átomos canónicos) que se agrupan en residuos genéricos (denominados residuos canónicos) y por un conjunto de propiedades inherentes a la molécula como los son los campos de fuerza. Entre ellos se destacan los relativos a los enlaces entre átomos, ángulos y diedros (por más información sobre los campos de fuerza ver Capítulo 2).

Los átomos canónicos se identifican por su nombre, el cual es único dentro del residuo en el que está definido. Todas las instancias de un átomo canónico en la molécula, comparten las propiedades definidas en el átomo canónico, exceptuando la posición en el espacio, la cual es diferente para cada instancia. Aunque el átomo canónico también tiene su propia posición.

Por otro lado, los átomos canónicos están asociados a un tipo de átomo, el cual posee ciertas propiedades, como por ejemplo la masa, los potenciales de Lennard-Jones, etc. Para algunos tipos de átomos, los potenciales están definidos a través de una relación de equivalencia con otro tipo de átomo.

Por su parte los residuos canónicos también son identificados por su nombre, el cual es único dentro de la molécula. Estos residuos, contienen un conjunto de átomos canónicos y en él se definen las conectividades entre estos átomos, entre otras propiedades.

Las instancias de un residuo canónico dentro de la molécula poseen un número entero que los identifica y cada una de estas instancias agrupa un conjunto de instancias de átomos canónicos.

La molécula, es una secuencia de residuos conectados. Esta conexión entre residuos se hace efectiva mediante dos átomos denominados cabeza y cola (*head* y *tail*). El átomo cabeza de un residuo está enlazado con el átomo cola del residuo adyacente y así sucesivamente en toda la molécula. El primer y el último residuo pertenecientes a la molécula, solamente poseen cola y cabeza respectivamente. En el caso particular de que una molécula esté formada por un solo residuo, éste no posee átomo cabeza ni átomo cola.

1.10.2 Modelo de dominio

De acuerdo a la realidad planteada en la sección anterior, en la figura 8 se ilustra el modelo de dominio que representa a la molécula. Se pueden observar la relación existente entre los diferentes conceptos mencionados anteriormente, así como la representación de algunas propiedades relevantes dentro de una molécula, como lo son algunos de los parámetros de campos de fuerza. En este caso los relacionados a los enlaces entre átomos, ángulos de enlace y ángulos diedros.

Asimismo, se presentan los conceptos de transformación y de superátomo. Si bien estos conceptos no pertenecen a la definición de una molécula, fueron agregados para obtener una comprensión total de la realidad que el programa modela. Este modelo es una conjunción de la información que es representada en los archivos de formato *dat*, *lib*, *pdb* y el formato creado para esta aplicación, *transf*.

Según el modelo, una transformación es un residuo canónico y está asociado al residuo original para el cual éste es definido (ver asociación etiquetada “can_residue” de la figura 8). Por su parte un superátomo es un átomo canónico, el cual tiene asociado un conjunto de átomos canónicos que representan los átomos que el superátomo condensa (ver asociación etiquetada “atContained” de la figura 8). A su vez, se identifican los átomos canónicos participantes en la definición de la posición del superátomo. Esta posición puede ser el centro de masa o el centro geométrico de un conjunto de átomos o la posición de uno de los átomos condensados.

Toda la información que se va generando o modificando mientras la aplicación se ejecuta, puede ser persistida en los diferentes archivos. A saber: la información respecto a los átomos canónicos y residuos canónicos se distribuye en el *dat* y *lib*; la información respecto a las instancias de residuos e instancias de átomos se persiste en el archivo *pdb*, información referida a los campos de fuerza en el *dat* y por último, la información respecto a las transformaciones y los superátomos, una gran parte se almacena en el *dat* y *lib* (como los residuos y átomos) y otra parte se guarda en el archivo de transformaciones (archivo *.transf*).

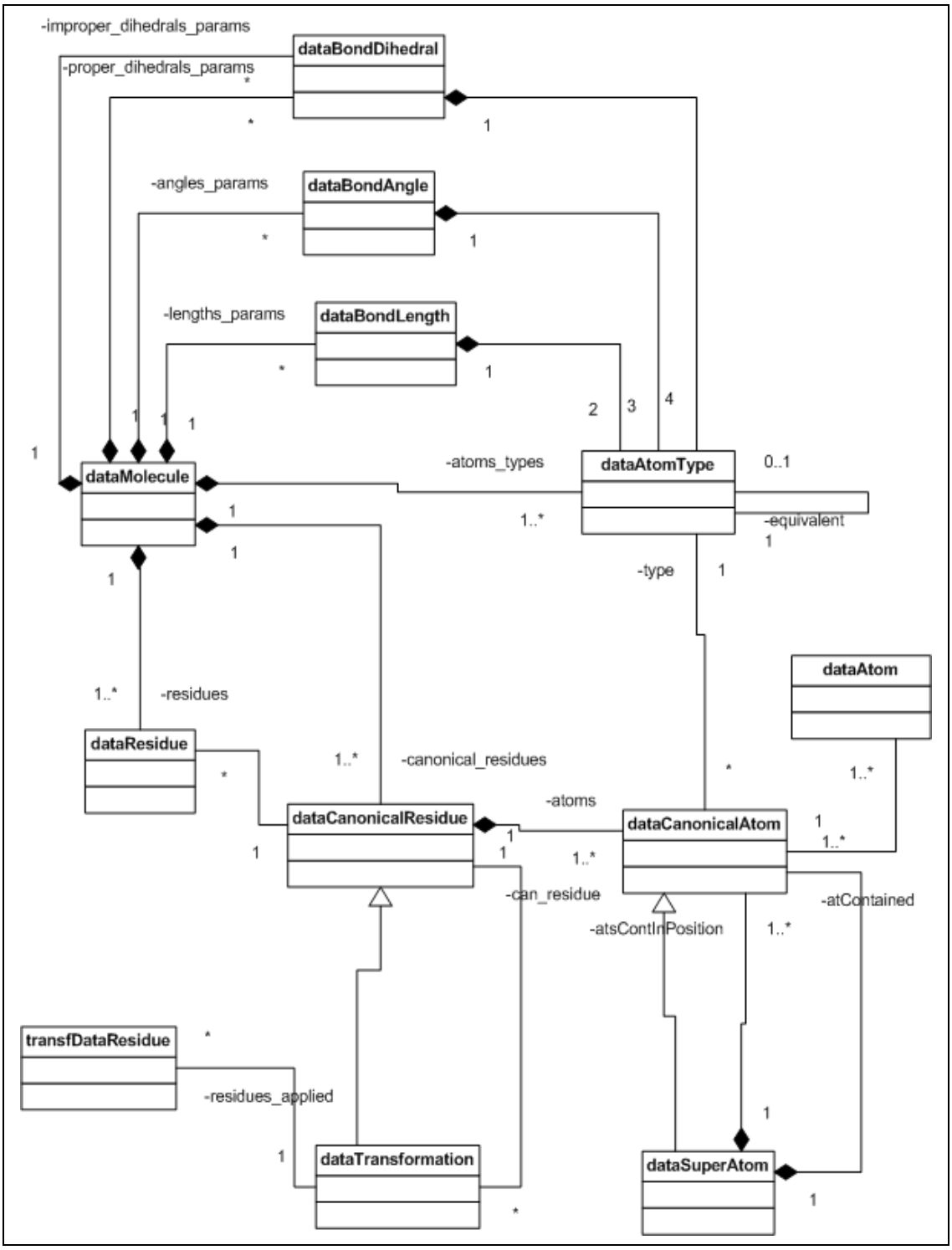


Figura 8 – Modelo de dominio que representa a la molécula.

A continuación se realizará una breve descripción de las clases especificadas en el modelo de dominio.

- dataMolecule: Representa al concepto molécula. A partir de este se relacionan los demás conceptos del modelo de dominio.

- dataBondDihedral: Campo de fuerza de diedros.

- dataBondAngle: Campo de fuerza de los ángulos.
- dataBondLength: Campo de fuerza de los enlaces entre átomos.
- dataAtomType: Representa el concepto de los tipos de átomos.
- dataCanonicalResidue: Residuos canónicos que están presentes en la molécula.
- dataCanonicalAtom: Átomos canónicos que están presentes dentro de un residuo canónico.
- dataResidue: Instancia de un residuo canónico.
- dataAtom: Instancia de un átomo canónico.
- dataTransformation: Representa al concepto de transformación. Como se puede apreciar en la figura 8, una transformación también es un residuo canónico.
- dataSuperAtom: Representa al concepto de superátomo. Como se puede apreciar en la figura 8, una transformación también es un átomo canónico.
- transfDataResidue: Copia de una instancia de residuo conteniendo información extra para el manejo de las transformaciones.

1.10.3 Arquitectura

Se plantea una arquitectura basada en componentes en donde cada uno de ellos encapsula y resuelve diferentes funcionalidades del sistema. Los componentes que conforman la arquitectura son: *GUI*, *GuiHandler*, *Molecule* y *FileHandler*. En la figura 9 se presenta la arquitectura definida para el sistema, mostrando los componentes principales y la interacción entre ellos.

Este diseño de arquitectura contempla dos aspectos fundamentales de la aplicación. Se contempla el diseño de la interfaz gráfica, en donde además de poder visualizar los residuos y átomos de la molécula, se pueden editar las diferentes propiedades relacionadas a estos; se contempla el manejo de los diferentes archivos que soporta el programa y por otro lado se encapsula la representación de la molécula en un solo componente. De esta manera, se distribuyen diferentes aspectos de la aplicación en los distintos componentes del software construido.

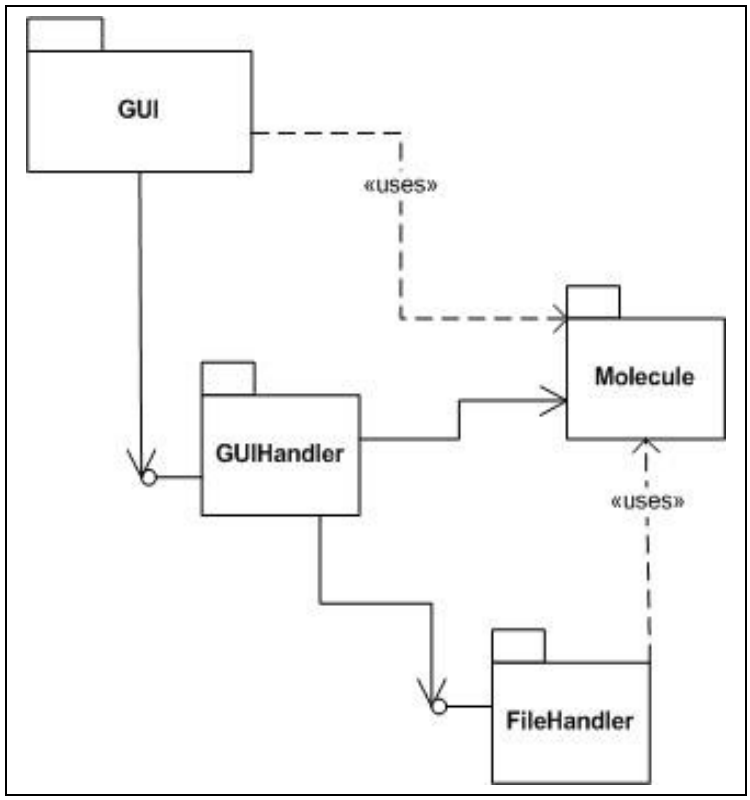


Figura 9 – Diagrama de la arquitectura del sistema.

El componente *GUI* tiene como objetivo el manejo de la interfaz gráfica de usuario. Está formada por dos subcomponentes, donde cada uno procesa las acciones que realiza el usuario y en caso de que la acción corresponda a otro módulo, se delega el procesamiento del evento a donde corresponda. Uno de los subcomponentes es el llamado *Canvas*. Este se encarga de presentar gráficamente el residuo y los átomos seleccionados. Como se mencionó en capítulos previos, el desarrollo de la interfaz gráfica se basa en la biblioteca Qt y la visualización de los residuos se desarrolló mediante la biblioteca OpenGL.

Por su parte, *GuiHandler* actúa de intermediario entre la molécula cargada en memoria y la interfaz de usuario. Mantiene el estado de las diferentes propiedades que se visualizan en la interfaz gráfica de usuario. Dependiendo de las acciones que el usuario ejecuta, se actualizan la información de la molécula cargada, o se envía información hacia la interfaz de usuario.

Por otro lado se encuentra el componente *Molecule*, que contiene toda la información de la molécula, residuos y átomos con sus respectivas propiedades en memoria.

Por último, se encuentra el módulo *FileHandler*, es el punto de entrada para obtener y generar los datos iniciales y finales de la molécula respectivamente. El diseño de este componente comprende la lectura y escritura de todos los archivos que contienen la información de la molécula y sus respectivas propiedades (por más información sobre el diseño relacionado al manejo de archivos, ver sección 4.6).

Un diagrama extendido de la arquitectura se encuentra en el Anexo G.

1.11 Interfaz Gráfica de Usuario

Un aspecto importante del proyecto fue el entender conceptualmente el problema a desarrollar, fundamentalmente los conceptos y propiedades que involucran a la molécula, residuos y átomos, y la relación existente entre ellos.

Los principales aspectos a considerar son: molécula, residuos, átomos, tipos de átomos*, bond lengths, angle bends* y *dihedrals*. Para el presente sistema se construyó una interfaz gráfica de usuario que agrupa y organiza estos conceptos y simultáneamente mantiene la coherencia y relación de las diferentes propiedades.

Esta interfaz gráfica permite la creación de representaciones moleculares en modalidad *Coarse-Grain* a partir de su representación *All-Atom*. A su vez, permite realizar la transformación inversa, esto es la creación de la molécula *All-Atom* a partir de su representación *Coarse-Grain*. Esto último es posible siempre y cuando se haya realizado la transformación directa a partir del presente sistema, (a partir de ahora CG-Manager) y se cuente con los archivos generados por CG-Manager a partir de dicha transformación. Por otro lado se pueden modificar diferentes propiedades inherentes a la molécula sin importar el tipo de representación de la misma (*All-Atom* o *Coarse-Grain*).

Existe una relación directa entre el modelo de dominio presentado en la sección 4.2 y el diseño de la GUI, en donde a través de diferentes elementos o widgets se pueden visualizar y editar los parámetros correspondientes tanto a la molécula en general, como a los residuos y átomos que la componen.

Al ejecutar la aplicación se despliega la ventana principal, ver figura 10. En esta ventana se pueden apreciar tres sectores. En el centro, se encuentra la región de visualización de residuos denominada *Canvas*. En la parte inmediatamente inferior al *Canvas* se encuentra una región para el ingreso y visualización de los campos de fuerza vinculados a la molécula, la cual denominamos *Sector inferior*. Por último, a la derecha del *Canvas* se encuentra lo que denominaremos *Sector derecho* que contiene información inherente a la molécula, residuos y átomos.

Figura 10 – Residuo seleccionado y desplegado en el *Canvas*.

1.11.1 Canvas

El *Canvas* es una área de dibujado en 3D en la que a través de la biblioteca OpenGL se implementaron algunos aspectos gráficos como lo son la cámara, el *picking*, la representación geométrica de los residuos y la iluminación de la escena. Estos aspectos se detallan en la sección 4.5.

Al seleccionar un residuo perteneciente a la molécula se visualizan los átomos contenidos en él, así como las conectividades existentes entre estos átomos. Además es posible identificar el tipo de átomo a través del color por el cual está renderizado. Los colores identifican a los diferentes elementos de la tabla periódica y están organizados según el esquema de colores CPK (ver Anexo E).

Por otro lado, al momento de definir una transformación, el usuario visualiza los SATs que se están creando. Puede establecer algunas propiedades tanto de la transformación como de los SATs directamente en el *Canvas* o haciendo click en el cuadro de diálogo de definición de transformaciones. Las diferentes funcionalidades y los pasos a seguir se detallan en el manual de usuario. El *Canvas* posee tres funcionalidades básicas a realizar sobre el residuo dibujado: rotarlo, hacer zoom in-out y seleccionar uno o varios átomos.

1.11.2 Sector Inferior

En este sector se encuentran algunas de las propiedades generales de la molécula, que se especifican en el archivo *dat* (ver Anexo A). Estas propiedades están organizadas en tres pestañas diferentes y cada una representa a uno de los campos de fuerza, los cuales involucran la relación existentes entre 2 o más átomos como lo son los *Bond Length*, *Angle Bend* y *Dihedral* (ver Capítulo 2). En cada una de estas pestañas se pueden visualizar y modificar cada uno de los parámetros correspondientes a cada campo de fuerza para cada uno de los ítems desplegados en las respectivas listas.

Como se muestra en la figura 10, dentro del sector inferior existe un conjunto de botones que permiten realizar diversas acciones: guardar la molécula y la transformación generada hasta ese momento ("Save Molecule"), definir la relación entre transformaciones e instancias de residuos ("Set residue transf"), generar el archivo *pdb* ("Generate PDB") y crear nuevos diedros ("New Dihedral").

A continuación se especificarán en detalle dos funcionalidades que son importantes en el CG-Manager. Estas son las correspondientes a los botones "Set residue transf" y "Generate PDB".

El botón cuyo nombre es "Set residue transf" permite definir una relación unívoca entre una transformación que debe ser aplicada a una instancia de un residuo: al presionar este botón se despliega un diálogo como el ilustrado en la figura 11, en el cual se muestran tres listas en el siguiente orden: residuos canónicos, transformaciones definidas para un residuo canónico y la lista de residuos específicos. A través de estas listas se establece una relación funcional entre una transformación y un residuo específico. A su vez, tanto la transformación como el residuo específico están relacionados con el residuo canónico seleccionado en la lista "Canonical Residues".

Figura 11 – Diálogo para establecer transformaciones a aplicar sobre residuos específicos.

Generar el archivo *pdb*: a través de esta acción se procede a aplicar la transformación definida por el usuario a cada uno de los residuos específicos que están indicados a través de la relación funcional mostrada por el diálogo de la operación "Set residue transf". Se aplica el algoritmo de transformación *All-Atom* → *Coarse-Grain* que se describe en el capítulo 5. Al efectuarse esta operación, el usuario indica el nombre del archivo en formato *pdb* que contendrá a la nueva molécula *Coarse-Grain*.

1.11.3 Sector Derecho

En este sector de la interfaz gráfica de usuario se despliegan las diferentes propiedades correspondientes a la molécula, residuos, átomos (incluye también a los súper átomos) y de los tipos de átomos. Están agrupados bajo tres etiquetas que pueden contraer y expandir su información. Estas etiquetas son: *Molecule*, *Residue* y *Atom*. En la figura 12 se puede ver a estas etiquetas contraídas.

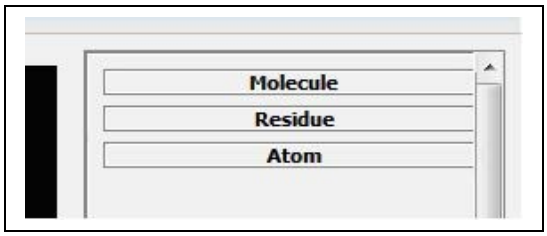


Figura 12 – Vista de las pestañas contraídas del sector derecho de la ventana principal.

A continuación se describe brevemente cada una de las etiquetas mencionadas que se ilustran en la figura 12.

La etiqueta *Molecule* agrupa diferente información general relacionada a la molécula cargada en la interfaz de usuario. En ella se muestran los residuos canónicos de la molécula, se visualizan todos los valores potenciales relacionados a los diferentes tipos de átomos, también es posible visualizar los átomos canónicos pertenecientes a un residuo canónico en particular, las transformaciones definidas para un residuo, así como también los SATs pertenecientes a la transformación que esté seleccionada. Además se permite crear y eliminar transformaciones.

La etiqueta *Residue* agrupa esencialmente información contenida en el archivo *lib* y que está relacionada con los residuos canónicos. Cuando se selecciona un residuo canónico se cargan los datos pertenecientes a este residuo en los diferentes *widgets* de esta etiqueta. La información detallada de esta etiqueta se puede consultar en el manual de usuario.

Por su parte, dentro de la etiqueta *Atom* se visualiza y se puede editar información relacionada al átomo canónico seleccionado o a un superátomo, ya sea seleccionándolo a través de la etiqueta Molecule o pickeando directamente sobre el átomo en el *Canvas*. La aplicación detecta y discrimina si el átomo seleccionado es un átomo canónico o se trata de un SAT.

1.11.4 Diálogo de transformaciones

Para definir transformaciones, se debe seleccionar un residuo canónico y luego presionar el botón *Define New Transformation*. En ese momento se despliega un cuadro de diálogo para la creación de la transformación, ver figura 13. Este cuadro de diálogo contiene un campo de texto para establecer el nombre para la transformación, el cual será el nombre del nuevo residuo *Coarse-Grain* que se está creando. Además posee dos listas: en la lista de la izquierda se presentan todos los átomos aún no condensados

pertenecientes al residuo (átomos originales) y en la lista de la derecha se encuentran los superátomos ya creados (átomos del nuevo residuo).

A la derecha del diálogo se encuentra un conjunto de botones que poseen ciertas funcionalidades útiles al momento de definir una transformación. Estos son para crear nuevos SATs, eliminar alguno ya definido y finalizar la definición de la transformación entre otros. Más información sobre la funcionalidad de estos botones se encuentra en el manual de usuario y en el Anexo G.

Una característica de este diálogo, es que mientras se definen SATs, la visualización del *Canvas* está coordinada con las acciones que el usuario realiza.

Figura 13 – Diálogo de definición de transformaciones luego de crear todos los Superátomos del nuevo residuo *Coarse-Grain*.

1.12 Implementación – Coordinación Canvas-GUI

En esta sección se describe brevemente la implementación desarrollada en lo que respecta a la interacción existente entre el *Canvas* y los restantes elementos de la interfaz gráfica de usuario.

Una tarea poco sencilla fue la sincronización entre las acciones que se realizan en la interfaz gráfica y lo que se visualiza en el *Canvas*. En este último, se despliegan residuos canónicos en forma completa, superátomos que se están creando junto a los átomos contenidos en el residuo, las conectividades, etc.

La sincronización se aplica para un subconjunto de operaciones, las relacionadas con la modificación de propiedades de los átomos y todo lo relativo a la creación de

transformaciones, esto último incluye la creación de superátomos, definición de conectividades, entre otras funcionalidades.

La implementación de la interfaz de usuario, incluido el *Canvas*, tiene la característica de que está orientada a eventos, es decir, el sistema reacciona de acuerdo a las acciones que el usuario realiza. La biblioteca Qt permite esto a través del uso de los llamados "Signal/Slot" [64]. Este es un mecanismo que proporciona Qt, en donde funciones de la aplicación están asociadas a diferentes eventos que los elementos de la interfaz de usuario pueden invocar, por ejemplo al presionar un botón. Es así pues, que se aprovecha la utilización de este mecanismo, para mantener una coherencia entre las acciones que el usuario realiza y lo que se renderiza en tiempo real en el *Canvas*, ver figura 14.

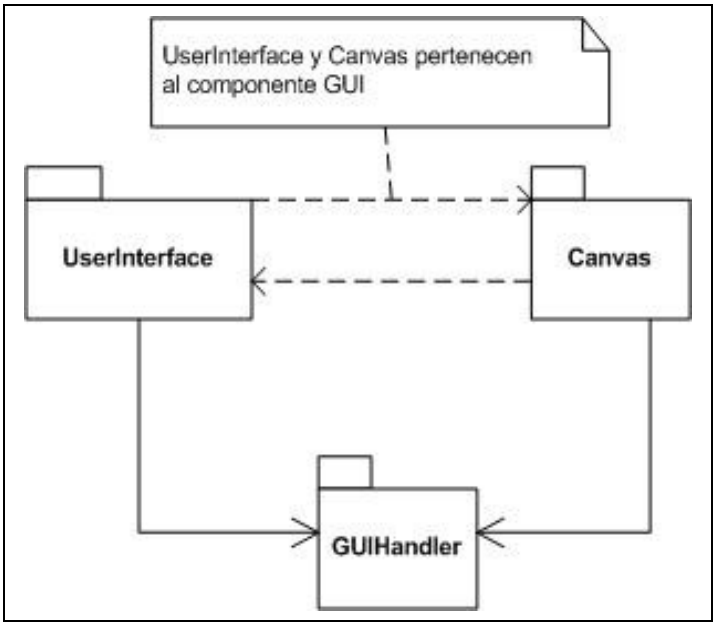


Figura 14 – Diagrama de relación entre los componentes.

En lo que se refiere al desarrollo en sí mismo, el componente *GuiHandler* mantiene el estado de lo que está sucediendo en la interfaz de usuario. Por ejemplo, dentro de este módulo se conoce el residuo que está seleccionado, el átomo que está seleccionado y así sucesivamente para todos los conceptos que se representan de la molécula. De esta manera, al ocurrir un evento en donde se deben mostrar datos, éstos se obtienen del componente *GuiHandler* mediante un conjunto de funciones que retornan información a ser desplegada, como por ejemplo valores numéricos o textos. Del mismo modo, cuando ocurre un evento dentro del *Canvas*, éste consulta qué elemento fue seleccionado y se envía una señal para que se actualice la información que se despliega en la interfaz de usuario.

Existe una comunicación bidireccional entre el *Canvas* y los elementos de la interfaz de usuario. Por otro lado, ambos están asociados al componente *GuiHandler* de manera que, al ocurrir eventos en cualquiera de los subcomponentes de GUI actualicen la información en *GuiHandler*

El diseño de la lógica del *Canvas* se basa en un conjunto de estados en los cuales se indican lo que éste debe realizar, como por ejemplo dibujar (ya sean bolas representando átomos o cilindros que representan a las conectividades), seleccionar

elementos que están presentes en el *Canvas*, mostrar un residuo canónico en forma completa, entre otros. Más información respecto al desarrollo del componente *Canvas* se encuentra en el Anexo G.

1.13 Implementación - Aspectos gráficos

Esta sección se enfoca en ciertos aspectos de implementación a nivel de los gráficos. Se describen la geometría utilizada para representar átomos y conectividades, el material y las luces utilizadas, y la implementación del *picking* que se permite en el *Canvas*.

1.13.1 Geometría

La geometría a dibujar en el sistema representa un problema de fácil solución. Los únicos objetos de la realidad que se deben representar gráficamente son residuos moleculares, conformados simplemente por átomos interconectados. A pesar de haber muchas representaciones posibles para una molécula (ver Anexo B) y por ende para sus átomos, para las necesidades del usuario es suficiente una representación *wireframe* (solamente líneas). Por consiguiente la geometría se reduce simplemente a la siguiente representación:

- Átomos: Son representados por esferas como se muestra en la figura 15. Se utilizó para el dibujado de una esfera la función *gluSphere()* de la biblioteca *GLU*.

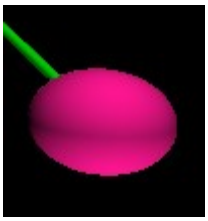


Figura 15 – Representación de un átomo.

- Conexión: Se representa mediante un cilindro entre los átomos involucrados, como lo muestra la figura 16. Para ello se creó una rutina que genera el cilindro ubicado entre dos puntos que se reciben como parámetros siendo ellos los centros de las bases del cilindro.

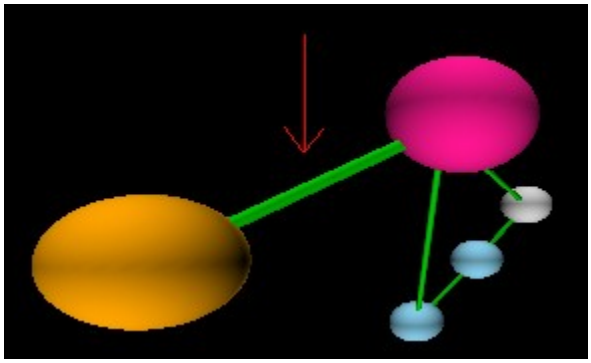


Figura 16 – Representación de un enlace entre átomos.

1.13.2 Cámara

La cámara se implementó con el objetivo de poder visualizar el residuo, o alguno de sus átomos desde cualquier ángulo y distancia. El modelo elegido consiste en posicionar la cámara siempre mirando hacia el centro geométrico de todos los átomos representados en la escena (punto B de la figura 17). Los movimientos de la cámara implementados son:

- Rotación: Es posible realizar rotaciones sobre el plano horizontal (flecha verde) o sobre el plano perpendicular al horizontal que pasa por la cámara (flecha azul), ver figura 17.

- Desplazamiento: El único desplazamiento posible de la cámara es en dirección a la recta determinada por la posición de la cámara y B. De esta manera se simula un Zoom sobre la escena. Este movimiento está acotado en cierto intervalo, de manera de que tenga sentido la escena, esto es, que la cámara no pueda posicionarse ni muy próxima ni muy lejana a B. De esta manera se garantiza que en todo momento la cámara apunta hacia el punto B.

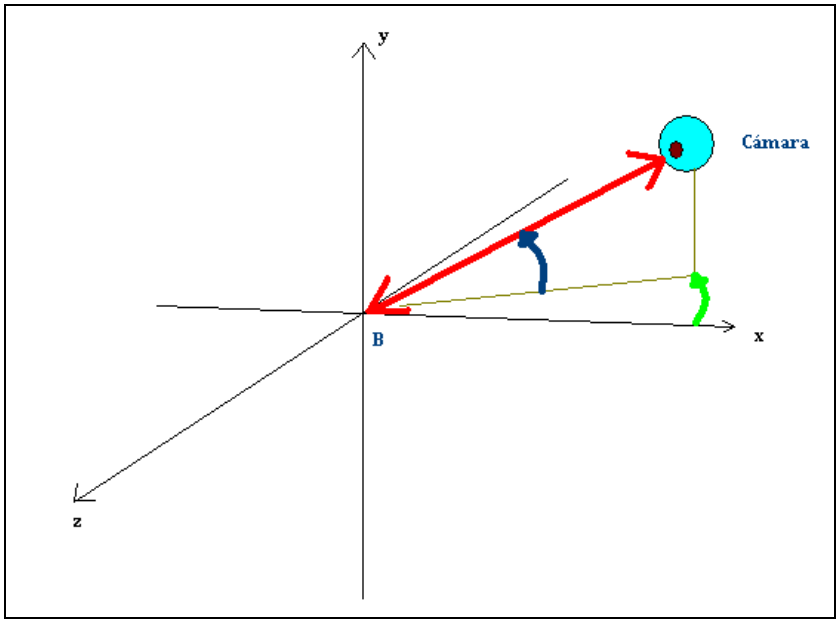


Figura 17 – Ubicación de la cámara.

1.13.3 Materiales y Luces

La iluminación de la escena se llevó a cabo utilizando las herramientas de materiales y luces que brinda OpenGL. En dicho modelo cada luz posee 3 componentes:

- Difuso: es el componente que representa la incidencia de luz sobre un objeto y proviene de un determinado punto. La intensidad con la que se refleja en la superficie del objeto puede depender del ángulo de incidencia, dirección, etc. Una vez que la luz incide sobre un objeto, ésta se refleja en todas direcciones por igual tal como se muestra en la figura 18.

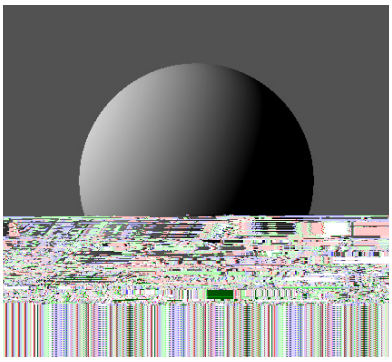


Figura 18 – Componente difusa de la luz.

- Especular: es el componente que representa la luz, que al incidir sobre un objeto, se ve reflejada con un ángulo similar al de incidencia. Usualmente se interpreta como la luz que produce los brillos. En la figura 19 se ilustran diferentes opciones que se pueden realizar con el componente especular.

Figura 19 – Diferentes opciones para el componente especular.

- Ambiente: Este componente representa el resto de la luz recibida por un objeto debido a la reflexión de luces en el resto de los objetos (tal vez luego de múltiples reflexiones) por ende no está asociada a una fuente y no es direccional. Observar que en la figura 20, asociada a la luz difusa se produce el efecto de que en la parte oscura de la esfera no es completamente negra en comparación a la esfera de la figura 18.

Figura 20 – Representación del componente ambiental de la luz.

Cada uno de estos componentes es representado por una terna RGB independizando cada canal de color.

Por otra parte se encuentran los materiales de los objetos. Un material define, para un determinado objeto, qué componentes refleja de cada tipo de luz. Por ejemplo, un plástico rojo, emitirá toda la componente roja de las luces ambiental y difusa, pero generalmente emitirá brillos de color blanco bajo una luz blanca, por lo que su componente especular será de color blanco. En definitiva, el color de un objeto queda definido por su material, como se muestra en la figura 21.

Figura 21 – Objeto con los mismos componentes de iluminación pero diferentes materiales.

En el caso de nuestra escena, se utilizaron 4 luces dispuestas de manera que desde cualquier posición posible de la cámara los átomos y sus conexiones estén siempre iluminados. Los materiales fueron definidos sobre los átomos de tal manera que el color resultante de la iluminación coincida con el estándar de colores CPK (ver Anexo E).

1.13.4 Picking

Picking es una acción que permite reconocer los objetos que se encuentran en un cierto entorno cercano al puntero del mouse dentro de un área visual, en el caso del proyecto en el *Canvas*. La técnica implementada se ejecuta al presionar el botón izquierdo del Mouse en el *Canvas* [37] [30].

Se implementó el *picking* utilizando las herramientas que nos provee OpenGL para dicho fin. La implementación consta de 4 fases durante el proceso de *picking* bien diferenciadas:

- **Modo Selección:** Se informa al hardware que se está en un estado (modo selección) en el cuál se simulará un dibujado. De esta manera, se le indica a OpenGL que lo que se dibujará no tendrá una consecuencia visual sino que simplemente se realizan todos los cálculos de proyecciones solamente con el objetivo de determinar qué objetos se obtienen como resultado del *picking*. Se le debe especificar una porción de memoria (buffer de selección), en la cual colocar los objetos que fueron seleccionados una vez que se retorne al modo *RENDER* (modo en el cual se dibuja la escena habitualmente). También se indica el tamaño de la región de selección cercana al puntero del Mouse, como se indica en la figura 22. Dicha región de selección es útil para determinar los objetos que fueron seleccionados, intersecando esta región con los objetos tal como se muestra en la figura 22.

Figura 22 – Implementación del *picking* en el *Canvas*.

- **Dibujado:** En modo selección se realiza el dibujado simulado de la escena. Como parte del dibujado, se asignan nombres a los objetos a dibujar, en el caso de los átomos, que son los objetos que desean manejarse mediante el *picking*, los nombres que se les asignó coinciden con sus identificadores dentro de la molécula. De estos se sabe que son únicos, evitando confusiones al momento de realizar el *picking*. En el ejemplo de la figura 22 los nombres de los objetos se establecen en las referencias.

- **Llenado del Buffer de Selección:** La biblioteca OpenGL determina los objetos que intersecan con la región de selección, almacenando posteriormente los nombres y las profundidades de los mismos en el buffer de selección. Dentro del buffer de selección se almacena la siguiente información:

1. Número de objetos seleccionados
2. Mínimo Z (profundidad) objeto 1
3. Máximo Z objeto 1
4. Nombre objeto 1
5. Mínimo Z (profundidad) objeto 2

.
.
.

- **Retorno a modo Render:** Posteriormente al llenado del buffer de selección, se retorna al modo *RENDER*. En ese momento se conoce que el contenido del buffer de selección es el descripto en el ítem anterior. En esta aplicación sólo se necesita seleccionar de a un átomo por vez por lo que se debe determinar el objeto con menor profundidad dentro del buffer, en este caso es el más cercano a la cámara, como el seleccionado. Luego a partir de la lógica subyacente implementada se determina la manera en que esta selección modificará el estado de los datos y el verdadero dibujado para luego proceder con el mismo. Por ejemplo, en el caso de seleccionar un átomo esto se indica rodeándolo mediante una esfera transparente.

1.14 Implementación – Manejo de Archivos.

Un punto clave de la aplicación fue la implementación de la lectura y escritura de los archivos que poseen la información de la molécula, la cual se crea a partir de la información que estos archivos contienen.

Según los requerimientos originales, se planteaba el manejo de dos formatos de archivos: los utilizados por el Framework AMBER y los utilizados por el programa GROMACS [35]. Éste último no es soportado por el programa debido a que no fue considerado dentro del alcance del mismo. Por lo tanto solamente se implementó la lectura y escritura de los archivos que utiliza AMBER. Como se mencionó en capítulos anteriores, los formatos de los archivos son: *dat*, *lib* y *pdb*.

Luego para implementar el manejo de las transformaciones, era necesario definir un cuarto formato de archivo, denominado "archivo de transformaciones" o archivo *transf*.

El archivo *transf*. contiene la información de cómo están definidas las transformaciones y no la información de los nuevos residuos canónicos que se crean. Esto se debe a que la información de los nuevos residuos y átomos (transformaciones y superátomos) se guardan en los archivos *dat*, *lib* y *pdb*. La implementación de este formato, permite realizar las funcionalidades de cargar y salvar una molécula. Además, es imprescindible para poder realizar en forma correcta el algoritmo de transformación *Coarse-Grain* → *All-Atom*.

El diseño propuesto se encuentra dentro del componente *FileHandler* y se basa en la abstracción de cada formato y encapsular su manejo dentro de una clase especializada. Como se observa en el diagrama de diseño del manejo de archivos de la figura 23,

existe una interfaz que posee 4 operaciones. Estas son para cargar y salvar los archivos que representan a la molécula y para generar los archivos relacionados con el algoritmo *Coarse-Grain* to *All-Atom*. Quien lleva a cabo estas operaciones es el controlador llamado *FileManager*, que es quien delega la responsabilidad de leer y escribir cada uno de los archivos a cada clase particular. Así pues, este diseño permite extender fácilmente la lectura/escritura hacia otros formatos de archivos, por ejemplo los de GROMACS, agregando clases que sean expertas en el formato de archivo a agregar.

En la figura 23 se presenta el diseño para el manejo de archivos. Se puede observar la interacción existente entre el controlador de las operaciones y cada una de las clases particulares.

Figura 23 – Diagrama de clases de diseño para el manejo de archivos.

- FileManager: es una clase *controller* que implementa la interfaz *IFileManager*. Esta clase delega la responsabilidad de leer y escribir a la clase que corresponda.
- datManager: es la clase encargada del manejo del archivo de formato *dat*.
- libManager: es la clase que se encarga del manejo del formato de archivo *lib*.
- readPdb: esta clase realiza la lectura y escritura de los archivos en formato *pdb*.
- transfManager: es la clase especializada en el manejo de los archivos que contienen la información de las transformaciones creadas en el CG Manager. Estos son los archivos con formato *transf*.

Cada una de estas clases realiza un chequeo de integridad de los datos al momento de leer los archivos, además se verifica que todos los parámetros relevantes estén definidos. En caso de existir algún error, se le informa al usuario mediante una breve descripción del mismo.

5 Diseño de los Algoritmos (AA-CG y CG-AA)

En este capítulo se presenta el diseño de los algoritmos de conversión de moléculas que son una de las principales funcionalidades del sistema *CG Manager*.

Previo a la descripción de cada uno de los algoritmos, se comentan ciertas consideraciones generales con el objetivo de establecer el vocabulario y la nomenclatura que se utiliza a lo largo de este capítulo. Esta sección se entiende fundamental para la comprensión detallada de ambos algoritmos.

Seguidamente se presenta el algoritmo AA-CG que consiste en la conversión, a partir de la especificación del usuario, de una molécula en representación *All-Atom* (AA) a una molécula en representación *Coarse-Grain* (CG). La molécula en representación CG, obtenida como resultado de la aplicación del algoritmo AA-CG, es

una simplificación geométrica de la molécula en representación AA. En ella, se intenta conservar las propiedades químicas de la molécula original. Por lo tanto, en la presentación del algoritmo AA-CG se describe detalladamente la creación de la salida del algoritmo, es decir, la creación de la molécula en representación CG y del archivo de transformación que posee la información de la simplificación realizada sobre la molécula original.

Posteriormente, se presenta el algoritmo CG-AA. El mismo consiste en obtener, a partir de una molécula en representación CG (generada previamente a través del algoritmo AA-CG) y de un archivo de transformación que describe la simplificación AA-CG original (ver descripción del formato del archivo *transf* en el Anexo A), una molécula equivalente en representación AA.

1.15 Consideraciones generales

Una molécula está formada por un conjunto ordenado de residuos. Cada uno de los residuos posee un átomo denominado **cola** y otro átomo denominado **cabeza**. Con excepción del primer residuo de la molécula que posee solamente un átomo cola y el último residuo de la molécula que posee solamente un átomo cabeza.

Figura 24 – Molécula Simple formada por tres residuos.

En la figura 24 se puede observar un ejemplo de una molécula simple compuesta de tres residuos. Allí se identifica cuáles son los átomos cabeza y cola de cada residuo. La unión de los residuos se realiza a través de los átomos cabeza y cola, repitiéndose el mismo mecanismo para todos los residuos que compongan a la molécula.

Ambos algoritmos a describir se enfocan en resolver el problema en un solo residuo de la molécula, repitiéndose los mismos pasos hacia los restantes residuos. Por lo tanto, en las siguientes secciones el análisis de los algoritmos se realizará basándose en un residuo solamente.

En la naturaleza existe un conjunto finito de tipos de residuo denominados residuos canónicos. Cada residuo real dentro de una molécula está asociado a un **residuo canónico**, se dice que dicho residuo real es una **instancia** del residuo canónico asociado. En lo que respecta a la información relevante para los algoritmos, la diferencia entre una **instancia** de un residuo y su **residuo canónico** es solamente la posición de cada uno de los átomos que lo componen. Ciertas propiedades son invariantes entre la instancia y el residuo canónico. Estas propiedades son: el conjunto de átomos, las conectividades entre átomos, el átomo cola y el átomo cabeza, entre otras. Diremos entonces que la **instancia** y su **residuo canónico** asociado son topológicamente equivalentes y simplemente difieren en su geometría, tal como se ilustra en la figura 25.

Figura 25 – Equivalencia topológica entre el residuo canónico y las instancias de ese residuo.

Otro de los aspectos importantes a tener presente para la comprensión de las descripciones de los algoritmos, es la diferencia entre las versiones CG y AA de los residuos. Dado un residuo, se definen sobre el mismo cuatro versiones que resultan adecuadas para la descripción de los algoritmos:

- **Instancia AA** – Se trata de la representación *All-Atom* de la instancia del residuo canónico que se esté considerando. Dependiendo del algoritmo podrá tratarse del residuo instanciado antes de la aplicación (caso del algoritmo AA-CG) o luego de la aplicación (caso del algoritmo CG-AA).

- **Instancia CG** – Se trata de la representación *Coarse-Grain* de la instancia del residuo que se esté considerando. Dependiendo del algoritmo, podrá tratarse del residuo instanciado antes de la aplicación (caso del algoritmo CG-AA) o luego de la aplicación (caso del algoritmo AA-CG).

- **Residuo Canónico AA** – Indica la representación *All-Atom* del residuo canónico que se esté considerando.

- **Residuo Canónico CG** – Es la representación *Coarse-Grain* del residuo canónico que se esté considerando.

A continuación en la figura 26 se ilustra un ejemplo de las cuatro versiones definidas anteriormente.

Figura 26 – Esquema de las 4 versiones posibles de residuo.

Antes de comenzar la descripción, es importante aclarar que se realizará la explicación de ambos algoritmos basándose en una sola molécula, aunque lo habitual es que luego de la simulación se obtengan varias instancias de una molécula. Ésta simplificación, se puede realizar debido a que la aplicación del algoritmo sobre un conjunto de moléculas, simplemente consiste en aplicarlo molécula a molécula individualmente.

Aunque se describan ambos algoritmos de manera independiente, debe quedar claro que existe cierta relación entre estos. Se explicará entonces cómo se relacionan los algoritmos.

De acuerdo a las investigaciones que se llevan a cabo por el SBM, es deseable poder realizar una simulación utilizando una versión simplificada en representación CG de una molécula que se encuentra inicialmente en representación AA. Para ello se debe realizar la conversión AA-CG definiendo el usuario previamente la forma en que debe llevarse a cabo tal simplificación.

A partir de la aplicación del algoritmo AA-CG se obtiene una molécula equivalente a la original en representación CG, además se almacena la información que indica cómo se llevó a cabo la simplificación que permitirá luego la aplicación del algoritmo CG-AA. Posteriormente a la molécula obtenida en representación CG se le realiza una simulación de dinámica molecular, la cual modifica las posiciones de sus átomos y por consiguiente de sus residuos, manteniendo la topología de la molécula sin modificaciones.

La molécula CG obtenida luego de la simulación es la que se desea convertir a la representación AA, y junto con la información almacenada en el archivo de transformaciones generado en el algoritmo AA-CG, constituyen los datos de entrada del algoritmo CG-AA que permite obtener una molécula aproximada en representación AA de M. El archivo de transformaciones *transf* es el nexo entre ambos algoritmos, por lo que llegamos a la conclusión de que no existe independencia completa entre ellos ya que parte de la entrada del algoritmo CG-AA tiene que haber sido generada a partir de una aplicación anterior del algoritmo AA-CG, ver figura 27.

Figura 27 – Secuencia de pasos realizadas a una molécula M.

Según se ilustra en la figura 27, la molécula M'' obtenida luego de la aplicación de ambos algoritmos es una aproximación de la molécula M' obtenida computacionalmente luego de aplicar la simulación sobre la representación AA de M.

1.16 Algoritmo All-Atom – Coarse Grain

Dado un residuo cuya representación es *All-Atom*, se desea definir un residuo en representación *Coarse-Grain*. Se establece para ello una correspondencia o mapeo entre los átomos del residuo *All-Atom* original y superátomos del residuo *Coarse-Grain*.

De acuerdo a lo especificado por el usuario, se define una transformación a partir de un residuo canónico. En la misma queda determinada la conformación de cada uno de los superátomos en términos de átomos. También se establece la manera de calcular la posición del nuevo superátomo a partir de un subconjunto de los átomos que lo componen. Existen tres formas posibles de determinar la posición, estas son:

- Centro de Masa de un subconjunto de átomos
- Centro Geométrico de un subconjunto de átomos
- Posición de un átomo particular

También se especifica por parte del usuario los datos propios de cada superátomo creado (nombre masa, carga, etc.) y del nuevo residuo (nombre, campos de fuerza, etc.). Una vez definidas una serie de transformaciones, el usuario establece qué transformación se aplicará a cada una de las instancias de los residuos canónicos (puede no aplicarse a ninguna de ellas).

En el momento de la aplicación del algoritmo se recorren todas las instancias de la molécula y se aplica la transformación correspondiente, de acuerdo a lo establecido por el usuario a cada una de las instancias de los residuos canónicos.

A continuación se describe la forma de aplicar una transformación a una determinada instancia de residuo canónico. Dada una instancia de residuo en particular y una transformación, se procede en primera instancia, a crear cada uno de los superátomos. Para determinar la posición del superátomo dentro de la instancia del residuo se realiza el mismo tipo de selección (Centro de masa, Centro Geométrico u

Otra posición) sobre el conjunto de átomos que fueron considerados para la posición en el residuo canónico al momento de definirse la transformación.

Por ejemplo, tal como se muestra en la figura 28, si en la transformación está establecido que el superátomo A contiene a los átomos *a*, *b* y *c* y la posición de A para el residuo canónico se corresponde al centro geométrico de *a*, *b* y *c*. Para determinar la posición de la instancia del superátomo (A') se realiza el cálculo del centro geométrico de los átomos *a'*, *b'* y *c'* pertenecientes al residuo instanciado.

Figura 28 – Ejemplo de cálculo de la posición para una instancia de superátomo.

Para cada uno de los superátomos definidos en la transformación se realiza el proceso descripto en el párrafo anterior, determinando de esa manera las posiciones de los superátomos para el residuo instanciado CG.

Existen ciertos datos propios del nuevo residuo *Coarse-Grain* y de los superátomos que éste contiene que no son calculados a través del algoritmo, ya que el algoritmo solamente calcula las posiciones de los superátomos. Estos datos son definidos por el usuario al construir la transformación que convierte el residuo *All-Atom* en el residuo *Coarse-Grain*. A nivel del nuevo residuo los datos definidos por el usuario son:

- Nombre
- Cabeza
- Cola
- Valores para los campos de fuerza
- Conectividades entre los superátomos
- Valores para las diferentes propiedades del residuo.
- Nuevos tipos de átomos.

A nivel de los superátomos que el nuevo residuo contiene los datos definidos por el usuario son:

- Nombre
- Tipo de átomo
- Masa
- Carga

Entre otras propiedades.

1.17 Conversión *Coarse Grain* – All-Atoms

A continuación se describirá en detalle el algoritmo de conversión CG – AA para un residuo instancia en representación *Coarse-Grain*, extendiéndose este proceso a todos los residuos pertenecientes a la molécula *Coarse-Grain*. El objetivo geométrico de este algoritmo es poder hallar, en la zona donde se encuentra el residuo *Coarse-Grain* instanciado, las posiciones de los átomos en su representación *All-Atom*. Éstas posiciones son suficientes debido a que las conectividades entre átomos y el resto de los datos del residuo (masas, cargas, campos de fuerza, etc.) vienen dados por el residuo canónico original asociado.

Se tomará como caso de estudio el residuo canónico *Coarse-Grain* de la figura 29.

Figura 29 – Residuo canónico *Coarse-Grain*.

El superátomo A de la figura 30, contiene en su interior los átomos a, b y c:

Figura 30 – Composición del superátomo A.

Como se observa en la figura 31, se puede construir, sobre el centro del superátomo A, un sistema de coordenadas (x, y, z) unitario, tomando los vectores que se dirigen hacia los superátomos B, C y D. Considerando a los átomos a, b y c de la figura 30, las posiciones de los mismos son calculadas en relación al sistema de coordenadas (x, y, z). Estas posiciones deben ser determinadas tanto para el superátomo A representado en el residuo canónico, como para el superátomo A' representado en la instancia del residuo en el sistema (x', y', z') como se aprecia en la figura 31. Para esta última instancia se aplica el algoritmo CG-AA. Cabe aclarar que el residuo instanciado puede haber cambiado su forma respecto del canónico, como se ilustra en la figura 31:

Figura 31 – Transformación entre sistemas cartesianos.

Con el objetivo de calcular las posiciones de los átomos instanciados en el correspondiente sistema de coordenadas del SAT A instanciado (A'), se debe determinar la transformación lineal ***T*** que transforma (x, y, z) en este nuevo sistema (x', y', z').

Una vez obtenida la transformación se tiene que al aplicar la transformación lineal ***T*** a las posiciones de los átomos de A se obtienen las posiciones de los átomos de A'. O sea,

T(a) = a' donde ***a*** pertenece a ***A*** y ***a'*** es su correspondiente en ***A'***

Luego de determinar ***T*** se deben posicionar adecuadamente los átomos obtenidos luego de la aplicación de la transformación ***T*** para que sean interpretados respecto a la posición de A'. El movimiento que lleva desde la posición de *A* hacia la posición de A' es una traslación de vector AA'.

Al componer **T** con la traslación se obtiene una transformación afín [34]. Las imágenes de los átomos de A en esta transformación afín contemplan de manera aproximada la deformación sufrida durante la simulación.

$Tr_{AA'} \circ T(a) = a'$ donde ***a*** pertenece a ***A*** y ***a'*** es su correspondiente en ***A'***

Para completar el algoritmo, dentro del residuo se realiza lo explicado en el párrafo anterior para cada uno de los superátomos del residuo. La realización del procedimiento descripto a cada uno de los residuos de la molécula, permite obtener la molécula en representación AA con la mínima pérdida de información posible.

1.18 Pruebas

En la siguiente sección se presentan diferentes pruebas efectuadas sobre los algoritmos con el objetivo de mostrar la performance y el nivel de eficacia del algoritmo de transformación inversa *Coarse-Grain – All-Atom* mostrando cuán correcto es la realización de esta transformación y cuánto se ajusta a la molécula original.

1.18.1 Pruebas de performance

Las pruebas de performance consisten en realizar la ejecución de los algoritmos y tomar el tiempo de ejecución. Esta evaluación se realizó en diferentes *PCs* de prueba. A continuación se presentan las tablas con los valores obtenidos para cada uno de los casos. No se comparan tiempos con otra herramienta, ya que carece de sentido porque no existe una herramienta similar. Por lo tanto, los tiempos deben tomarse simplemente como informativos o en todo caso compararlos con el tiempo empleado en el proceso manual que se realiza hoy en día.

1.18.1.1 Algoritmo AA-CG

Se tomaron dos ejemplos de molécula para realizar el test de performance. El primer caso de prueba a considerar es una molécula con cien átomos y la molécula transformada *Coarse-Grain* contiene en promedio 2,5 átomos por superátomo creado. Los resultados se presentan en la tabla 1.

PC de prueba	Tiempo de ejecución (segundos)
Pentium 4 3,0 Mhz / 512 Mb RAM	0.121
Amd Sempron 1,8 Mhz / 1,5 Gb RAM	0.081
Core 2 Dúo 1,85 Mhz (x2) / 4 Gb RAM	0.016

Tabla 1 – Tiempo de ejecución algoritmo AA-CG para una molécula con 100 átomos.

En la segunda prueba de medición del tiempo de ejecución, se considera una molécula con mil átomos y la molécula transformada *Coarse-Grain* contiene en promedio de 3 átomos por superátomo creado. Los resultados se presentan en la tabla 2.

PC de prueba	Tiempo de ejecución (segundos)
Pentium 4 3,0 Mhz / 512 Mb RAM	1.211
Amd Sempron 1,8 Mhz / 1,5 Gb RAM	0.765
Core 2 Dúo 1,85 Mhz (x2) / 4 Gb RAM	0.167

Tabla 2 – Tiempo de ejecución algoritmo AA-CG para una molécula con 1000 átomos.

1.18.1.2 Algoritmo CG-AA

A continuación se presentan los resultados para el algoritmo inverso CG-AA considerando para ello una molécula creada a través del algoritmo AA-CG. En el primer caso la molécula a reconstruir contiene cien superátomos conteniendo cada uno en promedio 2,5 átomos. En cada prueba se cuenta con 1000 archivos pdb como entrada para ejecutar el algoritmo. En la tabla 3 se muestran los resultados obtenidos.

PC de prueba	Tiempo de ejecución (segundos)
Pentium 4 3,0 Mhz / 512 Mb RAM	435
Amd Sempron 1,8 Mhz / 1,5 Gb RAM	337
Core 2 Dúo 1,85 Mhz (x2) / 4 Gb RAM	72

Tabla 3 – Tiempo de ejecución algoritmo CG-AA para 1000 archivos de entrada y una molécula con 100 superátomos.

El segundo caso se basa en una molécula con mil superátomos conteniendo cada uno en promedio de 3 átomos. Al igual que en la prueba anterior, se cuenta con 1000 archivos pdb de entrada. En la tabla 4 se muestran los resultados obtenidos.

PC de prueba	Tiempo de ejecución (segundos)
Pentium 4 / 512 Mb RAM	1243
Amd Sempron 1,8 Mhz / 1,5 Gb RAM	901
Core 2 Dúo / 4 Gb RAM	184

Tabla 4 – Tiempo de ejecución algoritmo CG-AA para 1000 archivos de entrada y una molécula con 1000 superátomos.

1.18.2 Pruebas de correctitud de los algoritmos

Para poder evaluar la correctitud de los algoritmos definimos una medida que representa la similitud geométrica existente entre dos moléculas con el mismo conjunto de átomos y conectividades. Llamaremos a esta medida *Coeficiente de Diferenciación* y se calcula de la siguiente manera: Dadas las dos moléculas a comparar se toma de ambas moléculas la misma pareja de átomos. Se calcula la distancia entre dicha pareja de átomos en cada una de las moléculas y se considera la tasa de variación de la distancia tomada para la segunda molécula, respecto de la distancia tomada para la primera molécula. Este procedimiento se lleva a cabo para todas las parejas posibles de átomos. Luego, considerando todas las tasas medidas, se toma la mayor de ellas, siendo la elegida el *Coeficiente de Diferenciación*. Matemáticamente podemos expresar el *Coeficiente de Diferenciación* como:

$$C_d(M, M') = max\left(\frac{|dist(i,j) - dist(i',j')|}{dist(i,j)} \right) \text{ para todo } i,j \text{ en } M / (i \text{ distinto de } j) \text{ con } i', j' \text{ sus correspondientes en } M'$$

A continuación se presentarán los valores obtenidos en las pruebas. Se listan los resultados obtenidos con diferentes moléculas dependiendo del promedio de

superátomos por residuo, factor determinante a la hora de reconstruir la molécula ya que de ello depende la selección de los ejes cartesianos (Ver sección 5.3 de este capítulo).

1.18.2.1 Prueba de identidad

Consiste en tomar una molécula, aplicarle el algoritmo *AA-CG* e inmediatamente al resultado obtenido aplicarle el algoritmo *CG-AA*. Lo deseable para este caso es que el coeficiente de diferenciación entre la molécula original y la resultante de aplicar el algoritmo, sea exactamente la molécula original. En la tabla 5 se presenta el *Coeficiente de Diferenciación* para diferentes moléculas.

Media de superátomos por residuo	C_d
4	0
5	0
6	0

Tabla 5 – *Coeficiente de Diferenciación* para el caso "identidad".

1.18.2.2 Prueba Caso General

El caso "general", consiste en tomar una molécula, aplicarle el algoritmo *AA-CG*. Posteriormente se le aplican ciertas deformaciones y movimientos a la molécula resultante, tratando de reproducir lo que sucedería en una simulación de dinámicas moleculares. Por último, se aplica el algoritmo *CG-AA*. Lo deseable para este caso es que el *Coeficiente de Diferenciación* resultante sea cero, es decir, que se obtenga la molécula original. En la tabla 6 se presentan los resultados obtenidos para el caso de prueba general.

Media de superátomos por residuo	C_d
4	0.041
5	0.049
6	0.057

Tabla 6 – *Coeficiente de Diferenciación* para el caso "general".

1.19 Conclusión

Para establecer una conclusión respecto de la validez del algoritmo analizaremos las pruebas realizadas. En primer término, evaluando las pruebas de performance consideramos que los valores son más que aceptables en comparación a los tiempos que hoy en día se requieren para la creación de modelos *Coarse-Grain* por parte del SBM.

Para el caso de las pruebas de correctitud podemos concluir:

- En la prueba de identidad se presenta con claridad que al no existir pérdida de información la reconstrucción es perfecta, esto es, el denominado *Coeficiente de Diferenciación* es exactamente igual a 0.

- En el caso general el cálculo del coeficiente de diferenciación para los casos estudiados es menor a 0,1 valor meta preestablecido antes de nuestras pruebas. Una aclaración que se debe realizar es que el grupo SBM, luego de obtenidas las moléculas *AA* las post-procesan en una aplicación que modifica las posiciones geométricas de los átomos, de forma tal que se reconstruye la realidad química que mejor se aproxima a la geometría obtenida.

Se considera por lo tanto que los algoritmos globalmente tienen un comportamiento correcto y eficiente de acuerdo a los requerimientos propuestos por el grupo SBM.

6 Conclusiones y trabajo a futuro

En el presente capítulo se presentan las conclusiones finales del proyecto. Además, se describe un resumen de las características de la aplicación desarrollada y por último se mencionan diferentes aspectos a mejorar en el futuro.

1.20 Conclusiones finales

Hay varios aspectos a destacar sobre la realización de este proyecto de grado. En primer lugar, desde el punto de vista de las ideas iniciales que tuvimos al comienzo del proyecto, planteaban varios aspectos que presentaban dificultades para el desarrollo de la aplicación final. Una clara dificultad previa era el entendimiento de la realidad en donde se enmarca la propuesta presentada. Es un área que utiliza una terminología que era desconocida para nosotros. El otro aspecto previo en el cual veíamos dificultades era en el desarrollo del algoritmo CG – AA.

Posteriormente a la finalización del desarrollo del proyecto pudimos constatar que las dificultades previas que considerábamos efectivamente fueron complicaciones a lo largo del mismo. El entendimiento de la realidad fue progresando a través de sucesivas reuniones realizadas con SBM y de largas discusiones entre nosotros, tratando de "bajar a tierra" las ideas que iban surgiendo.

Otro punto de dificultad fue el de acotar y definir claramente los requerimientos a realizar. Esto se fue negociando con SBM en la medida que se iba avanzando en la investigación del estado del arte. Como se mencionó en el capítulo 4, inicialmente la idea de SBM era la de construir una aplicación de visualización molecular que contara con la capacidad de transformar moléculas en representación *All-Atom (AA)* en la representación *Coarse-Grain (CG)* y viceversa, es decir, a partir de una molécula *Coarse-Grain* y un archivo de transformaciones volver a generar la molécula *All-Atom* original. Al realizar una investigación del estado del arte en materia de aplicaciones que generaran moléculas *Coarse-Grain*, no encontramos ningún software que permitiera definir este tipo de moléculas completamente por el usuario que se adecuaran a los requerimientos de SBM. Sí existe una aplicación, VMD, que permite definir este tipo de moléculas, pero a través de algoritmos predeterminados, los cuales sirven para algunos casos, pero no en forma genérica, como es la necesidad de SBM. Paralelamente la investigación se centró en la búsqueda de bibliotecas que permitieran la construcción de una aplicación de visualización molecular. Al comparar la calidad y variedad de las representaciones gráficas de, por ejemplo VMD, los requerimientos se ajustaron de tal manera de considerar solamente la construcción de un editor de propiedades y generación de moléculas *Coarse-Grain* así como también tener la posibilidad de recuperar la estructura *All-Atom* a partir de moléculas *Coarse-Grain*.

El entendimiento de los archivos de topología y la definición del archivo de transformaciones fue una etapa que llevó bastante tiempo. A pesar de existir documentación sobre el formato del archivo dat y pdb, no lo fue así con el archivo lib, en donde se fue descubriendo el formato en forma conjunta con los investigadores de

SBM en diversas reuniones. El archivo de transformaciones fue un formato definido por nosotros el cual se iba mejorando a medida que se iba generando el algoritmo CG-AA.

En lo que se refiere a la interfaz gráfica de usuario, la implementación del reconocimiento del *picking* en el *Canvas* fue una difícil tarea a resolver, así como el manejo y coordinación de los eventos que ocurren tanto en el *Canvas* como en la GUI y viceversa. Estas dificultades surgieron en la etapa del desarrollo y eran aspectos que no habían sido previamente evaluados en cuanto a su dificultad.

La elección del lenguaje de programación creemos que fue correcta ya que C++ permite la integración de la biblioteca gráfica Opengl, es un lenguaje conocido y utilizado asiduamente por nosotros y permite ejecutar sobre cualquier plataforma, aunque el código debe ser nuevamente compilado en cada una de ellas. A su vez, para el desarrollo de la interfaz gráfica de usuario, se tuvo que realizar una investigación de bibliotecas que permitieran el desarrollo de la misma. Nos decidimos por Qt, que aunque es una biblioteca relativamente nueva, cuenta con bastante documentación y ejemplos que sirven de ayuda.

La interfaz gráfica muestra y permite editar las diferentes propiedades que poseen los diferentes archivos de entrada al sistema. Esto redunda positivamente en su utilización, ya que a través de una interfaz gráfica de usuario amigable se permite crear una molécula *Coarse-Grain* y/o editar propiedades de la molécula (CG ó AA original), generando nuevamente los archivos.

Un aspecto positivo de esta aplicación es que agiliza y automatiza el trabajo que hoy en día realiza SBM. El tiempo que se invierte en la definición de moléculas *Coarse-Grain* se reduce drásticamente, ya que actualmente para definir este tipo de moléculas, considerando que editan principalmente "a mano" los archivos de topología, se demora aproximadamente entre 15 días y 1 mes de trabajo. Con esta aplicación a lo sumo se puede demorar 8hs, dependiendo de la molécula con la que se esté trabajando. Por otro lado, el formato de archivos de topología permite interactuar directamente con programas de simulación de dinámicas moleculares, los que provee AMBER. En este sentido se pueden realizar extensiones en el tipo de formato de archivo de topología, por ejemplo, permitiendo la utilización de los archivos que soporta GROMACS, estos detalles se explican en la sección 6.3. Otro punto a destacar es la importancia del producto desarrollado, ya que este va a ser utilizado por un usuario final que es externo a la Facultad de Ingeniería. Nosotros asumimos el compromiso de realizar un producto que cumpliera con los principales requerimientos solicitados por SBM (edición de propiedades y algoritmos AA-CG y CG-AA), ya que una de las consecuencias de la realización de este proyecto es el poder distribuir esta aplicación en la comunidad científica. De hecho, está desarrollado a través de herramientas de código libre y se cumplió con el objetivo de que fuera multiplataforma, aunque una desventaja es que para ejecutar en las diferentes plataformas es que el código debe compilarse nuevamente y debe existir una versión de la biblioteca Qt para esa plataforma.

En cuanto a los algoritmos desarrollados, el más complejo de implementar fue el CG-AA, al que denominamos informalmente "vuelta hacia atrás". Si bien recibimos 2 o 3 sugerencias de cómo llevar a cabo el algoritmo por parte de SBM (ellos incluso tienen una solución pero solamente es adecuada para algunos casos particulares), creemos que nuestra solución es genérica, ya que considera la deformación de la geometría de la

molécula luego de realizar una simulación. Este algoritmo da buenos resultados si las deformaciones locales no son grandes (verificado empíricamente en la mayoría de los casos). La diferencia geométrica entre la molécula original y la obtenida luego de aplicar ambos algoritmos (AA-CG y luego CG-AA) que obtuvimos en las pruebas realizadas fue menor a un 10% valor que consideramos muy satisfactorio y una muy buena aproximación a la molécula original, que luego permita un post-procesamiento y normalización química a través de programas específicos utilizados por SBM (ver los resultados obtenidos en el capítulo 5 a través del coeficiente de diferenciación).

La implementación del producto a través del desarrollo de un modelo utilizando el paradigma de orientación a objetos, permitió una buena abstracción de la realidad, además de la separación en componentes en donde cada uno se enfoca en cierto aspecto de las funcionalidades que brinda la aplicación. Este diseño permite la extensión de posibles mejoras o la inclusión de nuevas funcionalidades.

El algoritmo CG-AA procesa los archivos de entrada en forma serial, esto tiene como desventaja en el mal aprovechamiento del tiempo de uso de CPU en el caso de contar con computadores con varios núcleos de procesamiento. Al ejecutar el algoritmo con grandes volúmenes de datos, es decir una cantidad importante de archivos pdb como entrada al algoritmo, la duración de la ejecución crece en forma lineal respecto de la cantidad de archivos (suponiendo archivos de similares características). Este punto se puede solucionar mediante la utilización de técnicas de procesamiento paralelo-distribuido, ya sea mediante threads, ejecución distribuída a través de bibliotecas de pasaje de mensajes como MPI, entre otras que consideramos fuera del alcance del proyecto por su complejidad agregada y por obtener tiempos de ejecución aceptables, a pesar de no haber implementado estas características adicionales.

1.21 Resumen de las características de lo implementado

A modo de resumen técnico mencionaremos algunos datos relativos a este proyecto de grado:

- Está implementado en el lenguaje C++ mediante el paradigma de orientación a objetos.
- Se utilizó la biblioteca OpenGL para el renderizado de los residuos y de los superátomos.
- La interfaz gráfica está implementada basándose en la biblioteca Qt en su versión 4.6.2.
- El sistema utilizado como entorno de programación fue QtCreator.
- Se implementaron 55 clases distribuidas en 4 módulos: GUI (25 clases), *GUIHandler* (1 clase), *FileHandler* (12 clases) y *Molecule* (18 clases).

- Hay 11268 líneas de código efectivas y el porcentaje de código en los distintos módulos se representa en la figura 32. Este gráfico se obtuvo con la herramienta LocMetrics ejecutada en Windows.

Figura 32 - Porcentaje de líneas de código por módulo.

- La aplicación se testeó en dos sistemas operativos diferentes: Windows (Xp y 7) y Linux (Ubuntu versión 10.4).

- Se realizaron diferentes instaladores para los sistemas operativos Windows y Linux. Los instaladores se hicieron con la herramienta para generar instaladores multiplataforma llamada *BitRock Install builder* versión 6.

- Durante la implementación del proyecto se utilizó como manejador de versiones de código el sistema TortoiseSVN, alojando el código del proyecto en un repositorio creado a través de la herramienta online Google Code.

1.22 Trabajo a futuro

A continuación se describen posibles mejoras a realizar así como funcionalidades que no se implementaron. Estas funcionalidades fueron recortadas del alcance del proyecto debido a que las mismas implicaban un nivel de dificultad elevado y por consiguiente una extensión del cronograma del proyecto.

Se enumeran diferentes mejoras en varias líneas de trabajo relacionadas con la parte de dibujado y representación de átomos, en el sistema de *picking*, mejoras en la interfaz gráfica de usuario en general, aumento de la cantidad de formatos de archivo que pueda interpretar *CG Manager*, etc.

- Extender los formatos de archivo para leer y escribir. Nuestro proyecto solo soporta el formato de archivos que utiliza el Framework AMBER (archivos *dat* y *.lib*). También permite el manejo del tipo de archivo *pdb*, que lo utilizan la mayoría de las aplicaciones de visualización molecular y está presente en todas las bases de datos biomoleculares. La extensión sería particularmente para los formatos que utiliza el programa GROMACS, que también es utilizado por el GSB. Estos formatos son los especificados por las terminaciones *Top*, *gro*, *itp*, *prmtop* y *inpcrd*.

- Incluir diferentes visualizaciones extra o la realización de gráficos que serían de ayuda para un post-procesamiento de las simulaciones y para obtener datos o información sobre diferentes propiedades de los residuos, aunque las herramientas existentes permiten la realización de estas funcionalidades.

- Mejorar la interacción con el usuario en la definición de nuevos diedros, que puedan ser creados a través del área de dibujado. Por ejemplo, definir diedros seleccionando los átomos involucrados directamente haciendo *picking* en el *Canvas*.

- Colorear los átomos renderizados según diferentes tipos de representación. Actualmente se colorean únicamente según el esquema CPK.

- Permitir el reconocimiento del *picking* sobre las conectividades con el objetivo de seleccionarlas para poder eliminar y asignar diferentes valores de *force-field* length, etc.

- Mejorar la interfaz gráfica de usuario agregando una barra de herramientas, más funcionalidades en el menú principal, permitir diferentes acciones con el botón derecho del Mouse de manera que la interfaz gráfica de usuario sea más amigable.

- Integrar una guía o manual de ayuda al usuario en la interfaz gráfica.

- Generar un ejecutable y realizar el testing de la aplicación para el sistema operativo MacOS.

- Implementar, utilizando threads, la conversión de una molécula *Coarse-Grain* al formato *All-Atom*. De esta manera se puede mejorar el acceso a los archivos. Recordar que la transformación antes mencionada implica acceder, modificar y crear un nuevo conjunto de archivos (con formato *pdb*), donde cada uno de los archivos originales contiene la representación *Coarse-Grain* de la molécula, variando entre ellos únicamente las posiciones de los superátomos, mientras que en los archivos finales se almacena la representación *All-Atom* de la molécula. Las particularidades de este proceso, donde el acceso, procesamiento y creación de un nuevo archivo es independiente del procesamiento de los otros archivos, sustenta la idea de paralelizar esta etapa del algoritmo.

A.1 Anexo A – Formatos de archivos

En el presente documento se explica el formato y contenido de los archivos DAT, LIB y PDB utilizados por la suite AMBER, los cuales se utilizan como entrada y salida de la aplicación CG Manager. Además se explica el formato y contenido del archivo TRANSF, el cual es propio de la aplicación CG Manager y es utilizado para poder almacenar las transformaciones generadas por la aplicación.

Como consideración preliminar se presentará el formato descriptor de archivos en el lenguaje FORTRAN, que se utiliza a lo largo del presente documento con el objetivo de describir el formato de cada tipo de archivo. En la figura 1 se presenta una tabla con el descriptor y su correspondiente significado.

Descriptores	Significado
Iw	w caracteres para un entero.
Fw,d	w caracteres para un real, con d lugares para decimales.
Ew,d	w caracteres para un real, con d lugares para decimales.
Lw	w caracteres para un dato lógico.
A[w]	w caracteres para un dato de tipo carácter.
nX	n espacios en blanco.

Figura 1 – Descriptores de entrada/salida de archivos en el lenguaje FORTRAN.

Un número ‘t’ usado como prefijo en cualquier descriptor, con excepción del descriptor de los espacios en blanco, significa que ese descriptor se repite tantas veces como lo indica el número ‘t’.

A.1 FORMATO DAT

En esta sección se explica el formato y el contenido del archivo de especificación de parámetros de fuerzas o *force-fields* de una molécula, conocido como formato .dat. Lo que se describirá del formato es una versión reducida de parámetros la cual se ajusta al alcance del proyecto.

A.1.1 Descripción del archivo de parámetros

El formato del archivo de especificación de parámetros de fuerzas se divide en 10 secciones ordenadas, de las cuales para este proyecto solo se consideran 9 de ellas.

Todas las secciones finalizan cuando se lee una línea en blanco, exceptuando la primera sección y a la sección 3. La primera sección tiene la característica de que está compuesta por una sola línea y, seguidamente sin separación, comienza la segunda sección. En cambio la sección 3 no necesariamente debe estar separada de la siguiente sección por una línea en blanco.

Para cada sección se adjunta el formato de cada línea, el cual fue extraído de [36]. Esta especificación de formato se basa en la nomenclatura para lectura y escritura de archivos del lenguaje FORTRAN. Por más información ver la figura 1 que se encuentra en la introducción del presente anexo.

La lectura del archivo .dat finaliza cuando la cadena de caracteres END es encontrada.

A.1.2 Esquema

El formato .dat tiene el siguiente esquema:

Titulo
Atom Types
Atom Types Hidrophilic
Bond Lengths
Bond Angles
Dihedrals
Improper dihedrals
H-Bond 10-12 potentials
Equivalents potentials atoms
Lennard-Jones potentials

A.1.3 Sección 1 - Título

Esta sección se compone por una sola línea y carece de información relevante. Solamente contiene el título que identifica el conjunto de parámetros. Dicho título no puede superar los 80 caracteres.

Formato: FORMAT(20A4)

Figura 2- Ejemplo correspondiente a la sección 1.

A.1.4 Sección 2 – Tipos de Átomos y Masas

En esta sección se definen los tipos de átomos que se pueden encontrar en la molécula, junto al valor de sus respectivas masas. Además, puede contener un segundo valor para cada tipo de átomo el cual no es tenido en cuenta debido a que no está dentro del alcance de este proyecto.

Cada línea corresponde a un único tipo de átomo donde los primeros dos caracteres indican el nombre del tipo de átomo, los siguientes dos caracteres son espacios en blanco y los siguientes 10 caracteres corresponden a la masa, cuya precisión es de dos decimales.

Formato: FORMAT(A2,2X,F10.2x,f10.2)

Figura 3 - Ejemplo de sección 2.

A.1.5 Sección 3 – Tipos de Átomos hidrofílicos

En esta sección se listan todos los tipos de átomos que son hidrofílicos. Ver definición en el Anexo F.

Puede contener un máximo de 20 tipos de átomos por línea, donde cada uno está separado mediante dos caracteres en blanco. El nombre del tipo abarca dos espacios, aunque el nombre contenga un sólo caracter.

Formato: FORMAT(20(A2,2X))

Figura 4 - Ejemplo de sección 3.

A.1.6 Sección 4 – Enlaces Simples

En esta sección se encuentran los valores de los parámetros del campo de fuerza correspondiente a los enlaces simples. Estos son la constante de fuerza harmónica y la distancia de equilibrio del enlace.

Cada línea contiene, en primer lugar, los nombres de los dos tipos de átomos involucrados en el enlace simple. Estos se encuentran separados a través de un caracter. En los siguientes 10 caracteres se encuentra el valor correspondiente a la constante de fuerza harmónica y en los 10 caracteres restantes el valor correspondiente a la distancia de equilibrio del enlace. Ambos valores tienen una precisión de dos decimales.

Formato: FORMAT(A2,1X,A2,2F10.2)

Figura 5 - Ejemplo de sección 4.

A.1.7 Sección 5 - Ángulos

Esta sección engloba los parámetros correspondientes a los ángulos que determinan al campo de fuerza *Angle bend*. Es similar a la cuarta sección, con la diferencia que para cada línea se encuentran los tres tipos de átomos que conforman ese ángulo.

Al igual que la sección anterior, los parámetros son dos, con la misma precisión y tamaño.

Formato: FORMAT(A2,1X,A2,1X,A2,2F10.2)

Figura 6 - Ejemplo de sección 5.

A.1.8 Sección 6 – Diedros Propios

Esta sección comprende la información de parámetros correspondientes a los ángulos diedros. Cada línea contiene una función diedro donde, en primer lugar, se declaran los nombres de los cuatro tipos de átomos que conforman el ángulo diedro. Cada uno de los nombres está separado entre sí por un caracter. A continuación del último tipo de átomo se encuentra un valor entero de 4 dígitos que indica el factor por el cual se divide la barrera de torsión. Los siguientes tres valores son números reales con un tamaño de 15 dígitos cada uno y con una precisión de 2 decimales. Estos valores se presentan en el siguiente orden: la altura de la barrera de torsión divido por un factor de dos, el ángulo de desplazamiento de la fase y la periodicidad de la barrera de torsión. Pueden existir una cantidad indeterminada de entradas (líneas en el archivo) para un ángulo diedro, de manera correlativa, cuya periodicidad se indica con el valor -1. Sin embargo debe existir uno y solo un diedro con periodicidad igual a 1. Por más información acerca de los ángulos diedros ver Capítulo 2 y el Anexo F.

Por otro lado, se pueden encontrar dos tipos de diedros: generales y específicos. Los diedros generales son aquellos que están definidos por al menos un tipo de átomo genérico, el cual se indica con la letra ‘x’ (por ej. x -x -ct-hc). En cambio los diedros específicos son aquellos que no están definidos por átomos genéricos (por ej. ct-ct-ct-hc).

Los diedros generales deben ubicarse en el archivo antes que todos los diedros específicos. De otra manera los generales sobrescribirán los específicos sin advertencias.

Formato: FORMAT(A2,1X,A2,1X,A2,1X,A2,I4,3F15.2)

Figura 7 - Ejemplo de las secciones 6 y 7.

A.1.9 Sección 7 – Diedros Impropios

La sección de correspondiente a los diedros impropios es idéntica a la sección 6, con la particularidad de que el primer valor, el correspondiente al factor por el cual se divide la barrera de torsión, puede no encontrarse. En este caso, se considera un valor por defecto igual a 1.

A su vez, esta sección puede no estar presente dentro del archivo dat.

A.1.10 Sección 8 – Potenciales H-Bond 10-12

Esta sección no es de interés para el grupo de simulaciones biomoleculares y por lo tanto no fue tenida en cuenta. En caso que esta sección aparezca dentro del archivo, la misma es descartada. Al momento de generar un nuevo archivo de parámetros, esta sección no es generada.

Formato: FORMAT(2X,A2,2X,A2,2x,5F10.2,I2)

A.1.11 Sección 9 – Potenciales equivalentes

La sección 9 define los tipos de átomos equivalentes para los parámetros potenciales de Lennard-Jones (también conocidos como parámetros 6-12). Cada línea de esta sección está compuesta por un máximo de 20 tipos de átomos. Cada uno de ellos ocupa dos caracteres y están separados entre sí por dos espacios en blanco. El primer tipo de átomo de cada línea, es el tipo con el cual el resto de los tipos de átomos, de la misma línea, han de ser equivalentes para la generación de los parámetros potenciales de Lennard-Jones. En el caso de existir más de 19 tipos que deban ser equivalentes a un mismo tipo de átomo, estos deben estar definidos en más de una línea.

Formato: FORMAT(20(A2,2X))

Figura 8 - Ejemplo de la sección de potenciales equivalentes.

A.1.12 Sección 10 – Potenciales de Lennard-Jones

En esta sección se encuentra la descripción de los parámetros potenciales de Lennard-Jones. La primera línea de dicha sección indica el modelo y tipo de potenciales a ser leídos. En primer lugar se encuentra una cadena de 4 caracteres el cual indica el modelo de potenciales. Para el proyecto el modelo requerido por el grupo SBM es MOD4. Luego, separado por 6 espacios en blanco se encuentra el tipo de potenciales a ser leído. Este puede contener los valores: SK, RE o AC.

Formato: FORMAT(A4,6X,A2)

Figura 9 - Ejemplo de la sección 10.

Independientemente del valor de la bandera el resto de las líneas comienzan con dos espacios en blanco, seguidos del tipo de átomo y de dos valores reales que ocupan 10 espacios cada uno, con una precisión de 6 decimales. A continuación se detallan los parámetros a leer según el valor de la bandera.

A.1.12.1 Flag SK – Parámetros Slater-Kirkwood

En este caso se leerán la polarizabilidad atómica para el centro del átomo, el número efectivo de electrones para el centro del átomo y por último el radio de Van der Waals también para el centro del átomo.

Formato: FORMAT(2X,A2,6X,3F10.6)

A.1.12.2 Flag RE – Radios de Van der Waals

En este caso se leerán el radio de Van der Waals del átomo y el potencial de Lennard-Jones.

Formato: FORMAT(2X,A2,6X,2F10.6)

Figura 10 – Ejemplo de Radio de Van der Waals

A.1.12.3 Flag AC – Coeficiente de Lennard-Jones

En este caso se leerán el coeficiente del término de potencia 12 y el coeficiente del término de potencia 6.

Formato: FORMAT(2X,A2,6X,2F10.6)

A.2 FORMATO LIB

En esta sección se explica el formato y el contenido del archivo de librerías de una molécula conocido como .lib.

Una gran desventaja de este formato es que no existe una documentación oficial en donde se describa las características de este formato. Para ello, fue necesario realizar una investigación a partir de diferentes archivos .lib de ejemplo y analizarlo en forma conjunta con el grupo SBM.

A.1.13 Descripción del archivo de librerías

Este archivo se compone de un conjunto de secciones que definen los residuos de la molécula, cada una de ellas está identificada por un cabezal y contienen una cierta cantidad de líneas con los atributos que se definen para esa sección. Cada sección finaliza al encontrarse el cabezal de la siguiente sección. En caso de encontrarse una línea en blanco se da por finalizado el archivo.

Todos los cabezales comienzan con el caracter '!' y todas las líneas donde se definen los atributos comienzan con un espacio en blanco.

El archivo .lib comienza con una sección que lista todos los residuos canónicos en el orden en que se definen en el archivo. La primera línea de la figura 1 se puede apreciar el encabezado de esta sección. Luego en las siguientes líneas, entre comillas y con un

espacio en blanco al inicio, se encuentran los nombres de los residuos mencionados anteriormente.

Figura 11 – Primera sección del archivo .lib

Luego de listar todos los residuos canónicos, se continúa con la definición de cada uno de ellos. Se considera un conjunto de secciones las cuales se repiten para cada residuo y se presentan a continuación.

A.1.14 Esquema

A continuación se presenta un sencillo esquema del formato .lib.

Name res 1
Name res 2
…
Name res N
Residue 1
    Entry atoms
    Entry atomspertinfo
    Entry boundbox
    Entry childsequence
    Entry connect
    Entry connectivity
    Entry hierarchy
    Entry name
    Entry positions
    Entry residueconnect
    Entry residues
    Entry residuesPdbSequenceNumber
    Entry solventcap
    Entry velocities
Residue 2
    …
Residue 3
…
Residue N

A.1.15 Sección atoms

En esta sección se definen los átomos que forman el residuo. Para cada átomo se define un conjunto de propiedades, siendo las más relevantes para SBM el tipo de átomo, el número de elemento correspondiente a la tabla periódica y la carga del átomo.

El formato de esta sección, como se puede ver en la figura 12, se compone de un encabezado del cual se distingue el nombre del residuo (en este caso el residuo ax3) y el

nombre de la sección. Por su parte, los átomos se definen de a uno por línea y cada atributo está separado del siguiente por un espacio en blanco. De todos los atributos que se despliegan por cada línea, son considerados el primero, el segundo, el penúltimo y el último. Cada uno de ellos representa al nombre del átomo, el tipo de átomo, el número de elemento de la tabla periódica y la carga eléctrica del átomo respectivamente.

Figura 12 – Ejemplo de la sección atoms.

A.1.16 Sección atomspertinfo

Esta sección no es relevante para SBM, solo es necesario que cada átomo y tipo de átomo definido en la sección atoms esté presente. En la figura 13 se ilustra un ejemplo de esta sección para el residuo ax3.

Figura 13 – Sección atoms pert info.

A.1.17 Sección boundbox

Esta sección no es de interés para SBM, solo se trata como un atributo más del residuo. Se puede ver un ejemplo de esta sección en la figura 14.

Figura 14 – Sección Boundbox.

A.1.18 Sección childsequence

Ídem a sección anterior, a diferencia que se compone por un solo atributo. Ver ejemplo en figura 15.

Figura 15 – Sección childsequence.

A.1.19 Sección connect

Esta sección define los átomos que son la cabeza y la cola del residuo (ver descripción en el capitulo 4.2.1), el primer atributo es la cabeza y el segundo la cola.

El átomo es identificado a través de un número, donde el número 1 se corresponde al primer átomo definido en la sección atoms, el número 2 corresponde al segundo, y así sucesivamente hasta el último átomo. El número 0 indica que el residuo no tiene cabeza o cola según sea el primero o segundo parámetro respectivamente.

En la figura 16 se muestra un ejemplo para el residuo ax3, en el cual solo tiene cabeza y es el primer residuo que se definió en la sección atoms.

Figura 16 – Ejemplo de la sección connect.

A.1.20 Sección connectivity

En esta sección se definen las conectividades entre los átomos de un residuo. Como se muestra en la figura 17, en cada línea se define una conectividad en donde el primer y segundo parámetro indican los número de los átomos que se enlazan (siguiendo el mismo criterio explicado en la sección anterior, exceptuando el valor 0 que no es válido para esta sección). El tercer parámetro no es de interés para SBM y por ende no fue tenido en cuenta.

Figura 17 – Ejemplo de la sección connectivity.

A.1.21 Sección hierarchy

Esta sección no es de interés para SBM, por lo que dicha información no es tenida en cuenta, la primera línea de esta sección debe ser igual a la que se muestra en la línea 48 de la figura 8. Luego, las siguientes líneas son todas iguales, a excepción del último parámetro el cual se incrementa en 1 para cada línea y deben aparecer tantas veces como átomos se definan en el residuo.

Se puede ver en la figura 18 un ejemplo de esta sección para el residuo ax3, el cual tiene 6 átomos.

Figura 18 – Ejemplo de la sección hierarchy.

A.1.22 Sección name

Esta sección, si bien es redundante, define entre comillas el nombre del residuo. Ver ejemplo en la figura 19.

Figura 19 – Ejemplo de la sección name.

A.1.23 Sección positions

Está sección indica la posición en el espacio para los átomos del residuo. La primera línea define la posición del primer átomo definido en la sección atoms, la segunda línea la del segundo átomo y así sucesivamente en todas las líneas, por lo que la cantidad de líneas es igual a las de la sección atoms.

En la figura 20 se muestra un ejemplo de la sección positions para el residuo ax3.

Figura 20 – Ejemplo de la sección positions.

A.1.24 Resto de las secciones

A continuación de la sección positions, se encuentra un conjunto de secciones que no son de interés para SBM. Si bien algunas de estas secciones permiten ser editadas en la interfaz gráfica de usuario no es relevante describir su significado.

Estas secciones son: residueconnect, residues, residuesPdbSequenceNumber, solventCap y velocities.

A.3 FORMATO PDB

El Protein Data Bank (PDB) es un formato de archivo que representa estructuras macromoleculares, como proteínas y ácidos nucleicos, la cual es comúnmente utilizada por la comunidad científica internacional, profesores y estudiantes. La información que contiene se extrae a partir de datos experimentales e incluye las coordenadas atómicas, información de la estructura primaria y secundaria de la molécula, entre otros datos. [16] [17]

Muchas aplicaciones de visualización molecular interpretan este formato de archivo permitiendo representar a la molécula que en él se describe.

Cada línea del archivo PDB contiene un máximo de 80 caracteres y describe a un átomo. En la figura 1 se muestra el formato de cada línea. Ésta puede comenzar con la palabra clave ATOM, la cual indica que la información desplegada está referida a átomos pertenecientes a proteínas y ácidos nucleicos.

Seguidamente a la palabra clave, se lista información correspondiente al átomo, incluyendo su nombre, su número dentro del archivo PDB, el nombre y el número del residuo al que pertenece y las coordenadas x, y, z, entre otras propiedades. Ver ejemplo en la figura 21.

Figura 21 – Ejemplo de un archivo PDB.

A continuación se presenta el formato de cada línea siguiendo la especificación del lenguaje FORTRAN para describir la lectura y escritura de archivos.

FORMAT(6A1,I5,1X,A4,A1,A3,1X,A1,I4,A1,3X,3F8.3,2F6.2,1X,I3)

Se debe aclarar que no todas las columnas de cada línea del archivo deben estar ocupadas con datos, puede haber espacios en blanco respetando siempre los lugares que ocupa cada columna de datos [17]. A continuación se muestra una tabla detallando la información de cada línea del archivo y el significado de los valores que ella contiene.

Column	Content

```
 1-6    'ATOM' o algún otro identificador.
 7-11   Número de secuencia del átomo.
13-16   Nombre del átomo, según el estándar IUPAC.
  17    Indicador de ubicación alterna, indicado por A, B o C.
18-20   Nombre del residuo, según estándar IUPAC.
  22    Identificador de cadena.
23-26   Número de secuencia del residuo
  27    Código para insertar residuos
31-38   coordenada X (en Angstroms)
39-46   coordenada Y (en Angstroms)
47-54   coordenada Z (en Angstroms)
55-60   Ocupación
61-66   factor de Temperatura
```

Los archivos en este formato contienen diversa información referente a la molécula. Para este proyecto solamente es tenida en cuenta la relativa al identificador 'ATOM'.

Los datos relevantes en cada línea corresponden a: número de secuencia del átomo (identificador del átomo dentro de la molécula), el nombre del átomo, nombre del residuo al que pertenece, número de secuencia del residuo (identificador del residuo) y las coordenadas x y z del átomo.

El fin del archivo está marcado por la palabra clave 'END', una vez encontrada esta palabra se finaliza con la lectura del PDB.

A.1.25 Esquema

El formato del archivo es el siguiente:

Atom 1
Atom 2
…
Atom n
END

A.4 FORMATO TRANSF

En esta sección se describe el formato del archivo de transformación, el cual se genera al realizar el pasaje *All-Atom* → *Coarse-Grain*. Recordando conceptos, definimos los siguientes términos:

- Súper Átomo: es un conjunto de átomos condensados en uno solo. Comparte las mismas propiedades que un átomo común.

- Transformación: es una función que define el pasaje de un conjunto de átomos de un residuo a Súper Átomos. También se redefine la conectividad entre ellos.

Este archivo contiene la información necesaria y suficiente para poder hacer el pasaje inverso, o sea, *Coarse-Grain* → *All-Atom* para todos los residuos de la molécula.

El formato consiste en bloques que se repiten, donde cada bloque contiene sub-ítems.

Primero se pone una lista de todos los nombres de los residuos canónicos a los cuales se les definió una transformación.

Por cada residuo canónico, se ponen los nombres de todas las transformaciones que se definieron para ese residuo canónico.

Por cada transformación se pone en primer lugar una lista con los identificadores de las instancias de residuo a las cuales se le aplica dicha transformación. Luego se ponen los nombres de los súper átomos que forman a esa transformación.

Por cada súper átomo se ponen los nombres de los átomos a los cuales está condensando y el tipo de posición del SAT (esto es: *Geometric Center*, *Center Mass* o *Other position*)

Por cada átomo se pone, un * en caso que dicho átomo participe en la conformación de la posición del súper átomo. Luego se listan los identificadores de átomos que se condensan para cada instancia de residuo. Esto permite recuperar los identificadores de los átomos en la conversión *Coarse-Grain* → *All-Atom*.

A.1.26 Esquema

Luego de la explicación de cada sección y subsección, el esquema del archivo es el siguiente:

```
Canonical Residue A
        Transformation A1
                Applied to: 2, 3, 6, 12, 17…
        SAT A11
                At1
                        Id 1
                        Id 2
                        …
                        Id n
                At2
                …
                Atn
                Position
        SAT A12
        …..
                …..
        SAT A1n
        Transformation A2
        Transformation A3
        ……
        ……
        Transformation An
```

Canonical Residue B
Canonical Residue C
….
….
Canonical Residue N

A.2 Anexo B - Distintas representaciones moleculares en RASMOL, Hyperchem, JMol y VMD

Como se mencionó en el Capítulo 3 Estado del Arte, todos los programas de visualización molecular ofrecen distintas representaciones visuales para una misma molécula como forma de obtener información sobre diferentes propiedades químicas y estructurales.

A continuación se ilustrará la visualización de algunas representaciones tridimensionales que se mencionaron en dicho capítulo. Estas descripciones no son exhaustivas sino ejemplos de visualizaciones. Se describirán las representaciones *wireframe* y *backbone* para el programa RASMOL, *spacefill* para el programa JMol y la representación de *ribbons* a través de VMD.

B Modo Wireframe

La imagen en la representación *wireframe* que se muestra en la figura 1 pertenece al programa RASMOL [37]. Esta es una representación en la cual los enlaces se representan con cilindros y los átomos como esferas. Los átomos son coloreados de acuerdo al tipo de átomo que representa [40]. Es muy usada en ámbitos de enseñanza porque se pueden identificar diferentes conformaciones o enlaces que forman distintos tipos de átomos, sobretodo cuando la molécula está formada por algunos átomos. Es importante destacar que esto es una abstracción diseñada para el mejor entendimiento de la estructura molecular. Los átomos no son esferas y los enlaces no son cilindros, solamente se indica la posición de un átomo y la existencia o no existencia de enlaces entre ellos.

Figura 1 – Representación *wireframe* en RASMOL de la molécula ARNt.

C Modo Backbone

El modo *backbone* representa el esqueleto de una molécula como una serie de segmentos cilíndricos conectados entre los átomos de carbonos alfa adyacentes en cada aminoácido dentro una cadena de polipéptidos [38]. Esta representación es también conocida como trace o de carbonos-alfa. En la figura 2 se puede ver la representación *backbone* en RASMOL para la molécula ARNt.

Figura 2 – Representación *backbone* en RASMOL de la molécula ARNt.

D Modo Spacefill

La representación *spacefill* genera una superficie lisa basándose en los radios de Van der Waals [39]. No se muestran los enlaces ni tampoco se incluyen los átomos que

estén encerrados dentro de Van der Waals de otros átomos. Esta representación es también conocida como VDW (Van der Waals). En la figura 3 se puede observar una representación VDW del ácido acético realizada en el programa de visualización JMol [39]. Esta tipo de visualización es útil para mostrar el espacio que ocuparía hipotéticamente la molécula, ya que cada átomo se muestra como una esfera con el radio de VDW.

Figura 3 – Ácido acético representado con esferas de van der Waals.

E Modo Ribbons

La representación *ribbons* es muy útil para visualizar las estructuras secundarias de una molécula. Esta estructura secundaria [80], se representa a través de cintas planas o tubos las cuales hacen referencia a la cadena principal de átomos en las proteínas (también llamada carbono α) y en las cadenas de enlaces fosfodiéster en ARN/ADN [38]. La figura 4 muestra dos representaciones *ribbons* realizadas en el programa de visualización VMD (ver Capítulo 3).

Figura 4 – *Ribbons* renderizados en VMD. Imágenes extraídas de [7] y [8] respectivamente.

A.3 Anexo C – Estudio de herramientas gráficas para la construcción de software de visualización molecular

Este anexo tiene el propósito de profundizar aspectos no abordados en el capítulo 3 de estado del arte. Aquí se describen diferentes bibliotecas para la construcción de software de visualización molecular y bibliotecas para el desarrollo de interfaces gráficas. También se mencionan las diferentes características, ventajas y desventajas de lenguajes de programación que se tuvieron en cuenta, en donde a partir de esa investigación, decidimos cuál era el lenguaje que más se adecuaba para el desarrollo del proyecto de grado.

A.4 Bibliotecas para construir software de visualización molecular

La mayoría de las aplicaciones de visualización utilizan o están construidas a partir de bibliotecas que proveen ciertas funcionalidades gráficas, de modo de ahorrar tiempo en implementar cosas que ya se han hecho. Esta fue una de las herramientas que fueron evaluadas por nosotros para tomar la decisión de cómo implementar la aplicación final.

Existen diversas bibliotecas para la construcción y desarrollo de software que requieren diferentes visualizaciones gráficas. En general, por lo investigado, estas bibliotecas son útiles para analizar un conjunto de datos (numéricos fundamentalmente) que están almacenados en algún formato en particular. A partir de estos datos se pueden realizar diferentes tipos análisis visual, ya sea en forma de gráficos en dos o tres dimensiones, diversos tipos de visualizaciones que reflejan la evolución o el estado de cierto parámetro o propiedad investigada o la exploración de fenómenos donde los cambios sean dinámicos.

Las técnicas más comunes de visualización son las de isosuperficies [64], de despliegue de volúmenes (conocida como volume rendering) [65], en rebanadas (slices) [71], usando la técnica de ray tracing [66] [67] y mediante técnicas no fotorealistas entre otras [62]. Se explicarán las diferentes características que ofrecen cuatro bibliotecas en el siguiente orden: SDL, OpenDX, VisAD y VTK. Las bibliotecas seleccionadas, tienen la característica en común de que son todas multiplataforma. Recordar que éste fue uno de los requerimientos de los clientes del proyecto de grado.

A.4.1 SDL – Simple DirectMedia Layer

SDL es un conjunto de bibliotecas multiplataforma y de código abierto, que ofrecen funciones básicas para dibujar en 2D, efectos de sonido, música, carga y gestión de imágenes orientado al desarrollo de videojuegos, emuladores, demos y aplicaciones multimedia. Esta biblioteca está desarrollada en C y soporta la gestión de concurrencia a nivel de procesos a través de hilos y semáforos binarios para la sincronización. Además, permite interactuar con diferentes periféricos o subsistemas, ya que oculta la forma de comunicación con estos [41].

Las principales desventajas que se pueden encontrar son que la documentación para el programador es escasa y que si bien SDL posee directivas básicas para dibujar en 2D, no permite el diseño de aplicaciones 3D con funciones propias, sino que para lograr esto se debe complementar con otras bibliotecas como OpenGL o DirectX.

A.4.2 OpenDX – Open Data Explorer

Es un programa de visualización de uso general dedicado al análisis de datos, especialmente de los obtenidos a partir de simulaciones que generan básicamente muchos datos numéricos. Esta biblioteca es multiplataforma, de código abierto y se puede integrar con diferentes lenguajes como Python y Tcl/Tk. Posee una interfaz gráfica, la cual brinda un conjunto de herramientas de GNU [43] (como *autoconf*, *make* y *gcc*), para desarrollar aplicaciones especializadas en visualización y análisis de conjuntos de datos. Además, permite la manipulación, transformación, procesamiento, renderizado y animación de imágenes [42] [11].

OpenDX fue creada en 1991 por la comunidad de visualización IBM Data Explorer con el objetivo de permitir la visualización de casi cualquier formato de archivos de entrada que se le quiera proveer. Para esto, se desarrolló un modelo general que unifica el manejo de archivos e implementa operaciones y extensiones definidas por el usuario para describir la estructura de los mismos. Tiene un diseño para trabajar en un ambiente cliente/servidor y para ejecutarse en paralelo en ambientes multiprocesadores de memoria compartida. También permite la creación de programas visuales mediante un editor de programas visuales (*Visual Program Editor*).

El editor consiste en uno o más campos en donde se colocan y unen varios componentes. Esto es similar a crear un diagrama de flujo de datos que se conocen como redes. La figura 1 muestra la interfaz gráfica del editor de programas visuales y un ejemplo de cómo se codifica un programa en el editor visual de OpenDX [44].

Figura 1 – Ejemplo de la interfaz para programación visual de OpenDX.

OpenDX se puede utilizar por tres principales tipos de usuarios:

1. El usuario que desea visualizar los datos, es decir, el usuario final.

2. El usuario que, teniendo los datos, utiliza las herramientas para elaborar la visualización.

3. El programador, que puede implementar diferentes módulos para la creación o el procesamiento de los datos a visualizar.

Las principales desventajas que se pueden enumerar son que el formato y la cantidad de datos debe ser conocido, y el proceso de creación del programa visual es relativamente complejo. Además, no maneja el formato de datos más usado en simulaciones moleculares, como lo es el archivo de formato PDB [16] [17]. Como solución a esto, se podría desarrollar un *metaformato* para que lo reconozca Opendx [74].

A.4.3 VisAD – Visualization for Algorithm Development

VisAd es una biblioteca multiplataforma que es utilizada para la visualización y análisis de datos. Está implementado en Java bajo el paradigma de orientación a objetos y fue desarrollada por William Hibbard, John Anderson y Brian Paul del Space Science and Engineering Center perteneciente a la Universidad de Wisconsin, en el año 1997.

Esta aplicación se considera como un modelo general de datos matemáticos. A su vez, permite un acceso transparente a los archivos, que pueden ser de diferentes formatos y almacenados en diferentes lugares, ya sea en disco, en memoria o en forma remota. Esto último permite construir aplicaciones que colaboren en un entorno heterogéneo a través de la red.

La organización en clases Java de VisAd se pueden clasificar a través de cuatro categorías de objetos: objetos de datos, objetos de visualización, objetos para cálculos y los objetos para la interfaz gráfica de usuario. Los objetos de datos son contenedores de datos numéricos como imágenes, serie de datos, grillas o simplemente números. Estos datos pueden ser representados en 2D, 3D o con animaciones en forma interactiva a través de los objetos de visualización. Los objetos de cómputo o cálculo son usados para cálculos que se basan en los objetos de datos. Por último, los objetos de interfaz de usuario conectan al usuario de la aplicación con el programa en sí mismo a través de botones, campos de texto, tablas y otros elementos gráficos utilizando tanto Java Foundation Classes (JFC) como Swing [46]. Otra forma de desarrollo que permite es la orientada a eventos, con lo cual se puede crear interfaces de usuario más complejas, que puedan aprovechar las capacidades de cómputo del hardware, implementando técnicas de multithreading y de computación distribuida [45].

VisAD puede ser usado dentro de código python a través de una interfaz conocida como Jython (Java based implementation of Python). La ventaja es que los programas desarrollados en Python tienen menos líneas de código que su equivalente desarrollado en Java. Por otro lado, existe una versión desarrollada de esta biblioteca para ser utilizada en el lenguaje C, pero se recomienda que se transforme a la versión en Java, ya que es una versión más estable. Una desventaja de VisAD es que el renderizado se basa en la biblioteca Java3D, en la cual la performance es pobre, debido a la existencia de la máquina virtual de Java.

A.4.4 VTK – The Visualization Toolkit

VTK es una biblioteca código abierto utilizada para la visualización, procesamiento de imágenes y renderizado en 3D. Es multiplataforma, existiendo versiones para los sistemas operativos Windows, Linux, Solaris, Iris y HP entre otras plataformas. Está desarrollado en C++ bajo el paradigma de orientación a objetos. También existen bindings para otros lenguajes de programación como Python, Java y Tcl/Tk.

Esta biblioteca posee una gran cantidad de algoritmos para la visualización y procesamiento imágenes en 2D y 3D. Incluye librerías para el manejo de escalares, vectores, métodos tensoriales, texturas e implementa técnicas avanzadas de modelación como el modelado implícito, reducción de polígonos y triangulaciones de Delaunay entre otras técnicas. El motor gráfico, que es independiente del sistema operativo,

soporta el renderizado en paralelo tanto para sistemas multithreading como de memoria distribuida.

La estructura central de VTK se basa en un pipeline de datos, desde una fuente de información hacia una imagen renderizada en la pantalla. Las principales desventajas son que no posee una interfaz gráfica para el usuario y que tiene un número limitado de clases de visualización de datos [47] [48] [49].

F Bibliotecas para desarrollar interfaces gráficas

Una de las categorizaciones que se pueden hacer de los lenguajes de programación es aquella en la que los divide entre lenguajes compilados e interpretados [50].

Un lenguaje compilado es un lenguaje de programación que se traduce a través de otro programa, llamado compilador. El compilador genera un archivo en código de máquina llamado archivo ejecutable. En general, un programa compilado suele ejecutarse más rápido que el mismo programa desarrollado sobre un lenguaje interpretado. Algunos ejemplos de lenguajes de este tipo son C, Fortran, Pascal, entre otros. Sin embargo, el código no es flexible ya que cada modificación del archivo fuente (el archivo comprensible para los seres humanos, o sea, el archivo a compilar) necesita de la compilación del programa para aplicar los cambios y generar un nuevo archivo ejecutable.

Por su parte, los lenguajes interpretados no precisan ser compilados. Solamente se necesita un intermediario o intérprete entre el código y el sistema operativo subyacente. El intérprete lee las instrucciones y las traduce a instrucciones del sistema. Las ventajas que poseen este tipo de lenguajes son la independencia del sistema operativo y de la plataforma donde se ejecuta el programa. Como consecuencia de esto, las aplicaciones desarrolladas no son ejecutadas por el sistema operativo, sino, por ejemplo, por una máquina virtual que está siendo ejecutada por el sistema operativo. Esto repercute directamente en problemas de performance o desempeño, siendo la ejecución más lenta respecto al mismo programa desarrollado sobre un lenguaje compilado. Algunos lenguajes interpretados o de scripting conocidos son: PHP, HTML, Tcl, Python, etc. Algunos consideran a Java como un lenguaje interpretado, de esta forma lo consideramos nosotros, ya que el código desarrollado en Java es traducido a bytecode (genera un archivo de extensión .class) y después se ejecuta en la JVM (Java Virtual Machine) [52].

Muchos lenguajes de programación no ofrecen la posibilidad de crear interfaces gráficas (GUI) a través del código propio. Pero existen muchas bibliotecas de alto nivel para crear interfaces gráficas que se pueden integrar junto a cierto lenguaje de programación y que funcionan bien en muchas plataformas, aunque este beneficio tiene algunas consecuencias negativas respecto a la flexibilidad. De hecho, para ser multiplataforma deben soportar las características que son genéricas en casi todos los entornos gráficos de los sistemas operativos. Estas bibliotecas pierden en performance y flexibilidad y muchas no permiten llevar el control de cómo son manipuladas las widgets (elementos u objetos de la interfaz gráfica) por el sistema operativo [51]. A continuación se mencionarán algunas características de tres lenguajes de programación que fueron considerados al momento de desarrollar este proyecto. Estos lenguajes son: C++, Java y Tcl/Tk.

A.4.5 Java

Java es un lenguaje de programación orientado a objetos, similar a C++, pero un poco más simple, debido a que carece de algunas características como la sobrecarga de operadores, aritmética de punteros y arreglos multidimensionales [52].

Una de las grandes ventajas es que los programas desarrollados en Java son fácilmente portables, debido a que su desarrollo y ejecución son independientes del sistema en que se realice porque son interpretados por la JVM. Como contrapartida cuando se analiza la eficiencia en la ejecución de la aplicación, ésta es menor a un programa de similar funcionalidad desarrollado en, por ejemplo C++, debido a que la interpretación se hace en la JVM. Este lenguaje posee dos bibliotecas para la creación de interfaces gráficas de usuario GUIs: AWT (Abstract Window Toolkit) y Swing ambos pertenecientes a la API llamada JFC (Java Foundation Classes).

AWT usa los elementos u objetos de interfaz, también conocidos como widgets, más comunes en todos los sistemas operativos que soportan interfaces gráficas. Luego la máquina virtual de java realiza una traducción hacia el sistema operativo en el que se está ejecutando la aplicación, sea sobre un sistema X, Macintosh o Windows. Esta biblioteca tiene algunos problemas: tiene dificultades de compatibilidad en varios sistemas; carece de algunos componentes avanzados como árboles y tablas, además al ejecutarse una aplicación basada en esta biblioteca, la aplicación utiliza muchos recursos del sistema (principalmente memoria RAM) [52].

La otra biblioteca llamada Swing surgió como una extensión para reemplazar, en la construcción de GUIs, a la biblioteca AWT tanto para aplicaciones de escritorio como en applets de Java. Posee varias ventajas respecto a AWT: los widgets son independientes del sistema operativo, la apariencia o "look and feel" de la aplicación se adapta dinámicamente al sistema operativo nativo y a la plataforma en donde se está ejecutando.

A.4.6 Tcl

Tcl (Tool Command Language) es un lenguaje interpretado disponible en muchas plataformas como Windows, Linux y Macintosh. Asociado a éste, se encuentra la biblioteca gráfica Tk la cual es utilizada para desarrollar interfaces gráficas de usuario [52].

Tk puede ser usado no solo por Tcl, sino que también por distintos lenguajes como Python, Perl y Ruby, etc. Existen bindings para estos lenguajes. Este lenguaje tiene la característica de que es interpretado y como consecuencia tiene casi las mismas desventajas que se mencionaron previamente sobre Java.

A.4.7 C++

C++ es un lenguaje que se basa en el paradigma de orientación a objetos que se lanzó al mercado en 1985 y cuyo creador fue Bjarne Stourstrup. Este lenguaje mantiene la mayoría de las características del lenguaje C, pero simplifica el manejo de la memoria y agrega más características como lo es el polimorfismo y la herencia por nombrar algunas [68].

Este lenguaje puede combinarse con bibliotecas o paquetes para la creación de GUIs como lo pueden ser wxWidgets, GTK, Qt, entre muchos otros [52].

Las desventajas que se pueden mencionar de C++ es que no es un lenguaje portable ya que para poder ejecutarse en diferentes plataformas, el código fuente se debe compilar en cada una de ellas. Otro aspecto en que se debe ser cuidadoso es el manejo de punteros, debido a la ocurrencia de alias a la hora de desreferenciar un objeto que estaba siendo apuntado por un puntero, sin antes haber liberado el espacio de memoria que ocupa ese objeto. Esto se conoce en la jerga informática como “dejar memoria colgada” o “memory leaking” en inglés.

G Bibliotecas para desarrollar interfaces gráficas en C++

Dentro de la investigación realizada optamos por desarrollar el proyecto en el lenguaje C++, ya que brinda un mejor desempeño computacional en el renderizado mediante la biblioteca OpenGL. Por otro lado, todos los integrantes del grupo conocen y utilizan este lenguaje de programación y esta biblioteca para el render 3D.

Para poder implementar una interfaz gráfica de usuario que sea amigable y multiplataforma investigamos algunas opciones que existen hoy en día. A continuación se describirán las características de ciertas bibliotecas que permiten el desarrollo de interfaces gráficas de usuario utilizando el lenguaje C++.

A.4.8 wxWidgets

wxWidgets es una biblioteca multiplataforma y de código abierto para el desarrollo de interfaces gráficas implementadas en el lenguaje C++. Esta permite crear interfaces gráficas basadas en las bibliotecas ya existentes en el sistema operativo nativo. Está disponible para diversos sistemas operativos como Windows, MacOS, OpenVMS, OS/2, Linux, Unix y sus variantes y Solaris. También está disponible para plataformas móviles como Microsoft Pocket PC y Palm OS. La biblioteca wxWidgets puede ser utilizada desde otros lenguajes de programación aparte de C++ como Java, Javascript, Perl, Python, Smalltalk y Ruby [53] .

Las aplicaciones que utilizan wxWidgets cuentan con una particularidad: de acuerdo a la plataforma en que se ejecute la aplicación, esta toma el look and feel del sistema operativo donde se está ejecutando. Es decir que la parte de GUI es solo una capa para la API nativa de cada sistema operativo.

Tiene una API orientada a objetos la cual permite otras funcionalidades, además de la construcción de interfaces gráficas, como la generación gráficos 2D y 3D mediante la utilización de OpenGL, el manejo de Bases de Datos (ODBC), el uso de Threads, entre otras [69].

Las razones por las que se podría elegir wxWidgets son, además de las características mencionadas, que cuenta con documentación en Internet, ayuda en línea, foros y diversos tutoriales y libros. También es un sistema que puede estar orientado a eventos, tiene soporte de MDI (Multiple Document Interface) [70], permite crear DLLs para Windows y bibliotecas dinámicas en Unix.

Entre las desventajas que se pueden mencionar es que su diseño orientado objetos no es muy bueno. A veces utiliza muchas macros para realizar ciertas operaciones (como las tablas de eventos), aunque esto hace que la programación sea más fácil, también complica el debugging. Por otro lado, debido a que el framework se comenzó a desarrollar hace bastantes años, no cuenta con soporte para algunas características relativamente nuevas del lenguaje C++ como por ejemplo el manejo de excepciones, y el uso de la biblioteca STL (Standard Template Library).

A.4.9 Qt – Quasar technologies

Qt es una biblioteca multiplataforma para desarrollar GUIs principalmente a través del lenguaje C++. Está disponible para diversos sistemas operativos como Windows, Linux, MacOS X, Solaris, etc. Además existen bindings hacia otros lenguajes como Python (PyQt), C# (Qyoto), Ruby (QtRuby), Java (Qt Jambi), entre otros.

Esta biblioteca fue desarrollada originalmente por la compañía noruega Trolltech y se lanzó al mercado en Mayo de 1995 con la característica de que era software libre para las versiones desarrolladas para GNU/Linux, existiendo una versión comercial bajo la licencia GPL. Luego, a partir del año 2008 pasó a ser parte de la compañía NOKIA y con la versión 4.5 la licencia pasó a ser LGPL, lo que permite desarrollar software de código abierto y propietario sin pagar licencias.

La versión 4 de Qt introdujo cambios importantes, se reescribió nuevamente el código de las versiones anteriores y se agregaron nuevas funcionalidades como el dibujado en 2D y la mejora de las acciones en varias widgets, entre otros cambios.

En lo que respecta al desarrollo, posee su propio IDE llamado QtCreator que se lanzó al mercado en los primeros días de marzo de 2009. Puede utilizarse en varias plataformas como Windows, MasOS X y Linux, en tanto que para otras plataformas hay que recompilar el código fuente del IDE. QtCreator incluye varias facilidades tanto para escribir código como para el desarrollo y diseño de la interfaz gráfica en sí, a través de la herramienta llamada QtDesigner. Además existe documentación de ayuda, contenida dentro de QtCreator llamada QtAssistant, en diversos libros y en foros on-line.

El API de la biblioteca Qt cuenta con métodos para acceder a bases de datos mediante SQL, manejo de archivos, además de las estructuras de datos tradicionales. Posee un manejo de señales propio llamado “signal” y “slots”, con la cual comunica y relaciona acciones entre los diferentes widgets [63] [61]. Por otro lado, posee una herramienta de internacionalización y la posibilidad de programar en forma embebida

con la biblioteca para gráficos en 3D OpenGL, existiendo otras funcionalidades [59] [60] [61]. En la figura 2 muestra el diagrama de la arquitectura base de la plataforma de Qt.

Figura 2 – Arquitectura base de Qt.

Muchas aplicaciones comerciales conocidas están desarrolladas con Qt como Adobe, Google y Google Earth, Skype, entre otras.

Algunas desventajas de esta plataforma es que en las versiones previas a la versión 4.5 existía una doble licencia, comercial y otra para código abierto. A partir de esta versión no es necesario pagar una licencia si se desea desarrollar un software propietario. Por otro lado, Qt-Creator, el IDE de Qt es una aplicación muy reciente. Esto tiene como consecuencia de que al compararlo con otros IDE's que tienen muchos años de existencia, sus funcionalidades no están tan desarrolladas como en, por ejemplo, Visual Studio, Netbeans o Eclipse, por nombrar algunos. Una carencia importante es que no se visualiza en forma clara los valores que están contenidos en distintas colecciones al momento de debuggear con Qt-Creator. Los errores de compilación a veces no son muy claros y esto genera complicaciones al momento del desarrollo. Con las versiones del IDE esto ha ido mejorando paulatinamente. Otra desventaja, si se puede encasillar como tal, es que el código desarrollado en Qt se traduce a código C++ generándose archivos de extensión "moc" (Meta Object Compiler). Un aspecto positivo que esto conlleva es que el código en Qt permite ser compilado en muchas plataformas, sin necesidad de modificarlo. Lo negativo es que ese proceso de traducción y luego compilación y linkedición produce un overhead importante en la etapa de desarrollo.

A.4.10 GTK+ - GIMP ToolKit

GTK+, originalmente conocido como "The GIMP Toolkit", es una herramienta para la creación de interfaces gráficas de usuario y que es compatible con muchas plataformas, tanto GNU/Linux, Windows (32 y 64 bits) y MacOS.

Inicialmente fue creada para desarrollar el programa de manejo de imágenes GIMP (GNU Image Manipulation Program) en 1998, sin embargo actualmente es muy utilizada por muchos otros programas en diferentes sistemas operativos.

Esta biblioteca está desarrollada en C, pero provee bindings para otros lenguajes como C++, Python, C# entre otros. También puede estar embebido en dispositivos móviles como algunos modelos de Nokia y para el proyecto "One Laptop per Child Project". Se rige bajo la licencia LGPL 2.1 de GNU permitiendo desarrollar tanto software libre como propietario sin pagar ninguna licencia [54].

GTK+ está construida sobre la base de varias bibliotecas desarrolladas por el mismo equipo de GTK+ y de GNOME, a continuación se describirán las características de estas:

- **GLib:** Es una biblioteca que se encarga de las instrucciones de bajo nivel, es independiente de la plataforma donde se ejecute y está dentro de la estructura básica de GTK+ y GNOME. Proporciona funciones para el manejo de estructuras de datos para C, interfaces para funcionalidades en tiempo de ejecución como hilos, carga dinámica entre muchas otras [55].

- **Pango:** Es utilizada para el diseño y renderizado de texto, hace hincapié especialmente en la internacionalización de modo que sea fácil el pasaje hacia otros lenguajes. Soporta la mayoría de los sistemas de escritura, como hebreo, árabe, etc [56].

- **Cairo:** Es una biblioteca para el renderizado de gráficos 2D. Está diseñada para aprovechar el hardware gráfico siempre y cuando esté disponible y soporta diferentes tipos de backends como X Window System, Quartz, Win32, PostScript, PDF, etc [57].

- **ATK:** Es una biblioteca que permite crear interfaces que proveen accesibilidad a personas con discapacidad. Pueden usarse distintas herramientas como lupas de aumento, lectores de pantalla, o alternativas a los dispositivos de entrada como el teclado o mouse [56].

- **GTK:** Esta biblioteca es la que realmente contiene los objetos y funciones para crear interfaces gráficas de usuario. Maneja widgets como ventanas, botones, menús, etiquetas, sliders, pestañas, etc [58].

- **GDK:** Actúa como intermediario entre gráficos de bajo nivel y gráficos de alto nivel manejando el renderizado básico como primitivas de gráficos, control del

mouse, manejo de eventos en los widgets y permite la funcionalidad de drag and drop [58].

Una desventaja que se puede mencionar es que GTK+ solo posee funcionalidad para desarrollar interfaces gráficas de usuario y no está lo suficientemente desarrollado en aspectos como bases de datos, redes, etc. Por otro lado, dependiendo el lenguaje que se vaya a desarrollar, hay que considerar que GTK+ está desarrollado en forma estructurada sobre el lenguaje C.

A.5 Anexo D - AMBER

En esta sección se explicará en forma sucinta en qué consiste AMBER, ya que lo mencionamos en algunos pasajes del informe y a alguna de las aplicaciones que son parte de este framework. Además se explica el conjunto de pasos que, según [72], se deben realizar para establecer una simulación de dinámicas moleculares.

Se explicará parte de los programas utilizados por el Grupo de Simulaciones Biomoleculares del Institut Pasteur de Montevideo para realizar sus investigaciones en esta área. Estos programas son Leap y Sander y son parte de AMBER.

Como hemos explicado a lo largo del informe final, los archivos generados por la herramienta desarrollada son utilizados como input en alguno de estos programas. A su vez, los archivos que resultan de la simulación de dinámicas moleculares, archivos de output, son utilizados por nuestra aplicación para efectuar la conversión de la molécula *Coarse-Grain* a *All-Atom*s.

Amber (Assisted Model Building with Energy Refinement) es un conjunto de programas, o Framework, que permiten al usuario llevar a cabo el modelado molecular y simulaciones de dinámicas moleculares, particularmente de biomoléculas o de sistemas biomoleculares, en donde por ejemplo una biomolécula interactúa con agua [71]. Actualmente lo integran más de 50 programas, cada uno de ellos resuelve diferentes problemas que están relacionados al campo de la dinámica molecular.

Leap es un programa que lee y escribe archivos conteniendo la información estructural de la molécula. Recibe tres archivos (*pdb*, *dat* y *lib*) y genera dos archivos que luego pueden ser usados por otros programas, por ejemplo SANDER. El archivo generado, cuya extensión es prmtop, contiene los parámetros del campo de fuerza (o force fields), la topología y los nombres de átomos y de los residuos de la molécula de entrada. El otro archivo que generado contiene las coordenadas de los átomos de la molécula. La extensión de este archivo es inpcrd.

Sander (**S**imulated **A**nnealing with **N**MR-**D**erived **E**nergy **R**estraints) es un programa que minimiza la energía de un sistema molecular entre otros cálculos que realiza [72]. Este programa es el que efectivamente realiza la simulación de dinámicas moleculares (ver definición de dinámica molecular en el Capítulo 2) y genera archivos de extensión *pdb* como resultado de la simulación. La cantidad de archivos obtenidos puede ser considerable. Cada uno de ellos contiene la variación de los vectores posición de cada átomo a lo largo de la simulación. Estos archivos pueden ser utilizados para analizar la trayectoria de los átomos y establecer conclusiones a partir de ello.

H Pasos para realizar una dinámica molecular

Al momento de realizar una dinámica molecular sobre una biomolécula se deben realizar una serie de pasos:

1- Construir un modelo atómico 3D del sistema a simular.

2- Realizar un cálculo inicial de minimización de la energía respecto de las posiciones de los núcleos de los átomos.

3- Simular el comportamiento del sistema a lo largo del tiempo bajo ciertas condiciones (temperatura, presión, volumen).

4- Analizar los resultados obtenidos de la dinámica molecular y relacionarlo con las propiedades a nivel macroscópico.

El siguiente diagrama presentado en la figura 1, ilustra los pasos descriptos anteriormente para establecer una dinámica molecular dentro del framework AMBER:

Figura 1 – secuencia de pasos para realizar una dinámica molecular en AMBER.

En el paso 1 se obtiene y edita la estructura inicial mediante el archivo en formato *pdb*. En el siguiente paso, se preparan los archivos con los parámetros de entrada para realizar la simulación con, por ejemplo, el programa SANDER. Luego se ejecuta la simulación y se generan los archivos con las trayectorias de los átomos. Por último se analizan los resultados estudiando los archivos de las trayectorias generados [72].

A.6 Anexo E – Estándar de colores CPK

El esquema de colores CPK designa a cada tipo de átomo un color de acuerdo al elemento que se corresponde en la tabla periódica de Mendeleiev [74]. Esta convención fue designada por los químicos Robert Corey, Linus Pauling y mejorada por Walter Koltun. [73]. El criterio de designación de los colores para los diferentes elementos químicos se realiza según el color del elemento en su estado puro.

En la figura 1 se muestra un esquema de los colores en donde se agrupan diferentes elementos químicos dentro de cada color.

Figura 1 – Esquema de colores CPK.

En la figura 2 se describe el color en formato RGB (Red, Green, Blue) para cada elemento o grupo de elementos según el esquema CPK [82].

ELEMENTO	COLOR	COLOR RGB
Carbono	Gris claro	[200,200,200]
Oxígeno	Rojo	[240,0,0]
Hidrógeno	Blanco	[255,255,255]
Nitrógeno	Azul claro	[143,143,255]
Azufre	Amarillo azufre	[255,200,50]
Cloro, Boro	Verde	[0,255,0]
Fósforo, Hierro, Bario	Naranja	[255,165,0]
Sodio	Azul	[0,0,255]
Magnesio	Verde Bosque	[34,139,34]
Zn, Cu, Ni, Br	Marrón	[165,42,42]
Ca, Mn, Al, Ti, Cr, Ag	Gris oscuro	[128,128,144]
F, Si, Au	Oro	[218,165,32]
Yodo	Púrpura	[160,32.240]
Litio	Morado Ladrillo	[178,34,34]
Helio	Rosa	[255,192,203]
Desconocido	Rosa oscuro	[255,20,147]

Figura 2 – Descripción del color según el esquema CPK, basándose en el modelo de color RGB.

A.7 Anexo F - Glosario

La siguiente es una lista de términos que se utilizan a lo largo de este documento, el cual intenta explicar en forma muy básica la terminología del área temática que abarca el proyecto de grado, con el objetivo de lograr un mayor entendimiento de la propuesta.

Ácido nucleico (ADN, ARN): Es una molécula que contiene y transmite la información genética, existiendo dos tipos esenciales de ácidos nucleicos: el ácido desoxiribonucleico (ADN) y el ácido ribonucleico (ARN). Está formada por una secuencia unida de nucleótidos (ver definición más adelante). La estructura y función de los ácidos nucleicos están directamente condicionadas por las propiedades las bases nitrogenadas que contiene. Existen 5 tipos de bases nitrogenadas agrupadas en 2 tipos: derivadas de purinas (Adenina, Guanina) y derivadas de pirimidinas (Citosina, Timina y Uracilo) [75].

Alfa hélice: Es un ejemplo del segundo nivel de organización de las proteínas [80] en el cual el esqueleto peptídico esta enrollado, como una estructura en espiral, alrededor de un eje imaginario (organización helicoidal de la cadena peptídica). Ver definición de proteína, péptido y enlace peptídico que se encuentran más adelante en el documento.

Figura 1 - alfa-helice de la calmodulina.

Ángulo Diedro: Figura formada por dos planos que se cortan (figura 2). El tamaño del ángulo diedro se define como el tamaño del ángulo formado entre dos líneas que se cortan (una en cada plano) que son ambas perpendiculares a la arista a lo largo de la cual se cortan los dos planos [78].

Figura 2 – ángulo diedro es el indicado con alfa.

En bioquímica, la unión peptídica entre un Carbono (C), un Oxígeno (O) y un Nitrógeno (N) forman un plano, como se muestra en la figura 3. En un polipéptido, la cadena que forman los carbonos puede rotar libremente alrededor de las uniones N-Cα (ángulo ϕ) y Cα-C (ángulo ψ), que son los dos ángulos diedros que se forman a partir de un carbono-α (se le denomina carbono-α, al carbono de la cadena principal). En la figura 3 extraída de [77], se ilustra dónde está presente el ángulo diedro dentro de un polipéptido.

Figura 3 – se indican los ángulos diedros ϕ y ψ.

Átomo: Fracción más pequeña de un elemento que conserva las propiedades químicas del mismo y no se puede dividir por medio de reacciones químicas. Los átomos de un mismo o distinto elemento se combinan entre sí para formar moléculas. Están formados por un núcleo compuesto de protones y neutrones y en el que se encuentra la casi totalidad de su masa y una "corteza", constituida por electrones en movimiento alrededor del núcleo. El número de electrones es igual al de protones por lo que el

átomo es eléctricamente neutro. Las propiedades químicas de un elemento están determinadas por el número de electrones que posee.

Aminoácido: Es la unidad elemental que forma parte de las proteínas (ver definición de proteínas más adelante). Básicamente la estructura consta de un átomo de Carbono (llamado carbono alfa), un grupo amino ($-NH_2$), un grupo carboxilo (-COOH), un átomo de Hidrógeno y un Radical, tal como se muestra en la figura 4. Los aminoácidos se diferencian unos a otros a través de los radicales, que es una cadena de estructura variable que determina la identidad y las propiedades de cada aminoácido. Aunque existen más de 300 aminoácidos y solamente un subconjunto de 20 se encuentran en las proteínas.

Figura 4 – estructura general de un aminoácido.

Biomolécula: Es una sustancia química constituida por enlaces covalentes (ver definición más adelante) entre átomos los cuales interactúan con otros conjuntos de átomos a través de interacciones físico-químicas como las de Coulomb [79], Van der Waals y otros tipos de interacciones. Más allá de las interacciones intramoleculares (Ver definición de Interacciones electrostáticas e interacciones de Van der Waals en la sección 2.1.2), las biomoléculas interactúan con otras biomoléculas, con iones y con moléculas solventes. Un ejemplo de molécula solvente es la molécula de agua. Para reflejar fielmente estas interacciones, es necesario utilizar un modelo que represente la molécula a escala atómica y además incorporar funciones detalladas para calcular la energía de ese sistema. Este es el modelo llamado *All-Atom*s que es muy utilizado en el estudio de aspectos físico-químicos de muchas biomoléculas [78].

Enlace covalente: Es un tipo de enlace que se forma al compartir electrones entre dos átomos, cada átomo comparte uno, dos o tres electrones. Como consecuencia se forma un enlace simple, doble o triple respectivamente.

Enlace iónico: Es la unión que ocurre entre átomos que poseen diferente carga eléctrica. El ejemplo más conocido es la unión entre un átomo de Sodio (Na) y otro de Cloro (Cl). El sodio cede un electrón al cloro, para que este sea estable a nivel de electrones, en consecuencia el Na al ceder el electrón se carga positivamente (Na+) y el Cl se carga negativamente (Cl-). Ambos átomos al tener cargas opuestas se atraen formando una molécula NaCl llamado cloruro de sodio. A este tipo de enlace se le denomina iónico.

Enlace peptídico: Es la unión entre el grupo carboxilo (COOH) de un aminoácido con el grupo amino (NH_2) de otro aminoácido, formando un enlace CO-NH. Durante este proceso se desprende una molécula de agua, tal como se muestra en la figura 5 [77].

Figura 5 – Formación de un enlace peptídico en donde se libera una molécula de agua.

Hidrofilicidad: Se refiere a la capacidad de una molécula de solvatarse en un medio acuoso (agua). Se le puede denominar como propiedad hidrofílica de una molécula.

Molécula: Es la unidad física de la materia y es la parte más pequeña de una sustancia que conserva sus propiedades [76]. Su estructura está formada por átomos unidos por fuerzas de enlace químicos o estructura en que dos o más átomos se mantienen en una ordenación determinada como consecuencia de fuerzas de atracción originadas por la interacción con una nube envolvente de electrones negativos.

Nucleótido: Es una molécula orgánica compuesta de ácido fosfórico, ribosa y una base nitrogenada [81]. Es la unidad básica que forma parte de la molécula de ADN.

Péptidos: Es la unión de dos o más aminoácidos que se unen a través de enlaces peptídicos [76]. Los péptidos son estructuras intermedias entre los aminoácidos y las proteínas. Sus propiedades físicas y químicas suelen reflejar en mayor o menor medida las de los aminoácidos que lo componen [77]. La unión de muchos aminoácidos se le llama polipéptido y si el número de aminoácidos supera los 50 se le llama proteína.

Proteína: Es una gran cadena de aminoácidos unidos por medio de enlaces peptídicos. Son compuestos nitrogenados muy difundidos en la naturaleza. Sus propiedades dependen de su composición, estructura y de la secuencia de aminoácidos que contiene. Las proteínas son un ejemplo de macronutriente, al igual que los carbohidratos, y las grasas.

Residuo: En el contexto de las proteínas, un residuo se refiere a cada aminoácido como una unidad en la cadena polipeptídica. En el contexto del ADN/ARN un residuo se refiere a cada nucleótido (base, azúcar y fosfato) que componen el polímero de ácidos nucleicos.

A.8 Anexo G – Documentación técnica

I Arquitectura detallada

En esta sección se presenta un diagrama extendido de la arquitectura propuesta para la aplicación *CG Manager*. Como se explicó en la sección 4.2.3 de la documentación principal y tal como se muestra en la figura 1, la arquitectura se compone de cuatro componentes en donde cada uno de ellos se encarga de ciertas funcionalidades del sistema. De esta manera se modulariza la aplicación y se trata de que sea fácilmente la extensión y el mantenimiento del código para incorporar nuevas funcionalidades o modificar las existentes.

Figura 1 – Esquema de la arquitectura.

El componente *GUI* se encarga del manejo de los eventos en la interfaz de usuario. Está formado por dos componentes: *UserInterface* y *Canvas*. El componente *UserInterface* se encarga de capturar las acciones que el usuario realiza en los diferentes widgets de la interfaz gráfica y comunicar las modificaciones al componente *GuiHandler*. Por su parte *Canvas*, se encarga de capturar los eventos que ocurren dentro del área de dibujado de residuos y átomos. Con el objetivo de mantener una coherencia

entre los elementos que se seleccionan directamente en el *Canvas* o en los widgets, es que existe una comunicación a través del envío de señales entre estos 2 componentes.

El componente *GuiHandler* es un único punto de acceso para realizar las funcionalidades directamente relacionadas con la molécula. De este modo se independiza la interfaz gráfica de usuario con el modelo de dominio que representa a la molécula, proveyendo *GuiHandler* un conjunto de funciones que son modificadoras del estado de la molécula. Por otro lado, se delega la responsabilidad del manejo de archivos al componente *FileHandler*.

Como se mencionó en la sección 4.2.3, el componente *Molecule* se encarga de proveer funcionalidades para modificar la molécula cargada en memoria. Por su parte el componente *FileHandler* es el encargado del manejo de los diferentes tipos de archivos, tanto en su lectura como en la escritura, delegando la responsabilidad a las clases desarrolladas que conocen cada formato de archivo.

J Descripción extendida de las etiquetas de la GUI

A continuación se describe brevemente cada una de las etiquetas mencionadas que se ilustran en la figura 13 del capítulo 4.

Etiqueta *Molecule*: tal como se ilustra en la figura 2, dentro de esta etiqueta se encuentran 4 listas: "Residues" es la lista que contiene a los residuos canónicos definidos en el archivo *.lib*;

"Transformations" es la lista de transformaciones creadas para el residuo canónico que se encuentra seleccionado;

"Residue Atoms" es el conjunto de átomos que pertenecen a un residuo canónico y por último "Transf Atoms" es el conjunto de súperatomos que pertenecen a una transformación que se selecciona de la lista correspondiente.

Luego existen dos botones para definir o eliminar transformaciones, un *combobox* que contiene la lista de tipos de átomos definidos en el archivo *dat* y un botón para poder editar el *atomType* seleccionado en el combo. Debajo de este combo hay una tabla que muestra los potenciales de todos los tipos de átomos.

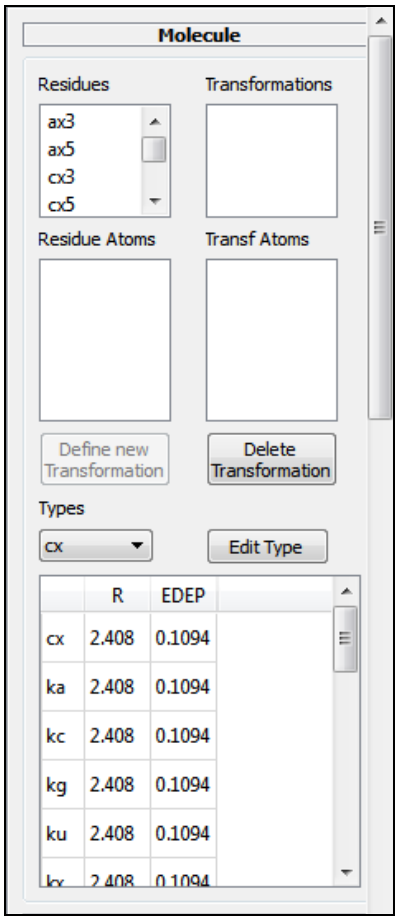


Figura 2 – Pestaña con los datos relativos a la molécula.

Etiqueta *Residue*: dentro de esta etiqueta se agrupa esencialmente información contenida en el archivo *.lib* y que está relacionada con los residuos canónicos. Cuando se selecciona un residuo canónico se cargan las tablas y listas pertenecientes a esta etiqueta conteniendo la información de ese residuo. La información detallada de esta etiqueta se puede consultar en el manual de usuario.

Etiqueta Atom: al seleccionar un átomo o SAT de la lista correspondiente o al seleccionar directamente en el *Canvas*, se carga la información del átomo dentro de esta etiqueta. Tal como se muestra en la figura 3, las propiedades mostradas son: el nombre, el tipo de átomo que tiene asociado, la carga e información extra llamada *atom entry*.

Existe un botón "New" que tiene la funcionalidad de crear nuevos tipos de átomos. Información más detallada de esta etiqueta se encuentra en el manual de usuario, en donde además se comentan algunos parámetros extras en el caso que se seleccione un SAT.

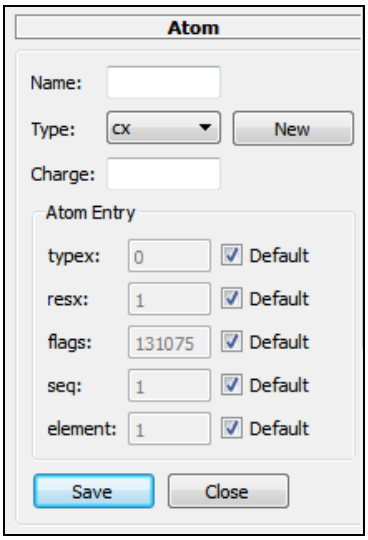


Figura 3 - Pestaña con datos relativos al átomo seleccionado.

K Descripción de las funcionalidades de CG Manager

A continuación se realiza una descripción detallada de las operaciones más relevantes que brinda la aplicación *CG Manager*. Los pasos para efectuar las operaciones se mencionan en el Anexo H, Manual de Usuario.

A.8.1 Cargar Molécula

Al iniciar la aplicación se debe decidir si iniciar una nueva definición de transformaciones o continuar con una definición previamente establecida. *Cargar Molécula* es considerada para el primer caso.

Se deben elegir los archivos de extensión dat, *lib* y *pdb* apropiados, que correspondan a una molécula en la representación *All-Atom*. Luego de seleccionados los archivos se procede a leerlos, construyendo la molécula según el modelo de dominio presentado en la figura 9 del Capítulo 4. Para la construcción de la molécula se consideran los archivos *dat* y lib. El archivo *pdb* contiene la información de las instancias de residuos que conforman a la molécula en cuestión.

En caso de ejecutar la operación correspondiente, se carga donde corresponda la información en la GUI, y se está listo para proceder con las demás operaciones del sistema.

A.8.2 Cargar molécula con una transformación

Esta operación es una extensión de la anteriormente descripta, pues no solamente utiliza los archivos *dat*, *lib* y *pdb*, sino que es utilizado el archivo que describe a las transformaciones, cuya extensión es *transf*.

El objetivo es continuar con una sesión establecida previamente, en la cual se definieron algunas transformaciones para la molécula cargada, pero a la cual todavía no se le ha realizado la conversión *All-Atom* → *Coarse-Grain*.

A.8.3 Crear diedros

Se definen nuevos diedros estableciendo los valores de sus propiedades y se agrega a la biblioteca que contiene información general de los diedros.

A.8.4 Crear tipo de átomo

Al crear un nuevo SAT, puede ocurrir que se requiera asociarlo a un tipo de átomo que no exista en la molécula original, para ello es necesario crear un nuevo tipo de átomo.

El diálogo relativo a los tipos de átomos aparece al presionar el botón de "Edit" o al crear un nuevo tipo de átomo, que se encuentra dentro de la etiqueta *Atom*. Las propiedades que se pueden editar son el nombre, la masa, si es equivalente a otro tipo de átomo y edición de los parámetros potenciales.

Como se explicó en otras secciones del documento, existen tres tipos de parámetros potenciales según las ecuaciones de Lennard-Jones. Los tres tipos se representan a través de las siglas SK, RE y AC que indican Slater-Kirkwood, potenciales de Van der Waals y potenciales 6-12 respectivamente. En lo que respecta a la implementación del proyecto de grado, el tipo de parámetros potenciales es determinado a través del archivo *dat*, en el bloque del archivo que corresponde a los parámetros potenciales.

El concepto de tipo de átomo equivalente significa que si un tipo de átomo es equivalente a otro, entonces posee los mismos valores de potenciales que el tipo de átomo equivalente. Este concepto existe por el hecho de no repetir los mismos datos en la sección de potenciales en el archivo *dat*. En lo referente a la implementación de ese concepto nosotros definimos ciertos criterios o reglas de los equivalentes. Los elementos que se cargan en el combo que aparece etiquetado como "*Equivalent*" son llamados claves, ver figura G6 del manual de usuario. Estas claves tienen la

característica de que se pueden editar los parámetros potenciales. Además, las claves no son equivalentes a ningún otro tipo de átomo, permitiendo un solo nivel de equivalencia. Por eso, al editar un tipo de átomo que es clave, en el combo se encuentra como equivalente el elemento vacío, indicando que no es equivalente a otro tipo de átomo.

A.8.5 Crear transformación

Se crea una transformación a partir de la selección de un residuo canónico de la molécula. Para que quede definida, se deben crear SATs, definir conectividades, establecer las propiedades que corresponden al nuevo residuo canónico el cuales una transformación. Opcionalmente se pueden definir los SATs que sean cabeza y/o cola del residuo transformación.

A.8.6 Crear SAT

Dentro de la operación *Crear transformación* se define un superátomo. Éste se crea a partir de un conjunto de átomos pertenecientes al residuo original. Luego se establecen las diferentes propiedades, a saber: el nombre, la carga, el tipo de átomo asociado, la posición del SAT y otras propiedades que no son relevantes.

Uno de los requisitos para crear un SAT es que el usuario debe definir la ubicación del mismo respecto a los átomos que va a condensar. Existen tres ubicaciones posibles:

1 – La posición respecto a uno de los átomos contenidos en él.
2 – El centro de masa de un subconjunto a definir de los átomos contenidos.
3 – El centro geométrico de un subconjunto a definir de los átomos contenidos.

El cálculo para obtener la posición del centro de masa se define en la figura 4:

$$Pos\_CM = \frac{\sum_{i=1}^{N} m_i * p_i}{\sum_{i=1}^{N} m_i}$$

Figura 4 – Fórmula para definir en el centro de masa de un SAT.

Donde N es la cantidad de átomos que se condensan, m_i es la masa del átomo i y p_i es la posición del átomo i.

El cálculo para obtener la posición del centro geométrico se define en la figura 5:

$$Pos\_CG = \frac{\sum_{i=1}^{N} p_i}{N}$$

Figura 5 – Fórmula para definir en el centro de masa de un SAT.

Donde N es la cantidad de átomos que se condensan y p_i es la posición del átomo i.

A.8.7 Definir Conectividad

Dentro de la operación *Crear transformación* se define una conectividad. Se considera una conexión entre dos SATs, para ello se deben elegir dos superátomos.

A.8.8 Definir head

Dentro de la operación *Crear transformación* se define cuál es el SAT que tiene el rol de Head dentro del residuo transformación.

A.8.9 Definir tail

Dentro de la operación *Crear transformación* se define cuál es el SAT que tiene el rol de Tail dentro del residuo transformación.

A.8.10 Guardar el estado de una molécula

Permite guardar las transformaciones definidas para los residuos canónicos de la molécula cargada. También se generan y guardan cada uno de los correspondientes archivos que representan la molécula y las transformaciones según una ruta establecida. Es importante notar que en este caso solamente se guarda la sesión sin aplicar el algoritmo de transformación *All-Atom* → *Coarse-Grain*.

En caso de ejecutar esta operación más de una vez, los archivos se guardan en las rutas establecidas en la primera ejecución de la misma.

A.8.11 Establecer relación funcional entre transformación e instancias de residuos

Relaciona una instancia de residuo de la molécula con una transformación, de modo que al ejecutar el algoritmo de transformación *All-Atom* → *Coarse-Grain*, se transforma la instancia de residuo generando un residuo *Coarse-Grain* según la especificación de la transformación.

A.8.12 Aplicar transformación AA-CG

Permite aplicar las transformaciones definidas sobre las instancias de los residuos de la molécula actual y generar los correspondientes archivos que representan la molécula simplificada, pudiendo definir la ruta donde se guardará cada uno de los archivos.

A.8.13 Aplicar transformación CG-AA

Permite generar instancias de residuos en representación *All-Atom* a partir de residuos *Coarse-Grain*, considerando para ello las transformaciones definidas sobre las instancias de los residuos de la molécula actual, que se encuentra en representación *Coarse-Grain*. Es la función inversa de AA-CG.

L Coordinación *Canvas*-GUI

El componente *GUI* es el encargado de presentar los diferentes elementos de la interfaz gráfica de usuario y de capturar los eventos que el usuario realiza dentro de la misma. Por su parte, el componente *Canvas* se encarga del dibujado de los residuos con sus correspondientes átomos y conectividades, así como de colorear los átomos de acuerdo al número de elemento asociado. También implementa la técnica de *picking*, en la cual el área de dibujado responde a los clicks que hace el usuario directamente con el mouse. Con esto se permite rotar el residuo, reconocer la selección de un átomo y hacer zoom. *Canvas* utiliza las primitivas que brinda la biblioteca de rendering 3D OpenGL. Este módulo se comunica con el componente *GuiHandler* con el objetivo de obtener datos del residuo y de los átomos a dibujar, por ejemplo, las posiciones de los mismos.

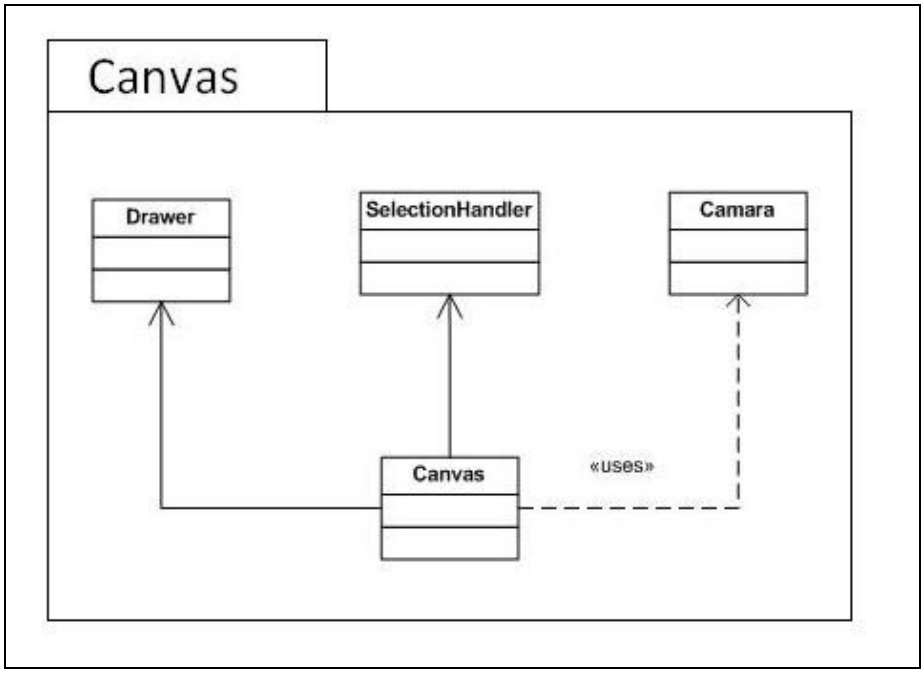


Figura 6 – Diseño del componente *Canvas*.

Como se mencionó en el capítulo 4, el componente *GuiHandler* mantiene el estado de lo que sucede en la interfaz de usuario. De esta manera, al ocurrir un evento en donde se deben mostrar datos, éstos se obtienen de *GuiHandler* a través de diferentes funciones que provee.

Para llevar a cabo la actualización de las acciones que el usuario realiza en la interfaz gráfica de usuario y que se vean reflejadas en el *Canvas*, y viceversa, es que existe una comunicación bidireccional entre el *Canvas* y los elementos de la interfaz de usuario. Esto se lleva a cabo a través de una secuencia de envíos de señales entre el *Canvas* y algunos widgets de la GUI. Un ejemplo de coordinación de las acciones es el caso en que se despliega un residuo canónico y se selecciona un átomo en la lista de átomos. Dentro de la etiqueta Molecule, al seleccionarlo, se pinta el átomo correspondiente en el *Canvas* de forma tal que el usuario visualice cuál átomo está siendo seleccionado para, por ejemplo, modificar algún parámetro del mismo. Asimismo, si el usuario hace *picking* sobre un átomo en el *Canvas*, automáticamente se selecciona el átomo correspondiente en la lista de átomos.

El diseño de la lógica del *Canvas* se basa en un conjunto de estados en donde cada uno de ellos especifica cuáles son las acciones que se permiten efectuar o que se realizan dentro del *Canvas*, como por ejemplo dibujar (ya sean bolas representando átomos o cilindros que representan a las conectividades), seleccionar elementos que están presentes en el *Canvas*, mostrar un residuo canónico en forma completa, entre otros.

Este conjunto de estados está relacionado a través de una máquina de estados que se presenta en la figura 7.

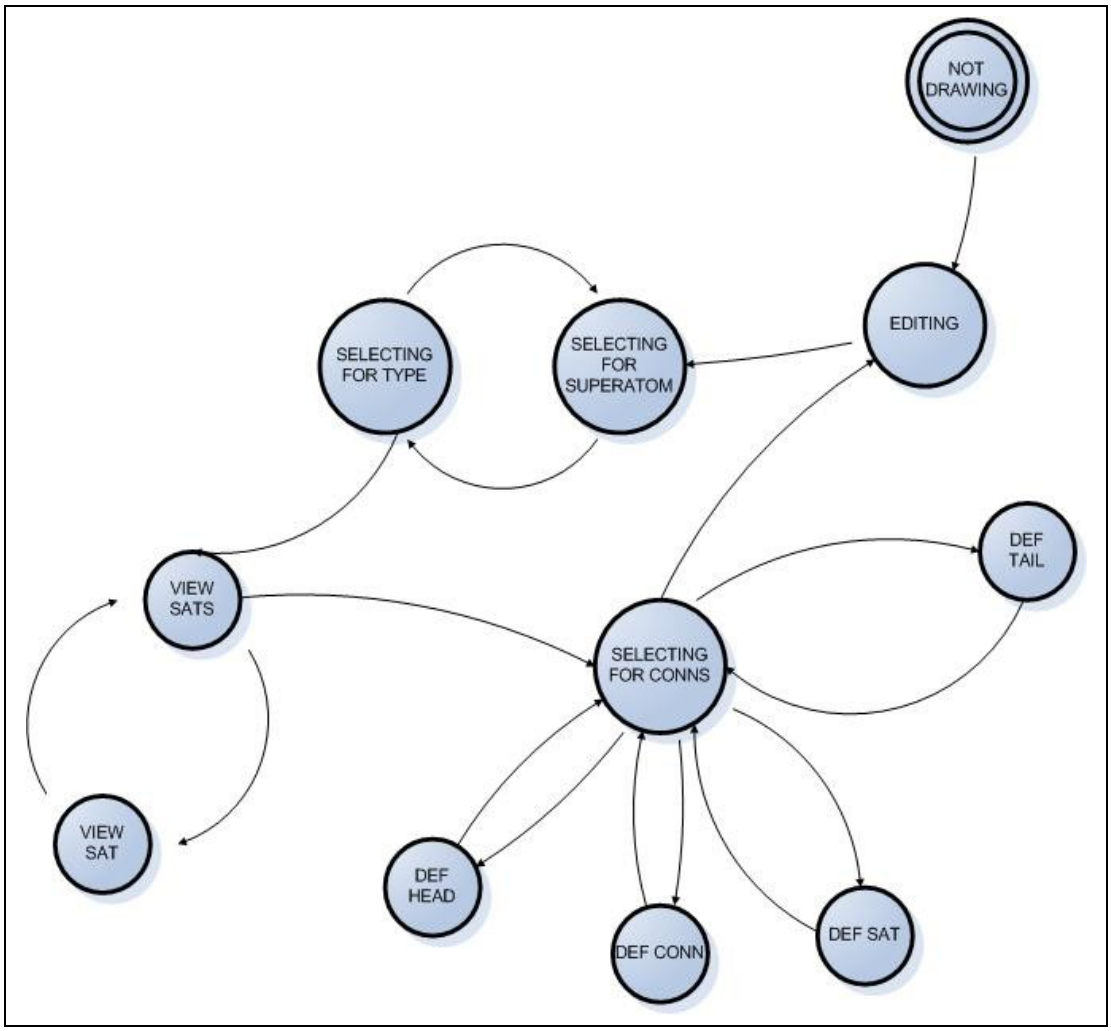


Figura 7 – Diagrama de estados del *Canvas*.

A continuación se describen los estados que definen cómo se comporta el *Canvas*.

- **NOT DRAWING**: Estado inicial en el que el *Canvas* se presenta vacío.

- **EDITING**: Se presenta el residuo seleccionado. Queda habilitado el *picking* individualmente para cada átomo. Al seleccionar un átomo se pueden consultar sus propiedades e inclusive editarlas.

- **SELECTING FOR SUPERATOM**: Estado en el que el usuario debe definir los átomos que conforman un nuevo superátomo. Se despliega el residuo completo y el usuario selecciona cada uno de los átomos que conformarán el nuevo superátomo.

- **SELECTING FOR TYPE**: Estado en el que el usuario define el tipo de posición que tomará el nuevo superátomo. Se despliegan los átomos que formarán el nuevo superátomo y el usuario selecciona el conjunto de átomos que intervienen en la definición de la nueva posición.
- **VIEW SATS**: Estado en el que se despliegan los superátomos que conforman el nuevo residuo *Coarse-Grain*.

- **VIEW SAT**: Estado en el que se despliegan los átomos que conforman el superátomo que esté seleccionado en el estado VIEW SATS.

- **SELECTING FOR CONNS**: Estado en el que se despliegan todos los superátomos que conforman el nuevo residuo dándole la posibilidad al usuario de definir todo lo relativo a conectividades, cabeza y cola del nuevo residuo o propiedades de un superátomo en particular.

- **DEF HEAD**: Estado en el que se está esperando que el usuario seleccione un superátomo que será head del residuo *Coarse-Grain*.

- **DEF TAIL**: Estado en el que se está esperando que el usuario seleccione un superátomo que será tail del residuo *Coarse-Grain*.

- **DEF CONN**: Estado en el que se está esperando que el usuario seleccione una pareja de superátomos definiendo así una nueva conexión entre ellos.

- **DEF SAT**: Estado en el que se está esperando que el usuario modifique las propiedades del superátomo.

7 REFERENCIAS

[1]: Molecular Dynamics Simulation Using Coarse-Grained Model to Study Protein Function and Beyond. M. Takano, J. Higo, H. Nakamura, M. Sasai. Publicado en el congreso de computación evolutiva, Diciembre 2003. ISBN 0-7803-7804-0.

[2]: Dinámica molecular de proteínas y ácidos nucleicos. M. Orozco, J. L. Gelpí, M. Rueda, J. R. Blas. Departamento de Bioquímica y Biología Molecular, Universidad de Barcelona

[3]: Material extraído del curso Modelado Molecular – Introducción práctica a la química computacional. Computational Chemistry and Biology Group, Facultad de Química, Universidad de la República. http://www.ccbg.fq.edu.uy/courses/QFM102/CHP01/deque.html, último acceso febrero 2010.

[4]: Material extraído del curso Estructuras de macromoléculas, J. L. Gelpí, M. Orozco. Departamento de Bioquímica y Biología Molecular, Universidad de Barcelona.

[5]: La Física General, *Ley de Hooke*, Beatriz Alvarenga, Máximo Alvarenga.

[6]: Curso de bioquímica perteneciente a La Universidad de La Laguna, Tenerife, España.

[7]: Clase 20 – Comunicación Visual y efectiva, del curso Interacción Persona Computadora, año 2009, Facultad de Ingeniería, Universidad de la República, Uruguay.

[8]: Interactive computer Graphics, A top down approach with OpenGL, 5th Edition. Edward Angel.

[9]: Computer Graphics Principles and Practice. James D. Foley, second edition, Addison-Wesley, 1996.

[10]: Diccionario de la Real Academia Española, término visualizar en http://rae.es/visualizar, último acceso Enero 2010.

[11]: A Critical history of computer Graphics and Animation http://design.osu.edu/carlson/history/lesson18.html, último acceso Enero 2010.

[12]: Computer Graphics in Undergraduate Computational Science Education. S. Cunningham, A. B. Shiflet. Simposio técnico en la educación de las ciencias de la computación SIGCSE 2003, Reno, Nevada, USA.

[13]: http://www.ub.es/dpfisii/recursos/practicas/ Visualización Molecular, último acceso Setiembre 2008.

[14]: http://www.science.oas.org/rlq/graficas_moleculares.html, Software: Visualización, último acceso Setiembre 2008.

[15]: World index of Biomolecular visualization resources http://molvis.sdsc.edu/visres/molvisfw/titles.jsp, último acceso Setiembre 2008.

[16]: http://www.rcsb.org/pdb/home/home.do, A Resource for Studying Biological Macromolecules, último acceso Febrero 2009.

[17]: http://www.wwpdb.org/, WorldWide Protein Data Bank,último acceso Febrero 2009.

[18]: http://rasmol.org/, Molecular Graphics Visualisation Tool, último acceso Octubre 2008.

[19]: http://www.biorom.uma.es/ayudas/man/RasMol_V2.7.2.1_Manual_es.htm, Manual de Rasmol, último acceso Octubre 2008.

[20]: http://www.marcsaric.de/index.php/, Comparación de las características de diferentes programas de visualización molecular, último acceso Octubre 2008.

[21]: http://www.addlink.es/productos.asp?pid=54, HyperChem para Windows, último acceso Enero 2010.

[22]: http://www.chemistry-software.com/hyperchem/hyperchem7.5.htm, HyperChem 7.5, último acceso Enero 2010.

[23]: http://www.hyper.com, último acceso Enero 2010.

[24]: Material del curso Bioinformática estructural: visualización y diseño asistido por PC de la estructura 3D de moléculas y macromoléculas. Dictado en la Facultad de Ciencias.

[25]: http://www.ks.uiuc.edu/Research/vmd, Sitio web oficial de Visual Molecular Dynamics, último acceso Enero 2010.

[26]: Using VMD – An introductory tutorial. J. Hsin, A. Arkhipov, Y. Yin, J. E. Stone, A. Aksimentiev, J. Cohen, J. Eargle, F. Khalili-Araghi, Z. Luthey-Schulten, M. Sotomayor, E. Tajkhorshid, E. Villa, Y. Wang, D. Wells, D. Wright, and K. Schulten. Universidad de Illinois en Urbana-Champaign, Urbana, Illinois. 2007.

[27]: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2 (5th Edition), Addison-Wesley, 2005.

[28]: http://www.opengl.org/, Sitio web oficial de la biblioteca OpenGl, último acceso Julio 2010.

[29]: http://gpwiki.org/index.php/OpenGL:Tutorials:Picking, Explicación de la técnica de Picking, último acceso Julio 2010.

[30]: http://www.lighthouse3d.com/opengl/picking, Explicación de la técnica de Picking, último acceso Julio 2010.

[31]: http://www.nvidia.com/object/gpu.html, Graphics Processing Unit, último acceso Agosto 2010.

[32]: http://www.ks.uiuc.edu/~johns/, Sitio web personal del investigador John E. Stone, último acceso Agosto 2010.

[33]: http://sites.google.com/site/akohlmey/, Sitio web personal del investigador Axel Kohlmeyer, último acceso Agosto 2010.

[34]: http://agt.cie.uma.es/~mgl/Docencia/Tema5_Apendice.pdf, Explicación de transformación afín, último acceso Setiembre 2010.

[35]: http://www.gromacs.org/, Sitio web oficial de Gromacs, último acceso Setiembre 2010.

[36]: http://ambermd.org/formats.html#parm.dat, explicación del formato .dat, último acceso Julio 2010.

[37]: http://rnaworld.bio.ku.edu/class/Biol807/tutorial/RNA_World_toc.html, último acceso Enero 2010.

[38]: Material del curso Bioinformática estructural: visualización y diseño asistido por PC de la estructura 3D de moléculas y macromoléculas. Dictado en la Facultad de Ciencias.

[39]: http://jmol.sourceforge.net/screenshots/index.es.html, Sitio web oficial del programa Jmol, último acceso Enero 2010.

[40]: http://personal.cscs.ch/~mvalle/ChemViz/representations/index.html, Data Representation in Chemistry, último acceso Enero 2010.

[41]: www.libsdl.org, Simple DirectMedia Layer, Sitio web oficial de la biblioteca SDL, último acceso Octubre 2008.

[42]: http://www.opendx.org, Sitio web oficial de la biblioteca OpenDx, último acceso Octubre 2008.

[43]: http://www.gnu.org, Sitio web oficial de la comunidad GNU, último acceso Setiembre 2008.

[44]: http://www.phys.ocean.dal.ca/docs/DX_tutorial.html, Tutorial de Data Explorer, último acceso Octubre 2008.

[45]: http://www.ssec.wisc.edu/~billh/visad.html, Sitio web oficial de la biblioteca VisAD, último acceso Febrero 2009.

[46]: A Java and World Wide Web implementation of VisAd. Hibbard, William; Anderson, John; Paul, Brian. Space Science and Engineering Center University of Wisconsin – Madison. Publicado en Advances in Space Research,Volume 22, Issue 11, p. 1583-1589. Año 1997.

[47]: http://www.vtk.org, Sitio web oficial de la biblioteca Vtk, último acceso Octubre 2008.

[48]: QteVtk – a Multi-Platform, Object-Oriented Visualization Environment Extending VTK. Stanislav L. Stoev y Wolfgang Strasser. Computer Science Department, University of Tübingen, Alemania. Publicado en el año 2000.

[49]: Material extraído de http://www.osc.edu/supercomputing/training/vtk. Página web del Ohio Supercomputer Center. Ohio, USA, último acceso Octubre 2008.

[50]: http://es.kioskea.net/contents/langages/langages.php3, Descripción general de los lenguajes de programación, último acceso Octubre 2009.

[51]: http://developer.apple.com/documentation/Porting/Conceptual/PortingUNIX/unix_environments/unix_environments.html, último acceso Febrero 2010.

[52]: Graphical User Environments for Scientific Computing. M. Ashworth, R.J. Allan, C.J. Müller, H.J.J. van Dam, W. Smith, D. Hanlon, B.G. Searle and A.G. Sunderland Computational Science and Engineering Department CCLRC Daresbury Laboratory, Warrington, WA4 4AD, United Kingdom.
Reporte técnico publicado en 2003.

[53]: http://www.wxwidgets.org/, Sitio web oficial de la biblioteca wxWidgets, último acceso Febrero 2009.

[54]: http://www.gtk.org, Sitio web oficial de la biblioteca GTK+ último acceso Febrero 2009.

[55]: http://library.gnome.org, Biblioteca de Documentación de GNOME, último acceso Febrero 2009.

[56]: http://www.pango.org/, Sitio web oficial de la biblioteca Pango, último acceso Febrero 2009.

[57]: http://www.cairographics.org/, Sitio web oficial de la biblioteca Cairo, último acceso Febrero 2009.

[58]: http://library.gnome.org/devel/gtk/, Manual de referencia de la biblioteca GTK+, último acceso Febrero 2009.

[59]: http://qt.nokia.com, Sitio web oficial de la biblioteca Qt, último acceso Octubre 2009.

[60]: The Book of Qt 4: The Art of Building Qt Applications. D. Molkentin. Open Source Press GmbH, Munich, Alemania. 2007. ISBN-10 1-59327-147-6

[61]: C++ GUI Programming with Qt 4. J. Blanchette, M. Summerfield. Prentice Hall. 2006. ISBN-10: 0-13-187249-4

[62]: Visualización científica con OpenDX, D. Suárez Pascual, Cómputo académico UNAM, México. 2006.

[63]: http://doc.trolltech.com/4.6/signalsandslots.html, Explicación del mecanismo Signals/Slots, último acceso Febrero 2010.

[64]: http://www.svi.nl/IsoSurface, SVI Support Wiki Pages, último acceso Julio 2010.

[65]: http://www.cc.gatech.edu/scivis/tutorial/linked/,Scientific Visualization, último acceso Julio 2010.

[66]: http://www.siggraph.org/education/materials/HyperGraph/raytrace/rtrace0.htm, Explicación de la técnica Ray Tracing, último acceso Julio 2010.

[67]:http://www.cidse.itcr.ac.cr/revistamate/ContribucionesV4n22003/jorgemonge/jmonge/pag1.htm, Explicación de la técnica Ray Tracing, último acceso Julio 2010.

[68]: http://www2.research.att.com/~bs/C++.html, sitio web oficial del creador del lenguaje C++ Bjarne Stroustrup, último acceso Julio 2010.

[69]: http://docs.wxwidgets.org/, Documentación de wxWidgets, último acceso Julio 2010.

[70]: http://whatis.techtarget.com/definition/0,,sid9_gci214090,00.html, What is Multiple Document Interface, último acceso Julio 2010.

[71]: http://ambermd.org/, Sitio web oficial de Amber, último acceso Agosto 2010.

[72]: Introduction to the AMBER Molecular Dynamics Package. Ping Lin 2008, perteneciente a Materials Simulation Center, Pennsylvania State University.

[73]: http://life.nthu.edu.tw/~fmhsu/rasframe/COLORS.HTM, Explicación del formato de colores CPK, último acceso Agosto 2010.

[74]: Tabla Periódica de elementos de Mendeleiev.

[75]:http://www.uib.es/facultat/ciencies/prof/josefa.donoso/campus/modulos/modulo1/modulo1_1.htm, Biopolímeros, último acceso Enero 2010.

[76]: http://ehu.es/biomoleculas/cibert.htm, Curso de Biomoléculas, Universidad del País Vasco, último acceso Enero 2010.

[77]: Material del curso de Bioquímica de la Facultad de Agronomía, UdelaR. Elaborado por F. Agius, O. Borsani, P.Díaz, S. Gonnet, P. Irisarri, F. Milnitsky y J. Monza.

[78]: http://www.mathematicsdictionary.com/spanish/vmd/full/d/dihedralangle.htm, Diccionario de Matemático, último acceso Enero 2010.

[79]:http://www.educando.edu.do/sitios/pnc2005/recursos/recursos/ciencias%20de%20la%20naturaleza/fisica/fisica%20con%20ordenador%20ii/elecmagnet/campo_electrico/fuerza/fuerza.htm, explicación de la ley de Coulomb, último acceso Julio 2010.

[80]: http://www.aula21.net/Nutriweb/proteinas.htm, definición de proteínas, último acceso Julio 2010.

[81]: http://enciclopedia.us.es/index.php/Nucle%C3%B3tido, definición de nucleótido, último acceso Julio 2010.

[82]: http://www.umass.edu/microbio/chime/pe_beta/pe/shared/cpk-rgb.htm, explicación del esquema de colores CPK, último acceso Agosto 2010.