Universidad de las Ciencias Informáticas

Facultad 6

Título: **Herramienta de visualización dinámica de simulaciones del proceso de difusión en microfluidos con componentes biológicos.**

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor:

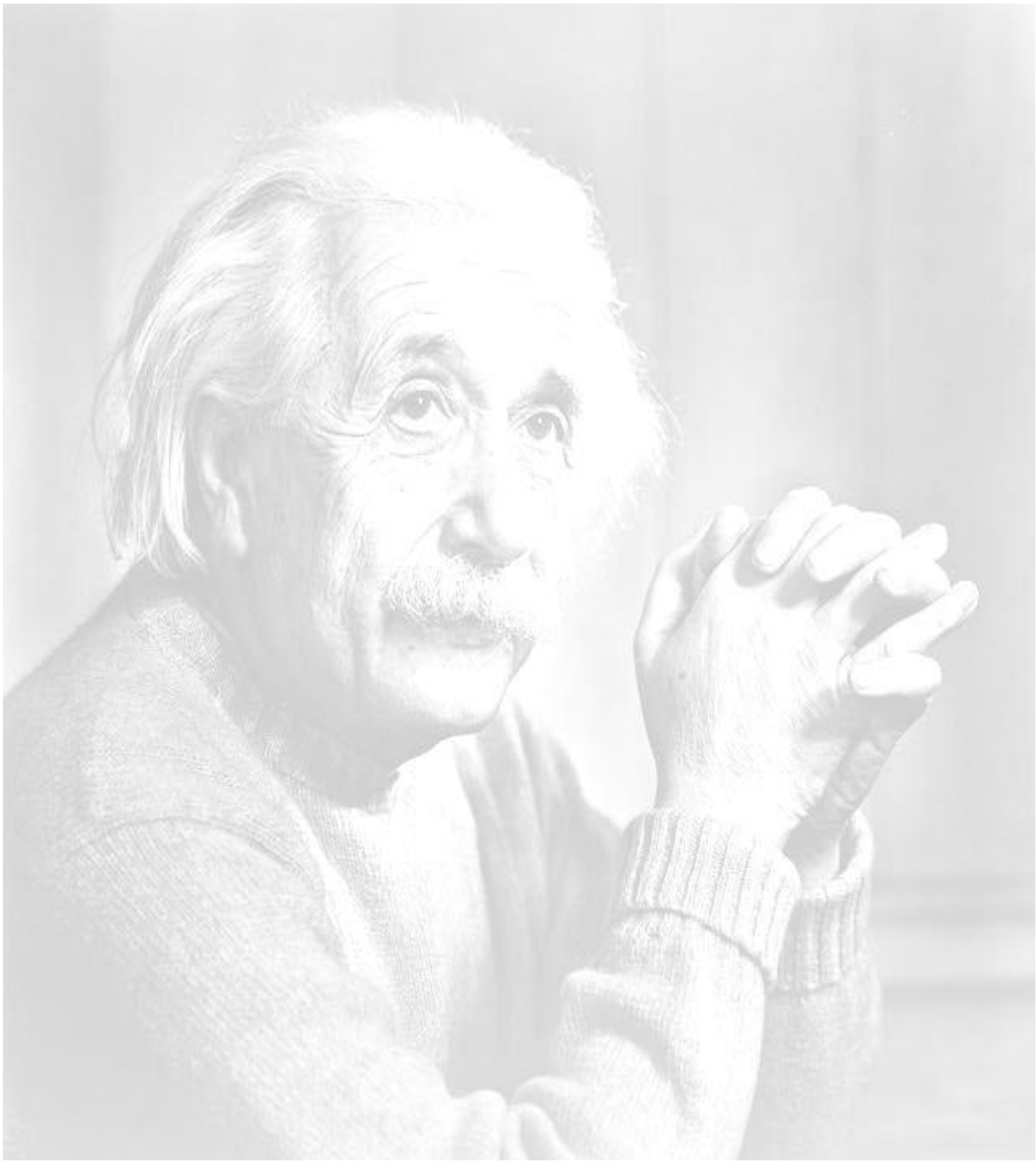
David Alejandro Martínez Pérez

Tutores:

Lic. Edisel Navas Conyedo

Dr. Jorge Gulín González

Ciudad de la Habana, Junio del 2015



Dar ejemplo no es la principal manera de influir sobre los demás; es la única manera.

DECLARACIÓN DE AUTORÍA

Declaro ser autor de la presente tesis y reconozco a la universidad de las ciencias informáticas los derechos patrimoniales de la misma, con carácter exclusivo.

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

David Alejandro Martínez Pérez

_____

Firma del Autor

Tutor: Lic. Edisel Navas Conyedo

_____

Firma del Tutor

Tutor: Dr. Jorge Gulín González

_____

Firma del Tutor

AGRADECIMIENTOS

A mi familia por haberme apoyado a lo largo de estos últimos meses.

A mis tutores Edisel Navas Conyedo y Jorge Gulin González por todo el apoyo que me han dado y por todo el tiempo que me han dedicado.

A mis compañeros del aula que de una forma u otra brindaron todo su apoyo.

A los compañeros pertenecientes al Centro de Estudios de Matemática Computacional que de una forma u otra colaboraron con el desarrollo de mi tesis.

DEDICATORIA

A mi madre por haber estado presente en los momentos más difíciles.

A mi padre por darme todos los concejos en el momento preciso.

A mi hermanita que quiere seguir mis pasos.

A mis amigos que siempre estuvieron ahí en el momento que los necesité.

RESUMEN

Los software para la visualización desempeñan un papel protagónico en la actualidad ya que permiten observar paso a paso la evolución de un sistema en estudio. Las visualizaciones pueden ser estáticas o dinámicas, en el primer caso toda la información necesita ser visible a la vez, lo que puede conducir a algunos problemas incluso si se trabaja con un conjunto pequeño de datos; en el segundo caso se pueden crear diferentes vistas de los mismos datos, las características fundamentales son la animación, la interacción y que la visualización se realiza en tiempo de ejecución. En los sistemas físicos, que se componen por cuerpos que se encuentran en constante interacción y que sus parámetros de comportamiento (posición, velocidad, etc.) son variables en el tiempo, si se quiere realizar una predicción del comportamiento del sistema se hace necesario que la visualización se realice de una manera dinámica debido a que de esta manera se pueden corregir estos parámetros para lograr un comportamiento lo más cercano al real.

En el Centro de Estudios de Matemática Computacional (CEMC) de la facultad 6, se creó en el año 2014 el proyecto Caracterización Computacional de Sistemas Difusivos, centrando sus objetivos en la descripción del proceso de difusión en sistemas que poseen un alto número de componentes correlacionados entre sí. En él, existen software de simulación que arrojan como salida datos que se almacenan en ficheros los cuales son analizados posteriormente en visualizadores especializados, siendo muy difícil la obtención de resultados de análisis parciales. Razón por la cual es necesario el desarrollo de una herramienta que se integre a los simuladores, que utilice un protocolo de comunicación basado en socket para el envío de los datos que van a ser representados en pantalla, y genere a su vez una visualización dinámica que se garantiza por el uso de técnicas de programación multihilo. La solución obtenida mejora los análisis de las simulaciones que se producen, pues no es necesario esperar el fin de una de ellas para ver los resultados, ya que la visualización ocurre en tiempo de ejecución. Esta ventaja permite que los parámetros de simulación puedan ser cambiados, lográndose un mejor entendimiento del sistema que es objeto de análisis.

Palabras claves:

software de visualización, difusión, componentes correlacionados, simuladores, socket, visualización dinámica, tiempo de ejecución, parámetros de visualización

ABSTRACT

The visualization software plays a key role in enabling today and see step by step the evolution of a system under study. The displays can be static or dynamic, in the first case all the information needs to be visible at a time, which can lead to some problems even if you work with a small set of data; in the second case you can create different views of the same data, the key features are animation, interaction and visualization is done at runtime. In physical systems, which are made by bodies that are in constant interaction and their performance parameters (position, speed, etc.) they are variable in time, if you want to make a prediction of the behavior of the system becomes necessary the display is performed in a dynamic way because this way you can correct these parameters to achieve a performance as close to real.

In the Center for the Study of Computational Mathematics (CEMC) faculty 6 the project characterization Diffusive Computational Systems was created in 2014, focusing its objectives in the description of the diffusion process in systems with a high number of components correlated each. In it, there are software simulation output yield data that is stored in files which are then analyzed in specialized viewers, making it very difficult to obtain partial results analysis. Why the development of a tool that integrates the simulators, which use a communication protocol based on socket for sending the data to be represented on screen is necessary, and in turn generate a dynamic visualization that is ensured by the use of multithreading techniques. The resulting solution improves the analysis of the simulations produced, it is not necessary to wait for one to see the results, since the display occurs at runtime. This advantage allows the simulation parameters can be changed, achieving a better understanding of the system is analyzed.

Keywords:

visualization software, diffusion, correlated components, simulators, socket, dynamic visualization, runtime, display settings

Tabla de Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN

En la actualidad la simulación de procesos correspondientes a fenómenos físicos ha tomado un gran auge en la comunidad científica a nivel mundial, debido a que a través de esta se pueden recrear escenas y predecir comportamientos de un sistema sin la necesidad de interactuar directamente con él. Según (Robert E. Shannon, 1975) se entiende por simulación computacional el proceso de diseñar y utilizar un modelo computarizado de un sistema o proceso, y conducir experimentos con este modelo, con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el mismo. Este término se encuentra altamente relacionado con el de visualización dinámica debido a que es muy complicado estudiar un proceso físico teniendo como referencia solamente datos numéricos, también es necesario observar en imágenes lo que ocurre en cada instante de tiempo.

La visualización dinámica es en concepto la proyección de una porción de la información visual en diferentes pequeños intervalos de tiempo, en cada uno de los cuales se muestra una porción diferente, consiguiendo así proyectar toda la información (Cruz, 2012). Este efecto se consigue por lo que se denomina tiempo de retención de la retina, lo cual explica que una imagen dura pequeñas fracciones de segundo en nuestra retina, si se logra presentar todas las porciones de la información visual antes de que termine este tiempo el cerebro procesa todo esto como una sola imagen porque aun retiene las porciones anteriores. Gracias al efecto anterior es posible proyectar una imagen obturándola a más de 24Hz (entiéndase poniéndola y quitándola más de 24 veces por segundo) y ser percibida, por el ojo humano, como una imagen fija. Este efecto es la base de los vídeos, solo que no se presenta la misma imagen únicamente sino que se presenta imágenes de forma simultánea cada una con un pequeño desplazamiento con respecto a la anterior.

En las simulaciones de procesos de difusión en microfluidos con componentes biológicos complejos donde los parámetros de comportamiento son variables en el tiempo, para realizar un análisis profundo de la evolución del sistema es imprescindible poder visualizar el comportamiento de las estructuras espaciales (sistemas de partículas, polímeros, superficies, entre otras) a lo largo de toda la simulación y esto solamente es posible si la visualización se realiza de manera dinámica.

Existen herramientas que complementan a las simulaciones y son los analizadores y los visualizadores, que permiten que se pueda procesar y observar respectivamente cada paso de lo que se intenta recrear. El objetivo de estas como un conjunto es transformar los datos que envían los simuladores de un estado en específico a imágenes y valores numéricos descriptivos que especifiquen lo que está ocurriendo, este envío de datos se realiza a través de un protocolo de comunicación (SOCKET, UDP, SPX, entre otros) si la visualización es dinámica, en otro caso la información de la simulación se guarda en ficheros, para ser analizados posteriormente. De esta

manera el usuario puede interactuar con una aplicación que le permita llegar a conclusiones orientando la visualización a sus intereses.

Actualmente existen numerosas herramientas que permiten la visualización de estructuras espaciales y parámetros de comportamiento, entre ellas se encuentran: *Origin*, que es un entorno para analizar datos y generar gráficos técnicos especialmente pensado para ingenieros y científicos; *VMD* (Visual Molecular Dynamics) un programa de modelamiento molecular y visualización de estructuras que incluye herramientas para trabajar con datos de volumen, secuencias y objetos gráficos arbitrarios como conos o cilindros, representa dinámicamente solo estructuras moleculares; *qtiplot,que es un clon libre de Origin*; y *gnuplot* un programa para generar gráficas de funciones y datos a partir de scripst programados, puede mostrar los resultados en pantalla o en múltiples formatos de imagen. Siendo las dos primeras privativas y las restantes de software libre. Pero ninguna de ellas puede producir las visualizaciones dinámicas que se necesitan para el estudio de los procesos de difusión en microfluidos con componentes biológicos complejos.

En Cuba, en la Universidad de las Ciencias Informáticas a lo largo de los últimos años la tecnología se ha modernizado y la innovación ha tomado un papel fundamental, razón por la cual se han creado numerosos proyectos investigativos. Un ejemplo evidente de este incipiente desarrollo es el Centro de Estudios de Matemática Computacional (CEMC) el cual posee al proyecto Caracterización Computacional de Sistemas Difusivos (CCSD).

Los software de simulación desarrollados en este proyecto se han diseñado para almacenar la información en ficheros que posteriormente son analizados con herramientas de visualización especializadas, por lo que no se pueden obtener resultados de análisis parciales que permitan poder variar los parámetros de simulación antes de que la misma termine, siendo los tiempos de simulación muy largos. Los sistemas de visualización existentes en software de simulación similares que poseen integración para la visualización dinámica no están diseñados para el tratamiento específico en los procesos de difusión en microfluidos con componentes biológicos complejos.

A partir de la situación problemática descrita anteriormente, se enuncia como problema a resolver: ¿Cómo visualizar en tiempo de ejecución las simulaciones del proceso de difusión en microfluidos con componentes biológicos complejos?
Se define como objeto de estudio la visualización $2D^{1}+t^{2}$ y $3D^{3}+t$ de estructuras espaciales y parámetros de comportamiento en microfluidos con estructuras biológicas complejas.

1 Este término describe lo bidimensional, aquello que sólo tiene dos dimensiones.

El campo de acción se enmarca en los sistemas de visualización dinámica por computadora 2D+t y 3D+t.

El objetivo general de la investigación es desarrollar una herramienta de visualización dinámica de simulaciones del proceso de difusión en microfluidos con componentes biológicos complejos integrada a los simuladores desarrollados en el proyecto CCSD que muestre en tiempo de ejecución las estructuras espaciales y parámetros de comportamiento.

Para dar cumplimiento a los objetivos planteados en este trabajo se necesita un grupo de tareas investigativas a las cuales se hará referencia:

1. Análisis del estado del arte sobre los sistemas de visualización 2D+t y 3D+t utilizados en las simulaciones de sistemas físicos.

2. Estudio de la arquitectura y herramientas de software para la simulación de microfluidos con componentes biológicos en el proyecto CCSD.

3. Definición una arquitectura de software y protocolo de comunicación entre los simuladores y la herramienta de visualización a desarrollar.

4. Desarrollo, prueba y validación.

2 Tiempo

3 El término se refiere a la tridimensionalidad, a la forma en que percibimos nuestra realidad.

Estructura del documento:

Capítulo 1. Fundamento teórico. Se muestran los principales lenguajes, conceptos, elementos teóricos, métodos y aplicaciones informáticas existentes del ámbito nacional e internacional necesarias para la comprensión del proceso de visualización de simulaciones del proceso de difusión en microfluidos con componentes biológicos complejos, así como la metodología de desarrollo de software escogida para todo el proceso de desarrollo de la aplicación.

Capítulo 2. Propuesta de solución. Se presentan las principales características de la solución al problema de la investigación: comunicación entre el simulador y el visualizador, su optimización usando programación multihilo, funcionalidades del sistema y arquitectura. Se realiza una descripción detallada de la propuesta de solución.

Capítulo 3. Validación de la solución. Se presenta la validación de los requisitos y las pruebas hechas al software: pruebas de diseño a partir de las métricas Tamaño operacional de clase (TOC, por sus siglas en inglés) y Relaciones entre clases (RC, por sus siglas en inglés) y las pruebas de implementación compuestas por las pruebas de caja blanca y caja negra.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

1.1 Introducción

En este capítulo se muestran los elementos teóricos y conceptos para comprender el proceso de difusión en microfluidos, ejemplos de aplicaciones informáticas usadas actualmente, así como los métodos, la metodología de desarrollo de software, y lenguajes escogidos para obtener una solución informática para el problema de la investigación.

1.2 Definición de términos

Para el estudio del proceso de difusión en microfluidos es necesario tener un conocimiento básico del comportamiento de las estructuras bases a visualizar, porque al conocer sus características podemos predecir cual serán las particularidades del nuevo estado en el que se encontrará el sistema en un período de tiempo determinado y de esta manera se pueden variar los parámetros de simulación para corregir posibles resultados no esperados.

En cuanto a la visualización se pueden definir dos elementos que resultan claves, uno es la información relevante a representar y el otro el modelo de representación esquemática de cada componente.

A continuación se explican los términos fundamentales relacionados con el proceso de difusión en microfluidos para un mejor entendimiento del mismo.

Difusión

En principio, la Teoría de la Difusión se desarrolló para tratar de describir fenómenos como la conducción del calor. El movimiento browniano es un modelo que fue descrito por primera vez por el botánico ingles Robert Brown en 1827 para describir el fenómeno de la difusión. Si trazamos líneas rectas entre dos posiciones distintas en el tiempo de la partícula browniana, se observa que realiza un movimiento desordenado e irregular. Se mueve siguiendo una trayectoria en forma de zig-zag (Jurado, 2007).

En general, la difusión es la propagación espontánea de la materia, calor o momento. Se entiende también por difusión al movimiento de partículas desde una concentración alta hacia una concentración baja.

Las diferentes formas de difusión pueden ser medidas cuantitativamente usando las ecuaciones de difusión, las cuales llevan diferentes nombres de acuerdo con la situación que se presenta en física.

Por ejemplo, la difusión del estado estable biomolecular es gobernado por la Primera Ley de Fick, la difusión del estado estable térmico por la Ley de Fourier y la difusión de los electrones en un campo eléctrico es conducido esencialmente por la Ley de Ohm (Jurado, 2007).

La difusión ocurre como resultado de la Segunda Ley de la Termodinámica, la cual establece que la Entropía o desorden de cualquier sistema cerrado siempre debe incrementarse con el tiempo, porque las sustancias se difunden desde una región de alta concentración hacia regiones de baja concentración, es decir, ellos van de un estado de alto orden a un estado de bajo orden, esto en concordancia con la Segunda Ley de la Termodinámica. Así pues, la difusión es un proceso natural y espontáneo (Mao y Sinnott, 2000).

En el caso de la difusión en microfluidos con componentes biológicos en algunas situaciones particulares presenta una difusión anómala, en ella la varianza crece de una manera distinta a una función lineal y en determinadas condiciones es producida por la alta correlación existente en el movimiento e interacciones entre los componentes del sistema.

$$\langle X^2 \rangle = t^\gamma \quad (1)$$

Ilustración 1. Imagen de una ecuación de la difusión anómala.

El exponente γ (GAMMA), en la ecuación de la ilustración 1, es igual a uno para el caso de difusión normal. Para los vuelos de Levy se cumple que 1< γ <2, la cual es llamada Superdifusión. Si γ = 2 corresponde al movimiento balístico, como en las partículas de una bomba cuando explota, que es el caso donde todos los caminantes se mueven alejándose unos de otros a una velocidad constante. En algunos casos, si γ <1 se llama subdifusion (Metzler y Klafter, 2000).

Sistema de partículas

En mecánica se considera un sistema de partículas como un conjunto de *N* partículas que se mueven por separado, si bien interactúan entre sí y están sometidos a fuerzas externas. Cuando el número de partículas es reducido se puede abordar el problema dinámico analizando cada una por separado. Cuando es elevado, es preciso recurrir a promedios y descripciones colectivas (como la mecánica estadística, la elasticidad o la mecánica de fluidos) (Martin y Serrano, 2014).

Los sistemas se clasifican en abiertos o cerrados. Un sistema cerrado es aquél en el que no entra ni salen partículas del sistema. Por tanto, su masa permanece constante. Un sistema abierto es aquel que permite el paso de partículas (y por tanto masa) a través de los límites del sistema. Entre las fuerzas internas en un sistema estarían por ejemplo, las fuerzas eléctricas de atracción entre las cargas de un sistema de protones y electrones, o la atracción gravitatoria entre las estrellas de una

galaxia. Entre las fuerzas externas figura, por ejemplo, el peso de un sistema de partículas, originado por la atracción de un cuerpo externo como la Tierra (Martin y Serrano, 2014).

Polímeros

Los polímeros se definen como macromoléculas compuestas por una o varias unidades químicas o partículas bases (monómeros) que se enlazan formando una cadena (Vincent, y otros, 2006).

La parte básica de un polímero son los monómeros, estos representan a cada una de las partículas que forman el polímero y son las unidades químicas que se repiten a lo largo de toda la cadena de un polímero, por ejemplo el monómero del polietileno es el etileno, el cual se repite x veces a lo largo de toda la cadena (Vincent, y otros, 2006). En la ilustración 2 se muestra un ejemplo de un polímero.

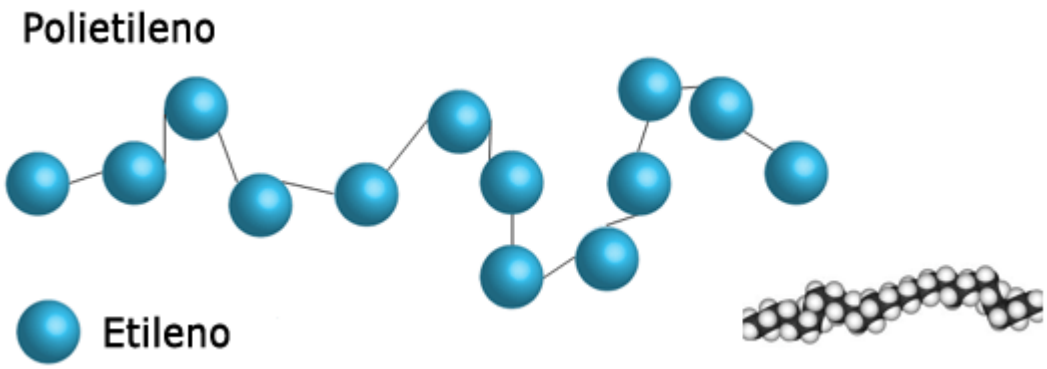


Ilustración 2. Imagen del polímero polietileno.

En función de la repetición o variedad de los monómeros, los polímeros se clasifican en (López, 2004):

- Homopolímero - Se le denomina así al polímero que está formado por el mismo monómero a lo largo de toda su cadena, el polietileno, poliestireno o polipropileno son ejemplos de polímeros pertenecientes a esta familia.

- Copolímero - Se le denomina así al polímero que está formado por al menos 2 monómeros diferentes a lo largo de toda su cadena, el Acrilonitrilo Butadieno Estireno (ABS, por sus siglas en inglés) o el Caucho Estireno-Butadieno (SBR, por sus siglas en inglés) son ejemplos pertenecientes a esta familia.

Mallas Geométricas

Una malla geométrica es una colección de vértices, aristas y caras, que define la forma de un objeto complejo en base a polígonos (2D) y poliedros (3D), es decir, en base a formas simples y conocidas. Se distinguen dos tipos de mallas (Cumsille, 2010):

- Mallas Estructuradas: se componen de celdas del mismo tipo y tamaño, como por ejemplo, rectángulos o triángulos en (2D) y hexaedros o tetraedros (3D).
- Mallas No Estructuradas: se componen de celdas del mismo o distinto tipo pero de tamaños diferentes. Existen en general, algunas zonas con celdas pequeñas y otras con celdas de mayor tamaño.

La ventaja de las mallas estructuradas con respecto a las mallas no estructuradas, es que son más fáciles de generar y de estudiar sus propiedades, sin embargo no permiten modelar problemas o formas complejas. Son de gran utilidad para describir estructuras como membranas celulares, liposomas, glóbulos rojos entre otras. En la ilustración 3 se muestra un mallado del corazón.

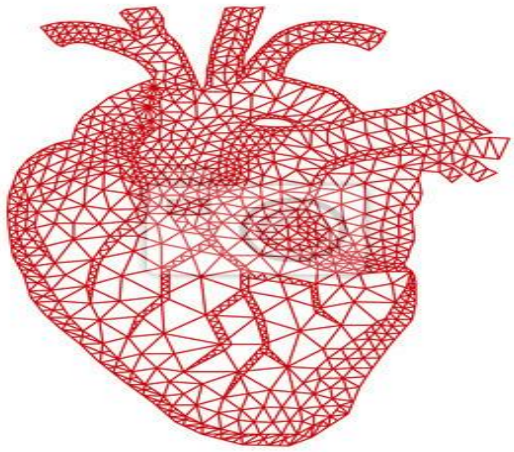


Ilustración 3. Imagen de una malla geométrica.

1.3 Soluciones informáticas existentes

Antes de comenzar con el desarrollo de la herramienta de visualización dinámica es de vital importancia conocer las características fundamentales así como la arquitectura de algunos software similares, debido a que se pueden reutilizar algunas funcionalidades en el caso que estos visualicen estructuras que se asemejen a las representadas en los procesos de difusión en microfluidos.

A continuación se describen las características fundamentales de algunos software diseñados para la visualización de estructuras que son similares a las que se representarán en el visualizador que se desea desarrollar:

Gaussian View

Gaussian View es la interfaz gráfica más avanzada y potente disponible para Gaussian. Con este software, se pueden importar o construir las estructuras moleculares que sean de interés, configura, monitorea y controla en los cálculos de Gaussian, y recupera y muestra los resultados de los cálculos usando una variedad de técnicas gráficas, todo esto sin tener que salir de la aplicación. Incluye muchas características nuevas diseñadas para facilitar el trabajo con grandes sistemas de interés químico (Gaussian View).

Este software es muy útil, pero no permite simular microfluidos, ni tiene una interfaz para la visualización dinámica; el estudio del mismo es motivado por el estudio de su arquitectura por la similitud de los sistemas tratados.

Odissey

Odissey Edición Instructor es un programa orientado a la enseñanza de química. Incluye una biblioteca de compuestos moleculares y un kit de modelos fáciles de seguir para la construcción de casi cualquier sistema químico. Es una herramienta innovadora para cualquier profesor de ciencias que esté interesado en la visualización de la química a nivel molecular (Odissey Edición Instructor).

Entre las funciones y contenidos más destacados se encuentran:

- Simulación de reacciones químicas: elaboración y ruptura de enlaces.
- Evaluar la función: evalúa los modelos creados por el usuario, dando los nombres y determinando su existencia.
- Construir Sólidos: construcción de cientos de estructuras sólidas.
- Biblioteca de compuestos
- Laboratorios Moleculares / Tutoriales / Química Aplicada
- Hojas del alumno: más de 700 preguntas interactivas, visualización basada en el aprendizaje Visualización de forma y reactividad.

Este software brinda muchas opciones para la visualización, y no obstante a estar orientado a la enseñanza, tampoco permite simular microfluidos. Pero en su arquitectura existen elementos que son comunes con el software de visualización dinámica, razón por la cual se hace un estudio del mismo.

Jmol

Jmol es un visor de código abierto de estructuras químicas en 3D. Devuelve una representación tridimensional de una molécula que puede usarse como herramienta de enseñanza o para la investigación, por ejemplo en química y bioquímica. Es software libre y de código abierto, escrito en Java y por ello se puede ejecutar en Windows, Mac OS X, Linux y sistemas Unix. Existe una aplicación independiente y un kit de herramientas de desarrollo que puede integrarse en otras aplicaciones Java.

La particularidad más notable: es una miniaplicación (applet) que se puede integrar en páginas web para mostrar las moléculas de muchas formas. Por ejemplo, las moléculas se pueden mostrar como modelos de "bola y palo", modelos "que llenan el espacio", modelos de "cinta", etc. Jmol es compatible con una amplia gama de formatos de archivo moleculares, incluyendo Protein Data Bank (PDB), archivo de información cristalográfico (CIF), MDL Molfile (mol) y Chemical Markup Language (CML) (Jmol).

Entre sus principales características se encuentran:

- Miniaplicación (applet), aplicación y componente para la integración en sistemas JmolApplet es una miniaplicación para el navegador web que puede integrarse en páginas web. Es ideal para el desarrollo de material docente a través de la web y para bases de datos químicas accesibles por internet.
- Multi-idioma Traducido a numerosos idiomas.
- Vibraciones de sistemas moleculares.
- Respaldo básico para la celdilla unidad y operaciones de simetría.
- Formas esquemáticas para las estructuras secundarias de biomoléculas.
- Distancia
- Ángulo

Este programa realiza una buena representación de las estructuras moleculares, pero tampoco permite realizar simulaciones de microfluidos. La visualización de las estructuras se realiza a través de la carga de ficheros que contienen los datos de lo que se quiere visualizar, por lo que no permite visualizaciones dinámicas. Este software tampoco permite la representación de algunas estructuras que son críticas en los procesos de difusión que son objeto de estudio en el proyecto CCSD como por ejemplo las membranas. Para el desarrollo de la herramienta de visualización se seleccionaran algunas funcionalidades de Jmol como son: el movimiento de la cámara, la iluminación y la opción de guardar lo que se representa en pantalla como una imagen.

1.4 Metodología de desarrollo de software

Las metodologías de desarrollo de software según (Piattini Velthuis, 1996) están definidas como un conjunto de procedimientos, herramientas y un soporte documental que ayuda a los desarrolladores a realizar nuevos software, pero como tal no existe consenso alguno entre los autores sobre el concepto de metodología y por ello no existe una definición general aceptada. Sí hay un acuerdo en considerar a la metodología como un conjunto de pasos y procedimientos que deben seguirse para el desarrollo del software.

Las metodologías de desarrollo de software se pueden agrupar en dos grandes grupos (Piattini Velthuis, 1996):

- **Metodologías tradicionales o pesadas:** hacen mayor énfasis en la planificación y control del proyecto, en especificación precisa de requisitos y modelado, estableciendo estrictamente las actividades involucradas, los roles definidos, los artefactos que se deben producir, las herramientas y la documentación usada (RUP (Rational Unified Procces), MSF (Microsoft Solution Framework), Win-Win Spiral Model, Iconix).

- **Metodologías ágiles o ligeras:** orientadas a la generación de código con ciclos muy cortos de desarrollo manteniendo un proceso incremental, son capaces de permitir cambios en los requisitos de último momento, además el equipo de desarrollo mantiene una comunicación constante con el cliente (Extreme Programming (XP), SCRUM, Crystal Clear, Feature-Driven Development (FDD), Dynamic Systems Development Method (DSDM), Adaptive Software Development (ASD)).

No se opta por el uso de una metodología tradicional ya que en la investigación se hace mayor énfasis en la obtención del software funcional en poco tiempo, con una realimentación continua entre el equipo de desarrollo y el cliente.

1.4.1 Metodologías ágiles

SCRUM: surgió para el desarrollo de productos tecnológicos, pero también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; situaciones frecuentes en el desarrollo de determinados sistemas de software (Schwaber, 2013). Al ser una metodología de desarrollo ágil:

- Es un modo de desarrollo de carácter adaptable más que predictivo.
- Está orientado a las personas más que a los procesos.
- Emplea la estructura de desarrollo ágil: incremental basada en iteraciones y revisiones.

Las iteraciones en SCRUM son la base del desarrollo ágil y condicionan su evolución a través de reuniones breves diarias en las que todo el equipo revisa el trabajo realizado el día anterior y el previsto para el día siguiente.

Programación extrema (Extreme Programming, XP en adelante): centrada en potenciar las relaciones interpersonales como clave para el éxito en desarrollo de software, promoviendo el trabajo en equipo, preocupándose por el aprendizaje de los desarrolladores, y propiciando un buen clima de trabajo. XP se basa en realimentación continua entre el cliente y el equipo de desarrollo, comunicación fluida entre todos los participantes, simplicidad en las soluciones implementadas y coraje para enfrentar los cambios (Letelier Torres, y otros, 2003).

Proceso XP:

Un proyecto XP tiene éxito cuando el cliente selecciona el valor de negocio a implementar basado en la habilidad del equipo para medir la funcionalidad que puede entregar a través del tiempo. El ciclo de desarrollo consiste (a grandes rasgos) en los siguientes pasos (Jeffries, Anderson y Hendrickson, 2001):

1. El cliente define el valor de negocio a implementar.
2. El programador estima el esfuerzo necesario para su implementación.
3. El cliente selecciona qué construir, de acuerdo con sus prioridades y las restricciones de tiempo.
4. El programador construye ese valor de negocio.
5. Vuelve al paso 1.

En todas las iteraciones de este ciclo tanto el cliente como el programador aprenden. No se debe presionar al programador a realizar más trabajo que el estimado, ya que se perderá calidad en el

software o no se cumplirán los plazos. De la misma forma el cliente tiene la obligación de manejar el ámbito de entrega del producto, para asegurarse que el sistema tenga el mayor valor de negocio posible con cada iteración (Jeffries, Anderson y Hendrickson, 2001).

El ciclo de vida ideal de XP consiste de seis fases (Beck, 2000):

- Fase I: Exploración.
- Fase II: Planificación de la Entrega.
- Fase III: Iteraciones.
- Fase IV: Producción.
- Fase V: Mantenimiento.
- Fase VI: Muerte del Proyecto.

Se seleccionó la metodología XP debido a que se ajusta en gran medida a las exigencias y escenarios de la investigación, pues el equipo de trabajo está formado por un desarrollador y se estará en constante intercambio de información con el cliente, admitiendo en la solución cambios de última hora, siendo el período de trabajo aproximadamente de 6 meses.

1.5 Lenguajes de programación

Los lenguajes de programación son aquellos que a partir de una gramática o conjunto de reglas, crean instrucciones para ser procesadas por una computadora. Estos pueden ser usados para la creación de aplicaciones capaces de controlar el comportamiento lógico y físico de las computadoras. En la actualidad existe una gran variedad de lenguajes, entre las diversas opciones disponibles se analizaron los siguientes como posibles opciones de desarrollo.

Java: lenguaje de programación orientado a objetos. El lenguaje en sí mismo toma mucha de su sintaxis de Lenguaje de Programación C y C++, pero tiene un modelo de objetos más simple. Las aplicaciones Java están típicamente compiladas en un bytecode[4], en el tiempo de ejecución, el bytecode es normalmente interpretado o compilado a código nativo para la ejecución. Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU GPL, de forma

4 Es un código intermedio más abstracto que el código máquina. Habitualmente es tratado como un archivo binario que contiene un programa ejecutable similar a un módulo objeto, que es un archivo binario producido por el compilador cuyo contenido es el código objeto o código máquina.

que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java aún no lo es). Otra característica es su independencia de la plataforma, significa que programas escritos en el lenguaje Java pueden ejecutarse igualmente en cualquier tipo de hardware y sistema operativo (Arnold, y otros, 1997).

C/C++: El lenguaje de programación C sirve para crear aplicaciones y software de sistemas. Posee un conjunto completo de instrucciones de control, con los cuales se pueden definir todas las tareas dentro de un desarrollo web. Con el lenguaje C se puede trabajar un programa en módulos lo que permite que se puedan compilar de modo independiente. El lenguaje C trabaja con librerías de funciones en las que básicamente sólo se necesitan cambiar los valores dentro de una aplicación dada. Lo importante también es la seguridad que ofrece C, ya que no entrega sólo los mecanismos básicos para tratar los datos que manipula con el hardware (Ritchie, Kernighan, 1996). Esto hace que sólo el programador pueda desarrollar el sistema. Lo anterior hace que el código generado por el lenguaje sea muy eficiente ya que los datos son tratados directamente por el hardware de números, caracteres y direcciones de los computadores.

Por otra parte C++ es un lenguaje que se comenzó a desarrollar en 1980 creado por B. Stroustrup. Al comienzo era una extensión del lenguaje C que fue denominada C with classes. (Garrido, 2006) El nombre C++ hace referencia al carácter del operador incremento de C (++). Ante la gran difusión y éxito que iba obteniendo en el mundo de los programadores, la AT&T comenzó a estandarizarlo internamente en 1987. En la actualidad, el C++ es un lenguaje versátil, potente y de propósito general. Su éxito entre los programadores profesionales le ha llevado a ocupar el primer puesto como herramienta de desarrollo de aplicaciones. El C++ mantiene las ventajas del C en cuanto a riqueza de operadores y expresiones, flexibilidad, concisión y eficiencia. Además, ha eliminado algunas de las dificultades y limitaciones del C original. Hay que indicar que el C++ mantiene compatibilidad casi completa con C, de forma que el viejo estilo de hacer las cosas en C es también permitido en C++, aunque este disponga de una mejor forma de realizar esas tareas.

1.6 Hilo de Ejecución

Es la unidad de procesamiento más pequeña que puede ser planificada por un sistema operativo. Se conoce además por hebra o subproceso. Básicamente es una tarea que puede ser ejecutada en paralelo con otra tarea. Hilo de ejecución o thread en inglés, en sistemas operativos, es una característica que permite a una aplicación realizar varias tareas concurrentemente. Los distintos hilos de ejecución comparten una serie de recursos tales como el espacio de memoria, los archivos abiertos, situación de autenticación, etc. Esta técnica permite simplificar el diseño de una aplicación que debe llevar a cabo distintas funciones simultáneamente (TANENBAUM, 2009).

En la herramienta a desarrollar se hace necesario el uso de varios hilos de ejecución debido a que se necesitan ejecutar varias tareas a la vez, entre las cuáles se encuentran: levantar la interfaz de la aplicación, recibir los datos de las estructuras que serán representadas a través de un protocolo de comunicación (socket), y recibir los datos de una imagen que se envía desde el cliente mediante otro socket. Si todas estas tareas se ejecutan en un único hilo entonces la aplicación pudiera bloquearse momentáneamente y la visualización pudiera atrasarse algunos segundos.

El aporte significativo que le brinda la programación multihilo a la solución es que aumenta respectivamente la calidad del renderizado de las estructuras a visualizar y la rapidez con que se ejecuta la aplicación.

1.7 Librerías 3D para el desarrollo

Debido a que la herramienta de visualización dinámica tiene que ser multiplataforma y debido a que las estructuras tienen que ser representadas en el espacio, solo se analizan las principales librerías 3D de java. Para la representación y el modelado en tercera dimensión, Java posee excelentes librerías gráficas que de cierta forma facilitan el trabajo y la interacción con los objetos 3D, dentro de las principales librerías se encuentran: JOGL, JGL y Java3D. Las cuáles serán objeto de estudio para seleccionar la más idónea en el desarrollo del visualizador.

JOGL

Es una de las Tecnologías de código abierto iniciada por: “the Game Technology Group” (Edinburgh Napier, 2015) y “Sun Microsystems” (Oracle y Sun Microsystems, 2015) en el año 2003 (JOGL–JOAL-Jinput) con el objetivo de proporcionar aceleración 3D por hardware a aplicaciones Java. Encapsula dentro de unas pocas clases Java toda la potencia de la biblioteca OpenGL. Su utilización no implica usar únicamente ficheros .jar sino que también es necesario hacer uso de bibliotecas de enlace dinámico propias de cada plataforma (Terrance, 2004).

Uno de los problemas que plantea el uso de esta API es que no es realmente orientada a objetos, pues la biblioteca OpenGL no ha sido remodelada y adaptada a Java, sino sólo es envuelta por una capa de Interfaz Nativa de Java (JNI, por sus siglas en inglés), lo que obliga al programador experto en Java a retroceder en el tiempo y volver a la época de la programación modular o estructurada.

JGL

JGL es una biblioteca de gráficos 3D desarrollada para Java con una API similar a la de OpenGL que puede ser ejecutada en la plataforma Java 2 y versiones anteriores. Esta, en su versión actual, no soporta todas las funciones de OpenGL. Cuenta con un cargador de ficheros OBJ que se distribuye bajo licencia MIT/X. Una de las principales desventajas es que no soporta Programación

Orientada a Objetos (POO), lo cual obliga al programador a utilizar programación estructurada. JGL es software libre bajo los términos de la licencia GNU (Ruggeri, 1998).

A diferencia de Java 3D u otras bibliotecas, JGL no tiene que ser pre-instalado, todos los códigos necesarios se descargan en tiempo de ejecución. JGL está escrito en Java puro, por lo que es realmente independiente de la plataforma y podría ser ejecutado en cualquier máquina compatible con Java.

Java 3D

El API 3D de Java es un árbol de clases Java que sirven como interface para sistemas de renderizado de gráficos tridimensionales y un sistema de sonido. El programador trabaja con constructores de alto nivel para crear y manipular objetos geométricos en 3D. Estos objetos geométricos residen en un universo virtual, que luego es renderizado. El API está diseñado con flexibilidad para crear universos virtuales precisos de una amplia variedad de tamaños, desde astronómicos a subatómicos

Aprovechándose de los Hilos Java, el renderizador Java 3D es capaz de renderizar en paralelo. El renderizador también puede optimizarse automáticamente para mejorar el rendimiento del renderizado. Un programa Java 3D crea ejemplares de objetos Java 3D y los sitúa en una estructura de datos de escenario gráfico. Este escenario gráfico es una composición de objetos 3D en una estructura de árbol que especifica completamente el contenido de un universo virtual, y cómo va a ser renderizado (Palos, 2010).

Selección de la librería

La librería de Java3D presenta numerosas ventajas sobre las librerías antes mencionadas, por ejemplo: la librería JOGL necesita además de los ficheros .jar las librerías de enlaces dinámicos dependiendo de la plataforma donde se ejecute la aplicación, no siendo esto necesario si se utiliza Java3D, ya que no usa librerías de enlace dinámico. Tanto JOGL como JavaGL no están orientadas a objetos lo cual atrasa considerablemente el desarrollo de la aplicación, en cambio Java3D permite a los desarrolladores generar aplicaciones de forma rápida empleando las facilidades que brinda la POO. Además del soporte a los cargadores que permite que se adapte a un gran número de formatos de ficheros. Realizando un análisis de las características de las librerías gráficas para el modelado 3D, se demuestra que la más idónea para el propósito del visualizador sería Java3D.

1.8 Herramientas para el desarrollo

Visual Paradigm 8.0

Herramienta para desarrollo de aplicaciones utilizando el Lenguaje Unificado de Modelado (UML, por sus siglas en inglés), ideal para quienes están interesados en construcción de sistemas a gran escala y necesitan confiabilidad y estabilidad en el desarrollo orientado a objetos. Constituye una herramienta privada pero en la universidad de las ciencias informáticas, centro donde estudian los autores de la investigación, se tiene una licencia que es usada con fines educativos. Dentro de sus características se tiene (Visual Paradigm):

- Modelo y código permanecen sincronizados en todo el ciclo de desarrollo.
- Interoperabilidad con modelos UML.
- Generación de bases de datos - Transformación de diagramas de Entidad-Relación en tablas de base de datos.
- Ingeniería inversa de bases de datos - Desde Sistemas Gestores de Bases de Datos (DBMS) existentes a diagramas de Entidad-Relación.

Debido a la experiencia en el uso de esta herramienta CASE en los proyectos productivos se opta por su utilización.

NetBeans IDE 8.0

Es un entorno de desarrollo, una herramienta para que los programadores puedan escribir, compilar, depurar y ejecutar programas. Está escrito en Java, pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extender el NetBeans IDE. NetBeans IDE es un producto libre y gratuito sin restricciones de uso (NetBeans).

Presenta una serie de características que facilitan en gran parte el desarrollo:

- Buen editor de código, multilenguaje, con el habitual coloreado y sugerencias de código, control de versiones, localización de ubicación de la clase actual, comprobaciones sintácticas y semánticas, plantillas de código, herramientas de refactorización, etcétera.
- Simplifica la gestión de grandes proyectos con el uso de diferentes vistas, asistentes de ayuda, y estructurando la visualización de manera ordenada, lo que ayuda en el trabajo diario.
- Herramientas para depurado de errores: el debugger que incluye el IDE es bastante útil para encontrar dónde fallan las cosas. Se pueden definir puntos de ruptura en la línea de código que sea de interés y monitorizar en tiempo real los valores de propiedades y variables.

- Optimización de código: El profiler ayuda a optimizar aplicaciones e intentar hacer que se ejecuten más rápido y con el mínimo uso de memoria.

1.9 Conclusiones parciales

A partir de la investigación realizada se evidencia la necesidad de implementar un software de visualización dinámica, pues los sistemas existentes no se integran a los software de simulación del proyecto CCSD para producir visualizaciones de estructuras espaciales y parámetros de comportamiento. Se seleccionó a XP como metodología de desarrollo de software, Java como lenguaje de programación (utilizando el API Java 3D), Visual Paradigm 8.0 para el modelado del sistema, y el NetBeans 8.0 como herramienta para la implementación de la solución.

CAPÍTULO 2: PROPUESTA DE SOLUCIÓN

2.1 Introducción

El objetivo fundamental de este capítulo es realizar una descripción de las funcionalidades del sistema, en otros términos, los aspectos funcionales y no funcionales que darán paso a la creación de las Historias de Usuarios a implementar. Se tratan las fases de exploración y planificación propias de la metodología de desarrollo utilizada para la implementación de la herramienta de visualización dinámica. Se realiza la estimación de las iteraciones en las que se dividirá el proceso y se exponen las tarjetas de Clases, Responsabilidades y Colaboración (CRC) utilizadas durante el diseño de la herramienta.

2.2 Usuarios del sistema

Los usuarios del sistema son todas aquellas personas o sistemas que interactúan con el mismo con la finalidad de obtener un resultado específico (Alegsa, 2010). La herramienta puede ser usada por cualquier usuario que lo desee, la misma no presenta ningún tipo de restricciones para los que acceden a sus funcionalidades. En la tabla 1 se muestran los usuarios del sistema.

Usuarios del sistema	Justificación
Especialista en simulación	Debe ser una persona con conocimientos en el área de la difusión en microfluidos con componentes biológicos.

Tabla 1. Usuarios del sistema.

2.3 Lista de reservas del producto

Según define la metodología XP la **lista de reservas del producto** está compuesta por los requisitos funcionales con los que debe contar la aplicación para satisfacer con eficiencia las necesidades requeridas por el cliente. A continuación se muestra el listado de los mismos.

2.3.1 Requisitos funcionales (RF).

Definen los servicios que el sistema debe proporcionar, cómo debe reaccionar a una entrada particular y como se debe comportar ante situaciones particulares.

RF1. Visualizar partículas.

RF1.1.Mostrar partículas.

RF1.2.Mostrar el movimiento de las partículas.

RF1.3 Mostrar campo vectorial de las partículas.

RF2. Visualizar superficies.

RF2.1. Mostrar superficies

RF2.2. Mostrar los triángulos que dan origen a las superficies.

RF2.3. Mostrar los puntos que dan origen a las superficies.

RF3. Visualizar polímeros.

RF4. Recibir datos del simulador.

RF5. Establecer área límite del escenario gráfico.

RF6. Cambiar ángulo de la cámara.

RF6.1. Mostrar el plano x-y del escenario.

RF6.2. Mostrar el plano x-z del escenario.

RF6.2. Mostrar el plano y-z del escenario.

RF7. Reiniciar la vista.

RF8. Habilitar navegación.

RF8.1. Activar la navegación por teclado.

RF8.2. Desactivar la navegación por teclado.

RF8.3. Activar el zoom.

RF8.4. Desactivar el zoom.

RF8.5. Activar rotación.

RF8.6. Desactivar rotación.

RF8.7. Activar traslación.

RF8.8. Desactivar traslación.

RF9. Mostrar el diagrama de comportamiento de la simulación.

2.3.2 Requisitos no funcionales.

Son restricciones que afectan a los servicios o funciones del sistema, tales como restricciones de tiempo, sobre el proceso de desarrollo, estándares, etc. Se agrupan en aspectos tales como:

Software

- El sistema debe ser multiplataforma.
- La visualización de una estructura a partir de los datos enviados por el simulador debe producirse en un período de tiempo que no exceda los 100 ms.

Apariencia o interfaz externa

- La interfaz del visualizador debe estar compuesta por dos ventanas, la primera debe contener en el centro el área de visualización y en la parte superior el menú principal de

la aplicación; y la segunda debe mostrar la imagen del diagrama de comportamiento de la simulación que es objeto de estudio en ese instante de tiempo.

- La ventana que muestra la imagen debe estar a la izquierda de la ventana principal y el usuario debe tener la facilidad de cambiar el tamaño de la misma a su gusto.

Usabilidad

- El menú principal debe estar formado por las funcionalidades que se brindan en el sistema.
- El sistema debe contener elementos representativos (íconos) que identifiquen cada una de las funcionalidades.

Hardware

Los requerimientos mínimos de hardware con los que debe cumplir un ordenador para un correcto funcionamiento del visualizador son:

- Procesador: Dual Core 2.0 GHz
- Memoria RAM: 1 GB
- Tarjeta gráfica: que soporte JAVA 3D

Estos son determinados a partir del funcionamiento de la solución una vez terminada, durante la visualización de una simulación típica, como podremos ver en la sección 3.4.4.

2.4 Patrones de diseño

En la tecnología de objetos, un **patrón** es una descripción de un problema y la solución, a la que se da un nombre, y que se puede aplicar a nuevos contextos; idealmente, proporciona consejos sobre el modo de aplicarlo en varias circunstancias, y considera los puntos fuertes y compromisos. Muchos patrones proporcionan guías sobre el modo en el que deberían asignarse las responsabilidades a los objetos, dada una categoría específica del problema. (Larman, 2003).

Entre los patrones de diseño más conocidos están los Patrones Generales de Asignación de Responsabilidades de Software, más conocidos por sus siglas en inglés como GRASP (Larman, 2003). A continuación se describen los patrones que fueron usados en la solución:

Patrón Experto

Es asignar una responsabilidad al experto en información. El experto en información es la clase que posee la información necesaria para cumplir con dicha responsabilidad (Larman, 2003). Este patrón es usado en todas las clases del diseño ya que cada una posee la información necesaria para realizar la tarea que le corresponde. Un ejemplo se evidencia en la clase *Partícula* que es la encargada de crear una estructura de tipo partícula. Para la creación de este tipo de estructura es

necesario conocer el radio, el color, las coordenadas x,y,z del centro y las componentes de las velocidades en los tres ejes de coordenadas; informaciones que posee la clase *Partícula*.

Patrón Creador

El patrón Creador guía la asignación de responsabilidades relacionadas con la creación de objetos, una tarea muy común. La intención básica del patrón Creador es encontrar un creador que necesite conectarse al objeto creado en alguna situación. Eligiéndolo como el creador se favorece el bajo acoplamiento (Larman, 2003). Un ejemplo se evidencia en la método *Reiniciar* de la clase *Visualizador* donde se crean las instancias de las clases *Partícula*, *Polímero* y *Superficie*.

Patrón Bajo Acoplamiento

El acoplamiento es una medida de la fuerza con que un elemento está conectado a, tiene conocimiento de, confía en, otros elementos. Un elemento con bajo (o débil) acoplamiento no depende de demasiados otros elementos. Estos elementos pueden ser clases, subsistemas, sistemas, etcétera (Larman, 2003). Con el objetivo de lograr un diseño que sea flexible, con facilidad para el soporte y que posea elementos reutilizables se utiliza este patrón.

Patrón Alta Cohesión

La cohesión es la medida de la fuerza con la que se relacionan las responsabilidades de un elemento. Un elemento con responsabilidades altamente relacionadas, y que no hace una gran cantidad de trabajo, tiene alta cohesión (Larman, 2003).

En el diseño de las clases se tuvo en cuenta este patrón para lograr un software más robusto. Por ejemplo las responsabilidades de la clase Visualizador están estrechamente relacionadas entre sí, pues el objetivo de todas es visualizar las estructuras bases.

Los patrones de la Banda de los Cuatro (GoF por sus siglas en inglés) son otros que son muy utilizados. En la solución es utilizado uno de ellos llamado Patrón Decorador.

Patrón Decorador

El patrón decorador atribuye responsabilidades adicionales a un objeto de forma dinámica. Los decoradores proveen una alternativa flexible de subclases para extender la funcionalidad (Gamma, 1995). Este patrón es usado en la solución debido a que la visualización de los procesos de difusión en microfluidos con componentes biológicos complejos se realiza de forma dinámica, razón por la cual las responsabilidades del objeto estarán sujetas a cambios.

2.5 Exploración

En esta fase, los clientes plantean a grandes rasgos las historias de usuario que son de interés para la primera entrega del producto. Al mismo tiempo el equipo de desarrollo se familiariza con las

herramientas, tecnologías y prácticas que se utilizarán en el proyecto. Se prueba la tecnología y se exploran las posibilidades de la arquitectura del sistema construyendo un prototipo. La fase de exploración toma de pocas semanas a pocos meses, dependiendo del tamaño y familiaridad que tengan los programadores con la tecnología (Sánchez, 2004).

2.5.1 Historias de Usuario

Las historias de usuario (HU) tienen la misma finalidad que los casos de uso pero con algunas diferencias: Constan de 3 o 4 líneas escritas por el cliente en un lenguaje no técnico sin hacer mucho hincapié en los detalles; no se debe hablar ni de posibles algoritmos para su implementación ni de diseños de base de datos adecuados, etc. Son usadas para estimar tiempos de desarrollo de la parte de la aplicación que describen. También se utilizan en la fase de pruebas, para verificar si el programa cumple con lo que especifica la historia de usuario. Cuando llega la hora de implementar una historia de usuario, el cliente y los desarrolladores se reúnen para concretar y detallar lo que tiene que hacer dicha historia. El tiempo de desarrollo ideal para una historia de usuario es entre 1 y 3 semanas (Calabria y Píriz, 2003).

Las Historias de Usuarios tienen tres aspectos:

- **Tarjeta:** En ella se almacena suficiente información para identificar y detallar la historia.
- **Conversación:** Cliente y programadores discuten la historia para ampliar los detalles (verbalmente cuando sea posible, pero documentada cuando se requiera confirmación).
- **Pruebas de Aceptación:** Permite confirmar que la historia ha sido implementada correctamente (Eclipse).

A continuación se hace explicación y descripción de la plantilla de HU que se ha de utilizar para el desarrollo del sistema y en las tablas 3 y 4 se muestran dos ejemplos:

Historia de Usuario	
No.: Se ponen los números sucesivos a partir de 1.	**Nombre:** Se identifica la HU.
Usuario: Aquí se colocan los usuarios que interactúan con la HU.	

Historia de Usuario	
Prioridad en el Negocio: Define el impacto de la HU para el negocio, de acuerdo con las necesidades del usuario. Y se define como (Alta / Media / Baja)	**Nivel de Complejidad:** Define la dificultad técnica al desarrollar la HU, desde el punto de vista del programador. Y se define como (Alto / Medio / Bajo)
Estimación: Permiten estimar duración de implementación.	**Iteración Asignada:** Precisa la iteración a la que pertenece la historia de usuario.
Descripción: Explica en qué consiste las HU.	
Información adicional (Observaciones): Información que se estime agregar para hacer más entendible la HU.	

Tabla 2. Descripción de la HU.

Historia de Usuario	
No.: 1	**Nombre:** Visualizar partículas.
Usuario: Simulador y especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 1
Descripción: El sistema debe mostrar en pantalla un sistema de partículas, debe permitirle al usuario escoger entre dos vistas (partículas o campo vectorial) y debe mostrar el movimiento de las partículas si el usuario selecciona esa opción.	
Información adicional (Observaciones): Da cumplimiento al RF1 ”Visualizar partículas”.	

Tabla 3. HU Visualizar partículas.

Historia de Usuario	
No.: 2	**Nombre:** Visualizar superficies.
Usuario: Simulador y especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 1
Descripción: El sistema debe permitirle al usuario observar las superficies de tres formas distintas (volumétricamente, a través de los triángulos que la forman y a través de los puntos que la forman).	
Información adicional (Observaciones): Da cumplimiento al RF2 ”Visualizar superficies”.	

Tabla 4. HU Visualizar superficies.

Las restantes historias de usuario se encuentran en el **Anexo1**.

2.6 Planificación

En esta fase el cliente establece la prioridad de cada historia de usuario, y correspondientemente, los programadores realizan una estimación del esfuerzo necesario de cada una de ellas. Se toman acuerdos sobre el contenido de la primera entrega y se determina un cronograma en conjunto con el cliente. Una entrega debería obtenerse en no más de tres meses. Esta fase dura unos pocos días. Otro aspecto a tener en cuenta durante esta fase es la velocidad del proyecto, la cual se estima utilizando el tiempo empleado en el desarrollo de las iteraciones terminadas. Esta medida debe revaluarse una vez concluidas 3 o 4 iteraciones y en caso de no cumplir el tiempo estimado, debe ser negociando con el cliente un nuevo Plan de Entregas (Sánchez, 2004).

2.6.1 Iteraciones

Esta fase incluye varias iteraciones sobre el sistema antes de ser entregado. En la primera iteración se puede intentar establecer una arquitectura del sistema que pueda ser utilizada durante el resto del proyecto. Esto se logra escogiendo las historias que fuercen la creación de esta arquitectura, sin embargo, esto no siempre es posible ya que es el cliente quien decide qué historias se implementarán en cada iteración (para maximizar el valor de negocio). Al final de la última iteración el sistema estará listo para entrar en producción. Los elementos que deben tomarse en cuenta durante la elaboración del Plan de la Iteración son: historias de usuario no abordadas, velocidad del proyecto, pruebas de aceptación no superadas en la iteración anterior y tareas no terminadas en la iteración anterior. Todo el trabajo de la iteración es expresado en tareas de programación, cada una de ellas es asignada a un programador como responsable, pero

llevadas a cabo por parejas de programadores (Sánchez, 2004). En la tabla 5 se muestra el plan de iteraciones.

Iteraciones	Orden de las Historias de Usuario a implementar	Cantidad de tiempo de trabajo
Iteración 1	HU1. Visualizar partículas. HU2. Visualizar superficies. HU3. Visualizar polímeros.	6 semanas
Iteración 2	HU4. Recibir datos del simulador. HU5. Establecer área límite del escenario gráfico. HU6. Cambiar ángulo de la cámara.	5 semanas
Iteración 3	HU7. Reiniciar la vista. HU8. Habilitar navegación. HU9. Mostrar el diagrama de comportamiento de la simulación.	4 semana

Tabla 5. Iteraciones.

2.6.1 Plan de Entregas

El cronograma de entregas establece qué historias de usuario serán agrupadas para conformar una entrega, y el orden de las mismas. Este cronograma será el resultado de una reunión entre todos los actores del proyecto (cliente, desarrolladores, gerentes, etc.). XP denomina a esta reunión “Juego de planeamiento”, pero puede denominarse de la manera que sea más apropiada al tipo de empresa y cliente (Joskowicz, 2008).

Típicamente el cliente ordenará y agrupará según sus prioridades las historias de usuario. El cronograma de entregas se realiza en base a las estimaciones de tiempos de desarrollo realizadas por los desarrolladores. Luego de algunas iteraciones es recomendable realizar nuevamente una reunión con los actores del proyecto, para evaluar nuevamente el plan de entregas y ajustarlo si es necesario (Joskowicz, 2008). En la tabla 6 se muestra el Plan de Entrega.

Entregable	Final 1ra iteración 2 da semana de Marzo de 2015	Final 2da iteración 3 ra Semana de Abril de 2015	Final 3ra iteración 3 ra semana de Mayo de 2015
Visualizador	versión 0.1	versión 0.2	Versión 1.0

Tabla 6. Plan de Entrega.

2.7 Tarjetas CRC

Las tarjetas CRC (Contenido, Responsabilidad y Colaboración) permiten desprenderse del método de trabajo basado en procedimientos y trabajar con una metodología basada en objetos. Además, permiten que el equipo completo ayude en la tarea del diseño. (Carrillo, 2005)

Las características más sobresalientes de las tarjetas CRC son su simpleza y ductilidad. Una tarjeta CRC es una ficha de papel o cartón que representa a una entidad del sistema. (Casas, 2008)

Debido a la facilidad de su uso y entendimiento, se decidió utilizarlas para el diseño de la herramienta que se desea desarrollar. En la tabla 7 se muestra la plantilla de las tarjetas CRC.

Tarjetas CRC	
Clase: Nombre de la clase que se está modelando.	
Responsabilidades: Es una descripción de alto nivel del propósito de la clase.	**Colaboraciones:** Indica con que otras clases se relaciona.

Tabla 7. Plantilla para Tarjetas CRC.

Tarjetas CRC	
Clase: Visualizador.	
Responsabilidades: Actualizar	**Colaboraciones:** Color

Tarjetas CRC	
addLights createBehaviors createMenuBar dibujarParticulas dibujarVector guardarArchivo pintarCampoVectorial pintarLineasPolimero pintarSuperficies	DatoSocket HiloServer HiloSocket HiloSocket1 Imagenes Particula Polimero Superficie

Tabla 8. Tarjeta CRC de la clase Visualizador.

En la tabla 8 se muestra la tarjeta CRC de la clase Visualizador, las restantes se encuentran en el **Anexo2**.

2.8 Arquitectura

El concepto de arquitectura de software se refiere a la estructuración del sistema que, idealmente, se crea en etapas tempranas del desarrollo. Esta estructuración representa un diseño de alto nivel del sistema que tiene dos propósitos primarios: satisfacer los atributos de calidad (desempeño, seguridad, modificabilidad), y servir como guía en el desarrollo. Al igual que en la ingeniería civil, las decisiones críticas relativas al diseño general de un sistema de software complejo deben de hacerse desde un principio. El no crear este diseño desde etapas tempranas del desarrollo puede limitar severamente el que el producto final satisfaga las necesidades de los clientes. Además, el costo de las correcciones relacionadas con problemas en la arquitectura es muy elevado. Es así que la arquitectura de software juega un papel fundamental dentro del desarrollo (Cervantes, 2010).

Cuando se define la arquitectura para una aplicación se pretende orquestar una propuesta que satisfaga los requisitos, por ello la definición arquitectónica debe centrarse en garantizar que la propuesta desarrollada satisface los requisitos no funcionales y permite alcanzar los requisitos funcionales. En este trabajo se define una arquitectura que tiene como línea base arquitectónica la propuesta realizada en JavaSE (Java Standard Edition) 6. En la ilustración 4 se muestra dicha arquitectura. A continuación se explicarán los elementos fundamentales de la arquitectura de Java SE que fueron usados en la solución:

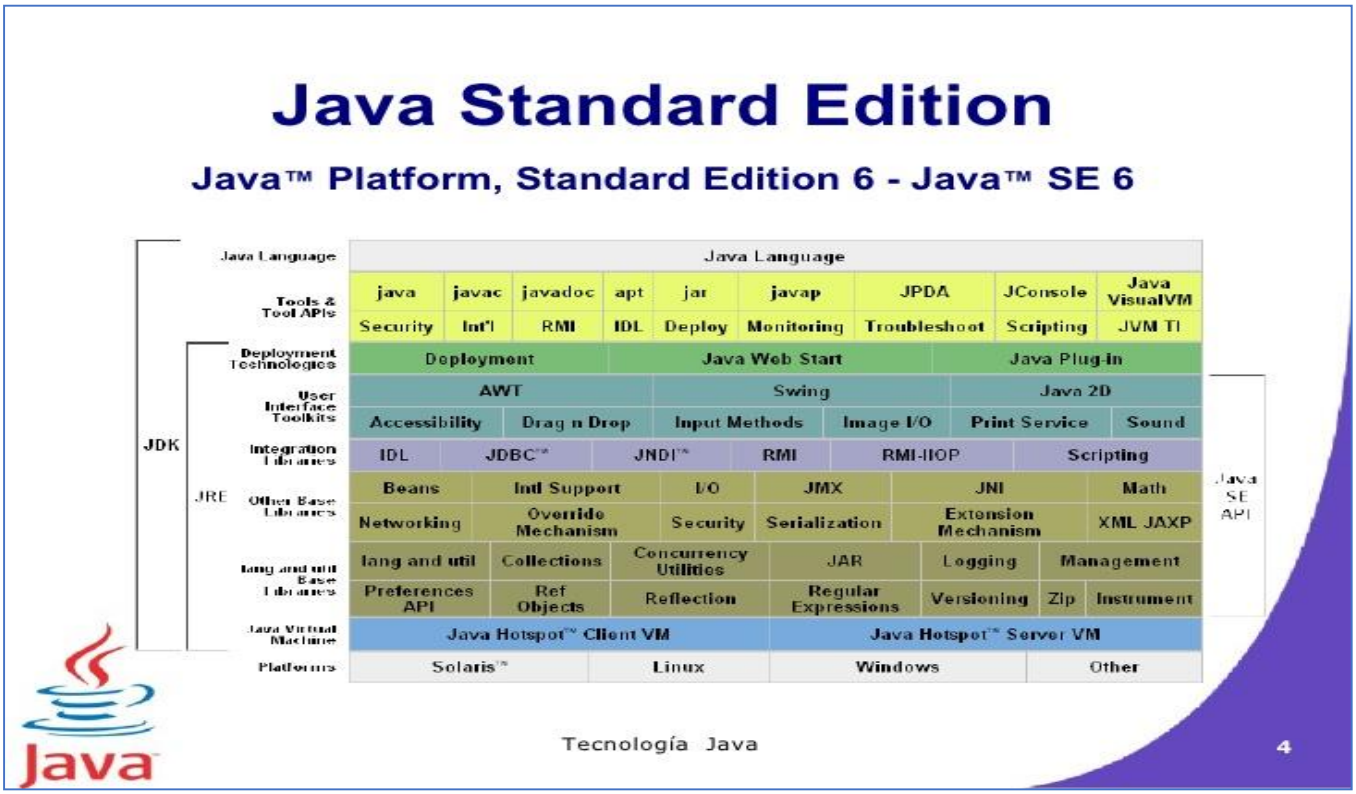


Ilustración 4. Arquitectura de Java Standard Edition 6 (Quesada, 2009).

JRE y JDK:

Oracle ofrece dos productos de software principales de la Plataforma Java ™, Standard Edition (Java ™ SE) de la familia:

Java SE Runtime Environment (JRE):

El JRE proporciona las bibliotecas, la máquina virtual Java, y otros componentes necesarios para ejecutar applets y aplicaciones escritas en el lenguaje de programación Java. Este entorno de tiempo de ejecución puede ser redistribuido con aplicaciones para que sean independientes.

Java SE Development Kit (JDK):

El JDK incluye los JRE además de línea de comandos y herramientas de desarrollo tales como compiladores y depuradores que son necesarias o útiles para el desarrollo de applets y aplicaciones.

Lenguaje de programación Java (Java Programming Language):

El lenguaje de programación Java es de propósito general, concurrente, orientado a objetos y basado en clases. Normalmente se compila para el conjunto de instrucciones de código de bytes y el formato binario definido en la máquina virtual de Java.

Máquina Virtual de Java (Java Virtual Machine):

La máquina virtual de Java es una máquina de computación abstracta que tiene un conjunto de instrucciones y manipula la memoria en tiempo de ejecución. Está portada a diferentes plataformas para proporcionar al sistema de la independencia entre hardware y sistema operativo.

La Plataforma Java, Standard Edition proporciona dos implementaciones de la máquina virtual Java (VM):

Java HotSpot VM cliente:

El cliente VM es una aplicación para plataformas usados típicamente para aplicaciones cliente. El cliente VM está sintonizado para reducir el tiempo de puesta en marcha y la huella de memoria. Se puede invocar utilizando la opción -client de línea de comandos al iniciar una aplicación.

Java HotSpot VM Servidor:

El servidor de VM es una aplicación diseñada para una máxima velocidad de ejecución del programa. Se puede invocar utilizando la opción -server de línea de comandos al iniciar una aplicación.

Bibliotecas Base (Base Libraries):

Las clases e interfaces que proporcionan funciones básicas y la funcionalidad fundamental para la plataforma Java.

Lenguaje y paquetes de utilidad (Lang and Util Packages)

Paquete java.lang (Package java.lang):

Proporciona clases que son fundamentales para el diseño del lenguaje de programación Java. Las clases más importantes son Object, que es la raíz de la jerarquía de clases y Class, las instancias de las cuales representan las clases en tiempo de ejecución.

Paquete java.util (Package java.util):

Contiene las colecciones del marco de trabajo, clases de colección heredadas, modelo de eventos, servicios de fecha y hora, la internacionalización, y clases de utilidad diversas (una cadena Tokenizer, un generador de números aleatorios, y un arreglo de bits).

Matemáticas (Math):

La clase Math contiene métodos para realizar operaciones numéricas básicas, como la primaria exponencial, logaritmo, raíz cuadrada, y funciones trigonométricas. A diferencia de algunos de los métodos numéricos de clase StrictMath, todas las implementaciones de las funciones equivalentes de la clase Math no se definen de devolver los mismos resultados bit a bit. Esta excepción permite implementaciones de mejor rendimiento que no requieren una estricta reproducibilidad.

Versión del paquete de identificación (Package Version Identification):

La característica del paquete de versiones permite el control de versiones a nivel de paquete para que las aplicaciones y applets pueden identificar en tiempo de ejecución la versión de una JRE, VM específica, y el paquete de clase.

Reflexión (Reflection):

Reflexión permite código Java para descubrir información sobre los campos, métodos y constructores de clases cargadas y utilizar campos reflejados, métodos y constructores para funcionar con sus contrapartes en los objetos subyacentes, dentro de las restricciones de seguridad. La API acomoda aplicaciones que necesitan acceso a cualquiera de los miembros públicos de un objeto de destino (en función de su clase en tiempo de ejecución) o los miembros declarados por una clase dada. Los programas pueden suprimir el control de acceso reflexivo defecto.

Marco de Colecciones (Collections Framework):

Una colección es un objeto que representa un grupo de objetos. El marco de las colecciones es una arquitectura unificada para la representación de las colecciones, lo que les permite manipular de forma independiente de los detalles de su representación. Reduce el esfuerzo de programación al tiempo que aumenta el rendimiento. Permite la interoperabilidad entre las API no relacionadas, reduce el esfuerzo en el diseño y el aprendizaje de nuevas API, y fomenta la reutilización del software.

Archivos Java (Java Archive (JAR) Files):

JAR (Java Archive) es un formato de archivo independiente de la plataforma que agrega muchos archivos en uno solo. Los applets de Java y sus múltiples componentes necesarios (.class archivos, imágenes y sonidos) pueden ser agrupados en un archivo JAR y posteriormente descargar en un navegador en una sola transacción HTTP, lo que mejora la velocidad de descarga. El formato JAR también soporta la compresión, lo que reduce el tamaño del archivo, mejorando aún más el tiempo de descarga. Además, el autor applet puede firmar digitalmente las entradas individuales en un archivo JAR para autentificar su origen. Es totalmente extensible.

Preferencias (**Preferences):**

La API de Preferencias proporciona una forma para que las aplicaciones almacenen y recuperen datos de usuario y de preferencias del sistema y de configuración. Los datos se almacenan de forma persistente en un almacén de respaldo dependiendo de la implementación. Hay dos árboles separados de nodos preferentes, uno para las preferencias del usuario y otro para las preferencias del sistema.

Otros Paquetes Base (Other Base Packages)

Serialización de objetos (Object Serialization):

La serialización de objetos extiende del núcleo de las clases Java de entrada / salida con soporte para objetos. La serialización de objetos apoya la codificación de los objetos, y los objetos alcanzables de ellos, en un flujo de bytes; y es compatible con la reconstrucción complementaria

del gráfico de objetos de la corriente. La serialización se usa para la persistencia ligera y para la comunicación a través de socket o invocación de método remoto (RMI).

Redes (Networking):

Proporciona clases para trabajar con funcionalidades de red, incluyendo direccionamiento, clases para el uso de URLs y URI, clases de socket para la conexión a los servidores, funcionalidad de seguridad en redes, y más.

Las librerías de interfaz de usuario (User Interface Libraries)

Imagen I/O (Image I/O):

La API de Java de imagen de E / S proporciona una arquitectura conectable para trabajar con imágenes almacenadas en archivos y se accede a través de la red. La API proporciona un marco para la adición de plugins específicas del formato. Plug-ins para varios formatos comunes se incluyen con Java imagen de E / S, pero terceras partes pueden utilizar esta API para crear sus propios plugins para manejar formatos especiales.

Java 2D™ Graficos e Imágenes (Graphics and Imaging):

El API Java 2D ™ es un conjunto de clases para gráficos avanzados en 2D y de imágenes. Abarca la línea de arte, el texto y las imágenes en un único modelo integral. La API proporciona un amplio soporte para la composición de la imagen y las imágenes de canal alfa, un conjunto de clases para proporcionar definición precisa del color espacio y conversión, y un rico conjunto de operadores de imagen de visualización orientada.

Kit de herramientas de ventanas abstracto (AWT):

Abstract Windowing Toolkit (AWT) de la plataforma Java™ proporciona APIs para la construcción de componentes de interfaz de usuario, tales como menús, botones, campos de texto, cuadros de diálogo, casillas de verificación, y para el manejo de la entrada del usuario a través de esos componentes. Además, AWT permite la prestación de formas simples tales como óvalos y polígonos y permite a los desarrolladores controlar el diseño de interfaz de usuario y las fuentes utilizadas por sus aplicaciones.

Componente gráfico de interfaz de usuario (Swing):

Las API de Swing también proporcionan componente gráfico (GUI) para el uso en interfaces de usuario. Las API de Swing están escritas en el lenguaje de programación Java sin ninguna dependencia de código que es específica de las facilidades GUI proporcionados por el sistema operativo subyacente. Esto permite que los componentes de Swing GUI se puedan cambiar mientras se ejecuta una aplicación.

Plataformas (Platforms):

Oracle proporciona implementaciones del JDK y JRE para Microsoft Windows, Linux y los sistemas operativos Solaris. Otras compañías pueden proporcionar implementaciones de la plataforma Java para otros sistemas operativos como Macintosh, AIX, etc.

2.9 Descripción de la solución

La herramienta de visualización propuesta permite la visualización dinámica de simulaciones de la difusión en microfluidos con componentes biológicos y se integra a los simuladores que existen en el proyecto CCSD. Su funcionamiento se basa en el uso de dos sockets, uno para recibir los datos de las estructuras, que se envían desde el simulador y el otro para recepcionar las imágenes que son enviadas desde la máquina cliente.

2.9.1 Sockets

Los sockets no son más que puntos o mecanismos de comunicación entre procesos que permiten que un proceso hable (emita o reciba información) con otro proceso incluso estando estos procesos en distintas máquinas. Esta característica de interconectividad entre máquinas hace que el concepto de socket nos sirva de gran utilidad. La comunicación entre procesos a través de sockets se basa en la filosofía CLIENTE-SERVIDOR: un proceso en esta comunicación actuará de proceso servidor creando un socket cuyo nombre conocerá el proceso cliente, el cual podrá "hablar" con el proceso servidor a través de la conexión con dicho socket nombrado (Walton, 2001).

El mecanismo de comunicación vía sockets tiene los siguientes pasos:

1. El proceso servidor crea un socket con nombre y espera la conexión.
2. El proceso cliente crea un socket sin nombre.
3. El proceso cliente realiza una petición de conexión al socket servidor.
4. El cliente realiza la conexión a través de su socket mientras el proceso servidor mantiene el socket servidor original con nombre.

Es muy común en este tipo de comunicación lanzar un proceso hijo, una vez realizada la conexión, que se ocupe del intercambio de información con el proceso cliente mientras el proceso padre servidor sigue aceptando conexiones.

2.9.2 Funcionamiento multihilo de la aplicación

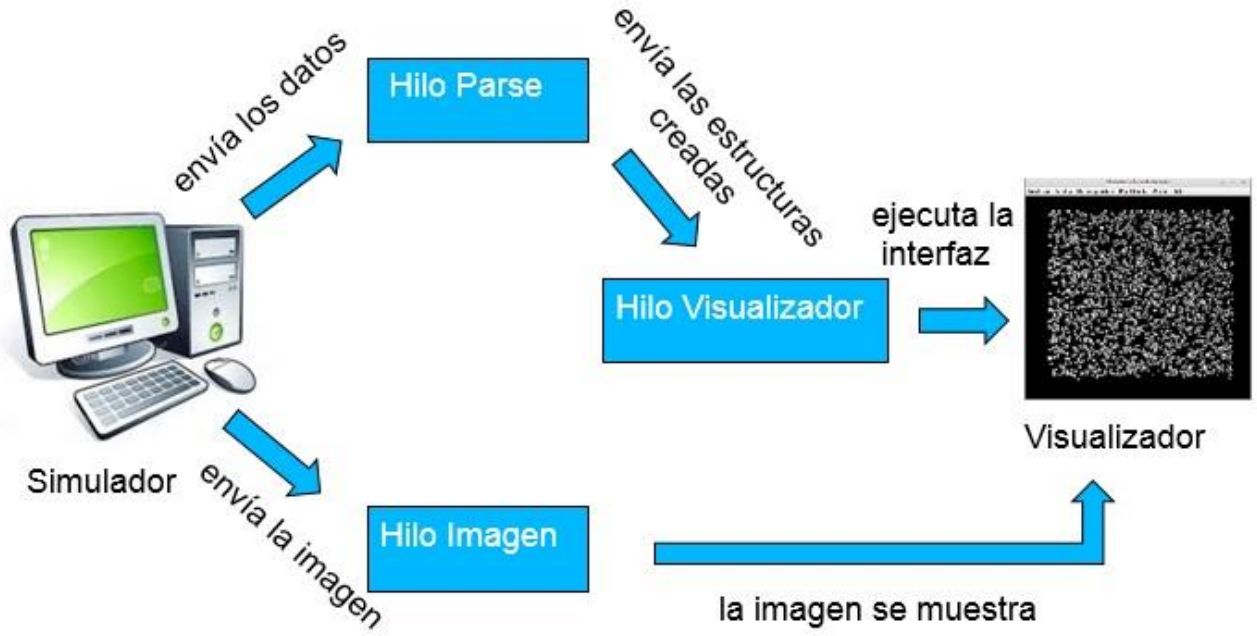


Ilustración 5. Funcionamiento multihilo de la aplicación.

Para garantizar la comunicación entre el simulador y el visualizador se utiliza una programación multihilo, de esta manera se optimiza el renderizado de las estructuras bases a visualizar, permitiendo un mejor análisis de los procesos de difusión en microfluidos con componentes biológicos complejos. En la ilustración 5 se muestra el funcionamiento multihilo de la aplicación.

A continuación se explica la relación existente entre el simulador, el visualizador y los hilos de ejecución:

El simulador envía los datos de las estructuras a visualizar a través de un socket, estos datos son recibidos en otro socket que se ejecuta en el Hilo Parse, aquí se crean las estructuras que representan un estado específico de la simulación en curso. Luego se envían al Hilo Visualizador que es el encargado de ejecutar la interfaz principal del visualizador y representar en pantalla todas las estructuras bases que han sido recibidas. Concurrentemente con este proceso se efectúa otro envío de datos por parte del simulador, en este caso es una imagen que representa el diagrama de comportamiento de la simulación. Esta foto se envía por otro socket y es recibida en el Hilo Imagen por un socket que tiene como responsabilidad recibir cada actualización que se produzca en el diagrama de un estado a otro. Finalmente la imagen se representa en otra ventana al lado de la aplicación facilitando un mejor entendimiento de los fenómenos que se simulan.

2.9.3 Estructura de la herramienta

La herramienta está formada por una ventana principal que tiene en su parte superior un menú con las siguientes opciones: Archivo, Vista, Navegación, Partícula y Superficie (ver ilustración 6).

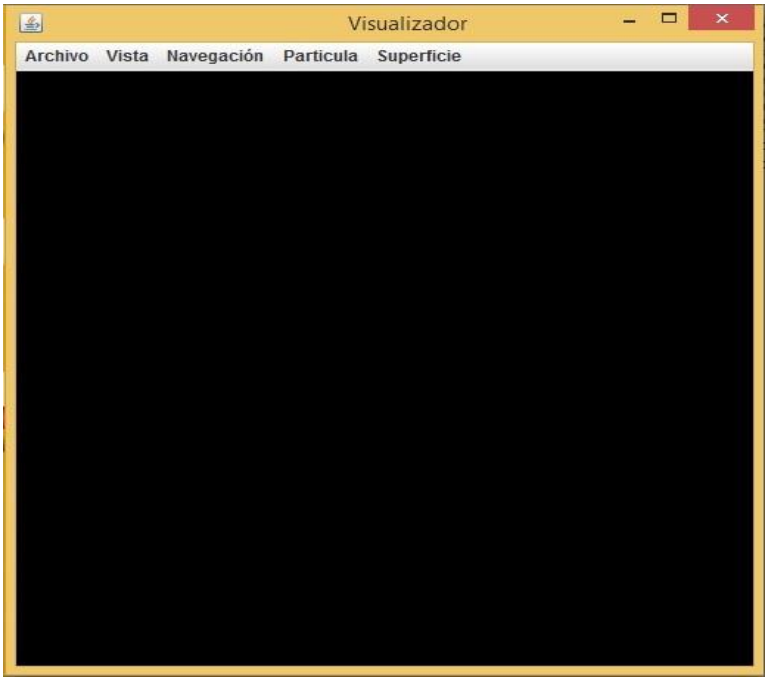


Ilustración 6. Ventana principal.

2.9.3.1 Menú Archivo

En el menú archivo aparecen las opciones de: Reiniciar, Guardar imagen, Mostrar varias estructuras y Salir (ver ilustración 7).

Reiniciar:

Esta opción permite deshacer todos los cambios que pudo realizar el usuario usando las opciones del menú Navegación, retornando la vista al estado inicial de la visualización en curso.

Guardar imagen:

Esta opción le brinda la posibilidad al usuario de guardar la imagen que se encuentra en pantalla en el directorio que él determine.

Mostrar varias estructuras:

Esta opción brinda la posibilidad de representar un conjunto de estructuras de diferentes tipos (superficies, mallas y polímeros) intercalados entre sí. Ya que la opción por defecto es representar una única estructura.

Salir:

Esta opción como su nombre lo indica permite el cierre de la aplicación.

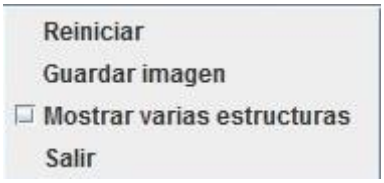


Ilustración 7. Menú archivo.

2.9.3.2 Menú Vista

En el menú vista aparecen las opciones de: Plano-xy, Plano-yz y Plano-xz (ver ilustración 8).

Plano-xy:

Esta opción le permite al usuario cambiar la posición de la cámara en el plano-xy, es decir podrá ver las imágenes por delante y por detrás.

Plano-yz:

Esta opción le permite al usuario cambiar la posición de la cámara en el plano-yz, es decir podrá ver las imágenes por la izquierda y por la derecha.

Plano-xz:

Esta opción le permite al usuario cambiar la posición de la cámara en el plano-xz, es decir podrá ver las imágenes por arriba y por abajo.

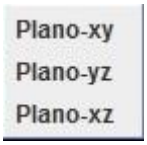


Ilustración 8. Menú vista.

2.9.3.3 Menú Navegación

En el menú Navegación aparecen las opciones de: Activar movimiento por cursores, Activar zoom, Activar rotación y Activar traslación (ver ilustración 9).

Activar movimiento por cursores:

Esta opción permite activar o desactivar el movimiento a través de los cursores.

Activar zoom:

Esta opción permite activar o desactivar el zoom, es decir la capacidad de acercar o alejar un objeto visual.

Activar rotación:

Esta opción permite activar o desactivar la posibilidad de que los objetos visuales roten alrededor de la pantalla.

Activar traslación:

Esta opción permite activar o desactivar la traslación por parte de los objetos visuales.

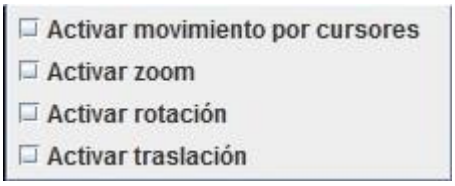


Ilustración 9. Menú navegación.

2.9.3.4 Menú Partícula

En el menú Partícula aparecen las opciones de: Ver movimiento, Partículas y Campo-Vectorial (ver ilustración 10).

Ver movimiento:

Esta opción permite ver el movimiento de todas las partículas que se encuentran en pantalla, inicialmente todas van a estar estáticas. Se muestra una animación que se repite de forma continua Hasta que se actualizan los datos suavizando la transición entre imágenes.

Partículas:

Esta opción permite mostrar una vista con las partículas representadas como esferas y también es posible mostrar su movimiento.

Campo-Vectorial:

Esta opción permite mostrar una vista de las partículas diferente, aquí se representa su campo vectorial, donde el tamaño de cada vector representa la magnitud de su velocidad, y la dirección y sentido permiten conocer para donde será su movimiento.

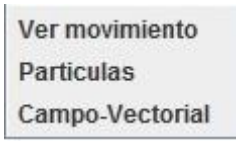


Ilustración 10. Menú Partícula.

2.9.3.5 Menú Superficie

En el menú Superficie aparecen las opciones de: Color y Modo de Vista (ver ilustración 11 y 12).

Color: Esta opción permite cambiar el color de las superficies que están representadas en pantalla.

Modo de Vista: Esta opción permite cambiar el modo de vista en que se muestran las superficies entre uno de los siguientes modos:

1. **Superficie:** permite que las superficies sean observadas de manera volumétrica.
2. **Triángulos:** permite observar los triángulos que forman a las superficies.
3. **Puntos:** permite observar los puntos que forman a las superficies.

Ilustración 11. Menú Superficie opción Color.

Ilustración 12. Menú Superficie opción Modo de Vista.

2.9.4 Interface simulador-visualizador

La comunicación entre el simulador y el visualizador se realiza mediante el envío de cadenas de texto a través de un socket, cada tipo de estructura tiene una nomenclatura específica, este será el lenguaje que el visualizador entenderá. A continuación se realiza una descripción de esta nomenclatura para cada tipo de estructura:

2.9.4.1 Partículas

1. "radio/color/posición x/posición y/posición z/velocidad x/velocidad y/velocidad z--0.1000/azul/0.4/-0.2145/0.1236/10/10/1”
2. El carácter “/” significa que un nuevo atributo viene a continuación.
3. Los caracteres “--” significan que una nueva partícula viene a continuación.

2.9.4.2 Superficies

"0.2/0.4/0>2/1=0.5/0.4/0>2/0=0.2/0.6/0>0/1"

"0/0/0>1/2/3=0/1/0>2/3=1/0/0>3=1/1/1>n"

1. El carácter “/” significa que un nuevo atributo viene a continuación.
2. Los tres primeros parámetros son las coordenadas x, y, z de un punto.
3. El carácter “>” significa que a continuación aparecerán los vecinos del punto(los números son las posiciones que ocupan en la lista de puntos).
4. El carácter “=” significa que un nuevo punto viene a continuación.

5. En el caso de que un punto no tenga vecinos se le pone una “n”.

2.9.4.3 Polímeros

“radio/color/posición x/posición y/posición z>1”

"0.1000/rojo/0.8/0.354/0.5236>1<0.1000/blanco/0.2145/0.5145/0.5236>0/2<0.1000/amarillo/0.5145/-0.5145/0.5236>n"

1. Los 5 primeros parámetros coinciden con los 5 primeros de las partículas.
2. El carácter “>” significa que a continuación aparecerán las partículas con las que se enlaza la partícula que se está analizando(los números son las posiciones que ocupan en la lista de partículas).
3. El caracter “<” significa que viene una nueva partícula.
4. En el caso de que una partícula no se enlace con ninguna se le pone una “n”.

La herramienta permite visualizar estructuras de una manera individual o mixta, es decir, se pueden representar solo sistemas de partículas, grupos de polímeros o solamente un grupo de superficies. Pero también es posible representar cualquier combinación entre ellas. La representación de las estructuras mixtas se realiza a través de tres listas que guardan de manera individual a cada ejemplar que pueda ser representado según la clase a la que este pertenezca.

El visualizador cada cierto período de tiempo refresca (elimina) todas las imágenes que se encuentra en pantalla para comenzar con un nuevo estado de la simulación en curso y esto se realiza a través del envío por el socket de la cadena “refrescar”.

2.9.5 Visualización de partículas, superficies y polímeros.

A continuación se muestran imágenes de estructuras representadas con la herramienta:

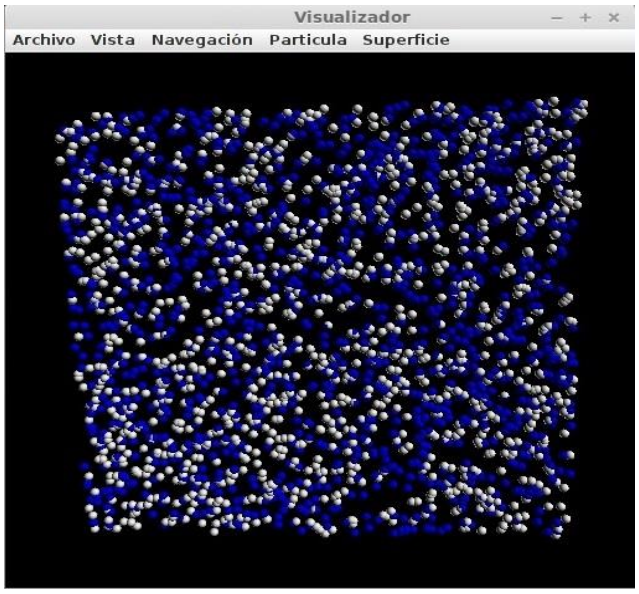


Ilustración 13. Visualización de partículas.

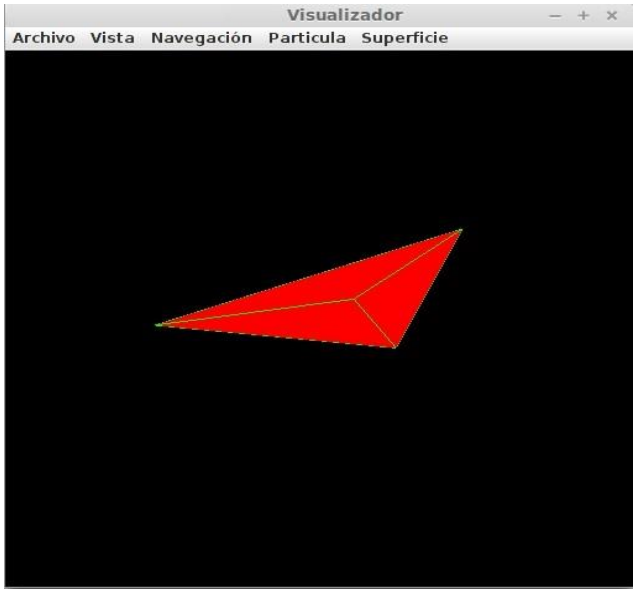


Ilustración 14. Visualización de una superficie.

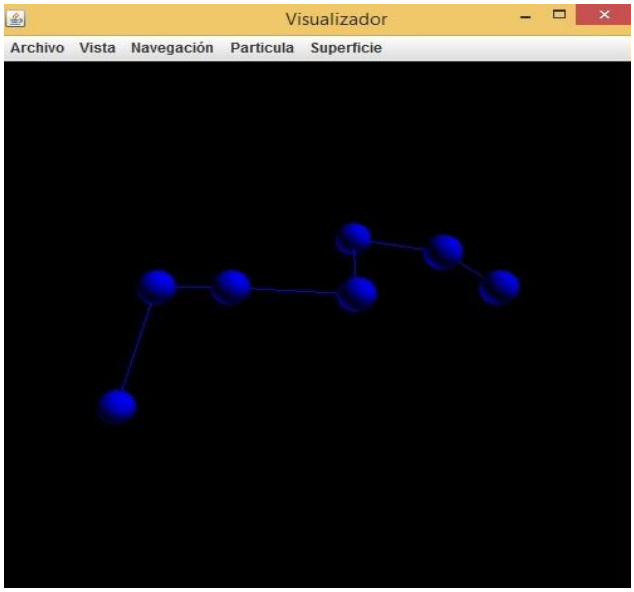


Ilustración 15. Visualización de un polímero.

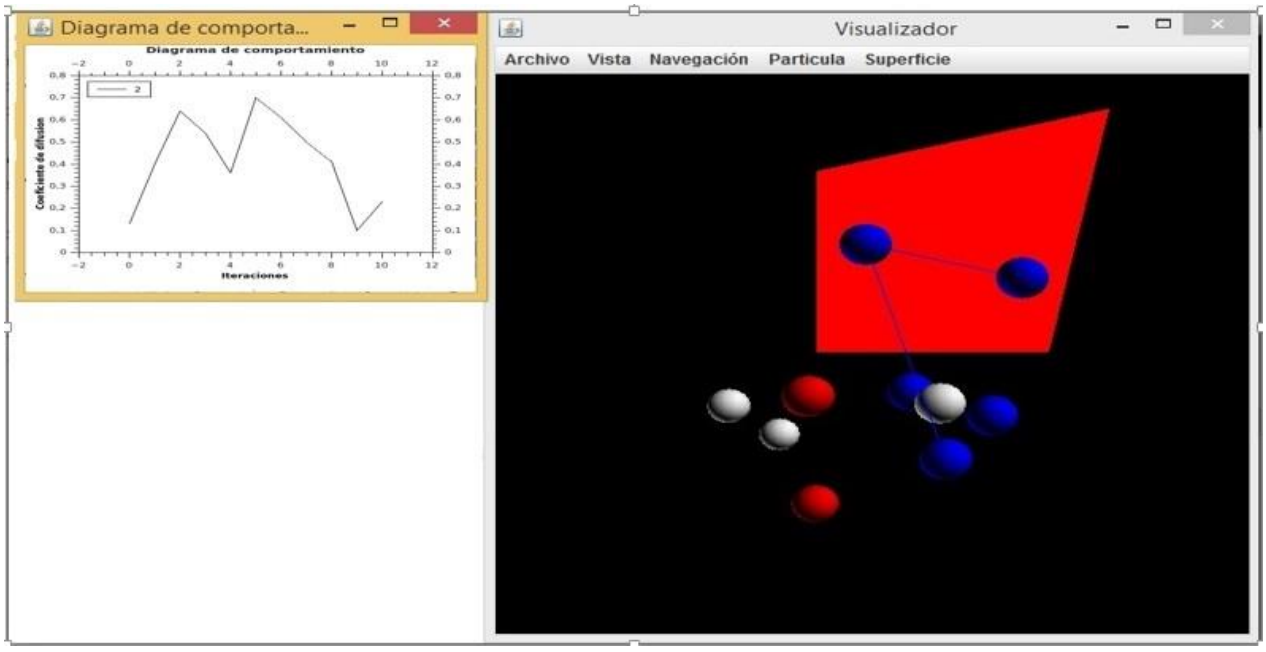


Ilustración 16. Visualización de varias estructuras incluyendo el diagrama de comportamiento.

2.91 Conclusiones parciales

En este capítulo se describió la propuesta de solución para la creación de una herramienta de visualización dinámica de simulaciones de la difusión en microfluidos con componentes biológicos. Donde el carácter dinámico se garantiza a través de la utilización de programación multihilo para disminuir los tiempos de respuesta de la aplicación y mejorar el renderizado de las estructuras a visualizar. Se realizó una descripción de las HU que permitieron representar las funcionalidades del sistema, se estableció las prioridades de las mismas para el negocio, en correspondencia a una estimación del esfuerzo de desarrollo, además se especificaron cuáles serían implementadas en cada iteración, se determinaron las fechas de entrega de cada una y consecuentemente la fecha de entrega final de la herramienta. Se realizó una definición de la arquitectura del sistema y por último se mostraron representaciones de estructuras producidas por el visualizador.

CAPÍTULO 3: VALIDACIÓN DE LA SOLUCIÓN

3.1 Introducción

En este capítulo se realiza la validación de la solución propuesta. Se evalúa tanto a nivel de diseño como de implementación para comprobar el correcto funcionamiento del sistema. Se muestran los resultados de la aplicación de las técnicas y métricas para validar los requisitos, y también los resultados arrojados después de haber realizado las pruebas de caja negra y caja blanca.

3.2 Validación de los requisitos

Para verificar que la solución propuesta cumple con todos los requisitos definidos previamente con el cliente, se efectuó un proceso de validación de estos requisitos, donde se emplearon las técnicas siguientes:

- **Generación de casos de prueba de aceptación:** para validar los requisitos funcionales de la solución, se diseñaron casos de pruebas de aceptación para cada una de las Historias de Usuario.
- **Construcción de prototipos:** mediante los prototipos se le mostró al cliente un modelo ejecutable de la solución que le permitió tener una visión preliminar de cómo sería el sistema.

La validación de requisitos es importante, pues de ella depende que no existan elevados costos de mantenimiento para el software desarrollado.

Para medir la calidad de la especificación de requisitos se aplicó una de las métricas definidas por la metodología XP, especificación de requisitos (ER).

Para determinar la ER o ausencia de ambigüedad en los requisitos se realiza la siguiente operación matemática:

ER= Nui/Nr

Dónde:

Nr: es el total de requisitos de especificación.

Nui: es el número de requisitos para los cuales todos los revisores tuvieron interpretaciones idénticas.

Mientras más cerca de 1 esté el valor de ER, menor será la ambigüedad.

Resultados de la validación de requisitos:

ER=Nui/Nr

ER=9/9

ER=1

Luego de la aplicación de la métrica para medir la calidad de la especificación de los requisitos se obtuvo un resultado satisfactorio con lo que se puede concluir que los requisitos no poseen ambigüedad.

3.3 Validación del diseño

El IEEE “Standard Glosary of Software Engering Terms” define como métrica: “una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado” (Ejiogo, 1991).

Pressman plantea en (Pressman, 2005) que los ingenieros de software usan las métricas del producto como apoyo para construir software de mayor calidad, aunque no suelen ser absolutas, proporcionan una manera sistemática de evaluar la calidad a partir de un conjunto de reglas definidas con claridad.

Para comprobar y evaluar la calidad del diseño del sistema se hará uso de las métricas TOC y RC propuestas por Lorenz y Kidd (Lorenz, y otros, 1994).

La métrica TOC permite medir la responsabilidad, la complejidad de implementación y la reutilización de las clases del diseño (Chidamber y Kemerer, 1994). Es importante destacar que para esta métrica, la responsabilidad y la complejidad son inversamente proporcionales a la reutilización, por lo que a mayor responsabilidad y complejidad de implementación de una clase, menor será su nivel de reutilización. Los resultados de la aplicación de estas métricas se muestran en la Ilustración.

Atributo de calidad	Categoría	Criterio
Responsabilidad	Baja	< =Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Atributo de calidad	Categoría	Criterio
Complejidad de implementación	Baja	< =Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio
Reutilización	Baja	> 2*Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<= Promedio

Tabla 9. Rango de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica TOC.

A continuación se muestran las clases del sistema evaluadas en los atributos de calidad mencionados.

Clase	Cantidad de procedimientos	Responsabilidad	Complejidad de implementación	Reutilización
Color	2	Baja	Baja	Alta
DatoSocket	3	Baja	Baja	Alta
HiloServer	1	Baja	Baja	Alta
HiloSocket	1	Baja	Baja	Alta
HiloSocket1	1	Baja	Baja	Alta
Imagenes	2	Baja	Baja	Alta

Clase	Cantidad de procedimientos	Responsabilidad	Complejidad de implementación	Reutilización
Particula	8	Alta	Alta	Baja
Polimero	4	Media	Media	Media
Superficie	5	Media	Media	Media
Visualizador	12	Alta	Alta	Baja

Tabla 10. Evaluación de las clases del sistema mediante la métrica TOC.

A continuación se muestran las gráficas con los resultados obtenidos:

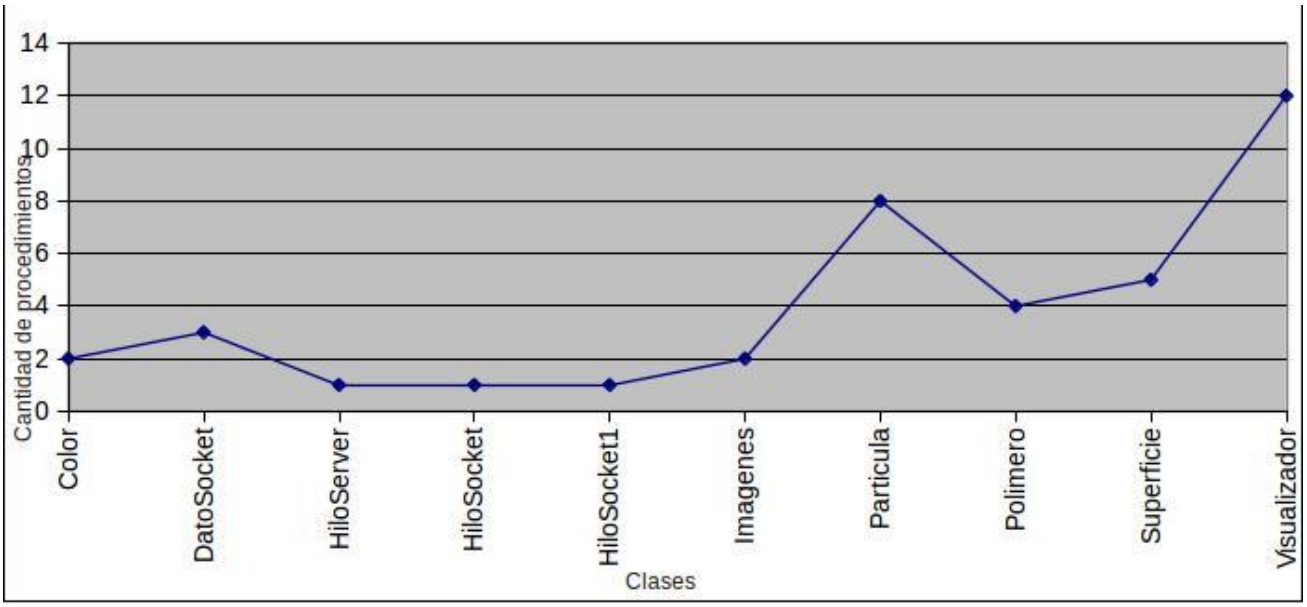
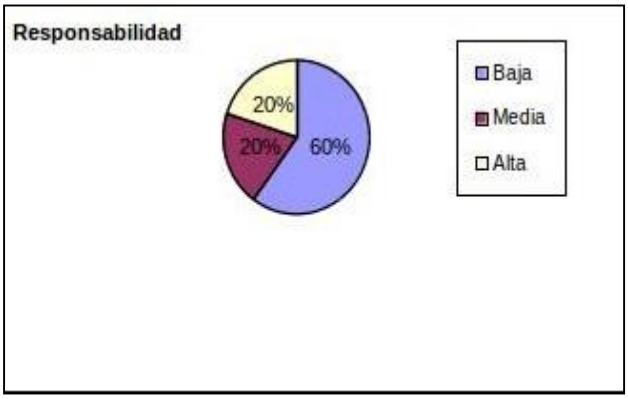
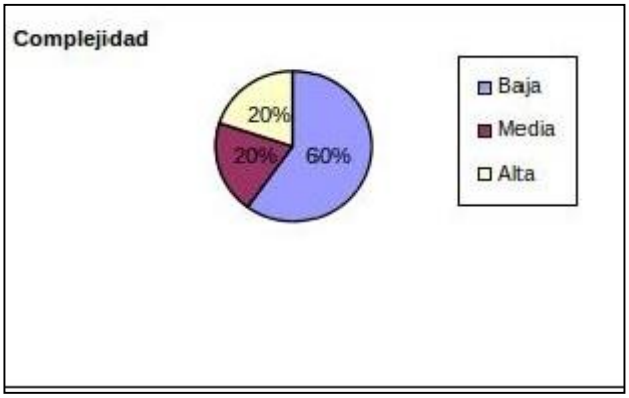


Ilustración 17. Relación entre las clases y la cantidad de procedimientos.





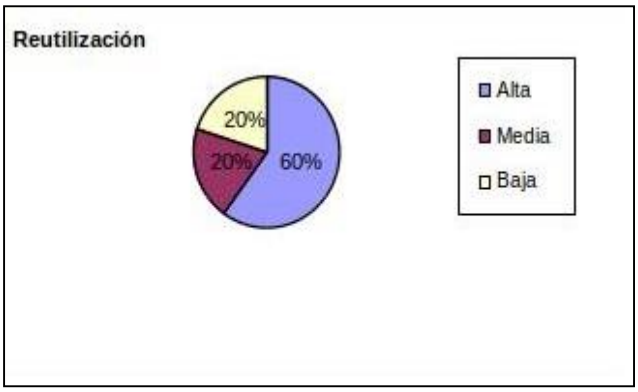


Ilustración 18. Responsabilidad, complejidad y reutilización de las clases del diseño.

Después de aplicar la métrica TOC se puede observar que las clases del sistema poseen una baja responsabilidad, baja complejidad de implementación y alta reutilización, por lo que el diseño de clases respecto a la cantidad de funcionalidades de cada una es aceptable.

La métrica RC permite evaluar el acoplamiento, la complejidad de mantenimiento, la reutilización y la cantidad de pruebas de unidad necesarias para probar una clase teniendo en cuenta las relaciones existentes entre ellas (Chidamber y Kemerer, 1994).

Atributo de calidad	Categoría	Criterio
Acoplamiento	Ninguno	0
	Baja	1
	Media	2
	Alta	>2
Complejidad de mantenimiento	Baja	< =Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Atributo de calidad	Categoría	Criterio
Reutilización	Baja	> 2*Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	<= Promedio
Cantidad de pruebas	Baja	< =Promedio
	Media	Entre Promedio y 2* Promedio
	Alta	> 2* Promedio

Tabla 11. Rangos de valores para la evaluación técnica de los atributos de calidad relacionados con la métrica RC.

A continuación se muestran las clases del sistema evaluadas en los atributos de calidad mencionados.

Clase	Cantidad de relaciones de uso	Acoplamiento	Complejidad de mantenimiento	Reutilización	Cantidad de pruebas
Color	3	Alto	Media	Media	Media
DatoSocket	2	Medio	Baja	Alta	Baja
HiloServer	2	Medio	Baja	Alta	Baja
HiloSocket	1	Bajo	Baja	Alta	Baja
HiloSocket1	1	Bajo	Baja	Alta	Baja
Imagenes	2	Medio	Baja	Alta	Baja
Particula	2	Medio	Baja	Alta	Baja
Polimero	3	Alto	Media	Media	Media

Clase	Cantidad de relaciones de uso	Acoplamiento	Complejidad de mantenimiento	Reutilización	Cantidad de pruebas
Superficie	2	Medio	Baja	Alta	Baja
Visualizador	9	Alto	Alta	Baja	Alta

Tabla 12. Evaluación de las clases del sistema mediante la métrica RC.

A continuación se muestran las gráficas con los resultados obtenidos:

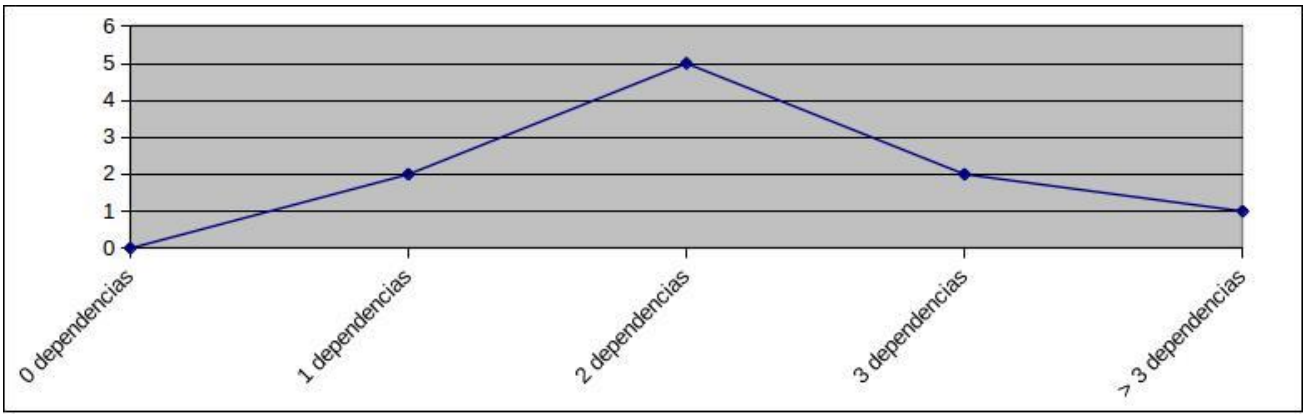
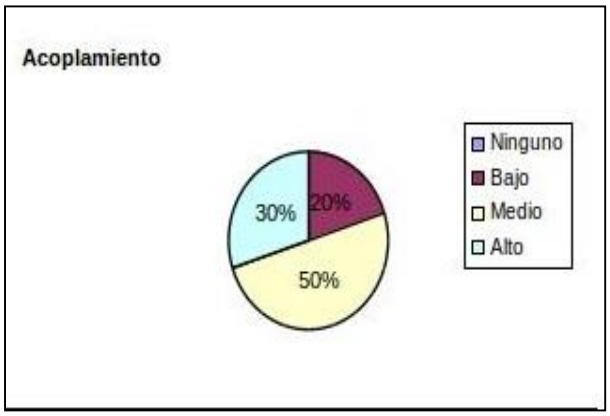


Ilustración 19. Dependencias entre clases.



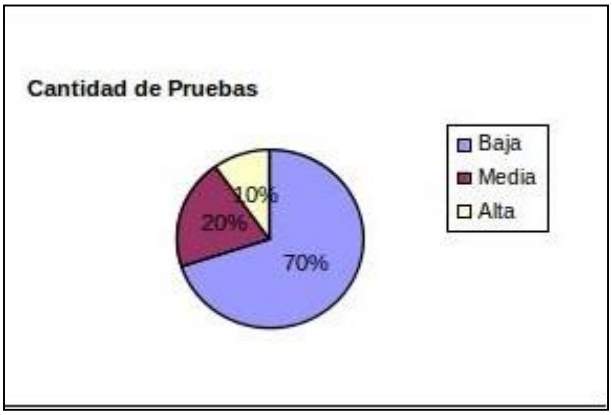


Ilustración 20. Complejidad de mantenimiento, acoplamiento, cantidad de pruebas y reutilización de las clases según la métrica RC.

Los resultados obtenidos en los atributos de calidad: complejidad de mantenimiento, reutilización, cantidad de pruebas y acoplamiento son satisfactorios de acuerdo a lo planteado en la métrica RC.

3.4 Pruebas de software

La IEEE define las pruebas de software como “una actividad en la que un sistema o un componente es ejecutado bajo condiciones especificadas, los resultados son observados o registrados, y una evaluación es realizada de un aspecto del sistema o componente”. Se realizan para para descubrir defectos en el software y demostrar el mismo que satisface los requerimientos establecidos con el cliente.

La metodología XP divide las pruebas en dos grupos: pruebas unitarias, desarrolladas por los programadores, encargadas de verificar el código de forma automática y las pruebas de aceptación, destinadas a evaluar si al final de una iteración se obtuvo la funcionalidad requerida, además de comprobar que dicha funcionalidad sea la esperada por el cliente (Beck, y otros, 2000). Se realizarán pruebas a nivel de unidad y sistema a través del método de caja blanca y caja negra, para revisar código de la aplicación y la correcta implementación de las funcionalidades del sistema respectivamente.

3.4.1 Prueba de caja blanca

La prueba de caja blanca es un método de diseño de casos de prueba que usa la estructura de control del diseño procedimental para obtener los casos de prueba. Mediante los métodos de prueba de caja blanca, el ingeniero de software puede obtener casos de prueba que garanticen que se ejercita por lo menos una vez todos los caminos independientes de cada módulo. (Pressman, 2005).

La prueba del camino básico es una técnica de prueba de caja blanca que permite al diseñador de casos de prueba obtener una medida de la complejidad lógica de un diseño procedimental y usar esa medida como guía para la definición de un conjunto básico de caminos de ejecución. Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa. (Pressman, 2005).

A continuación se muestra el método “existenTodosLosTiposDeEstructuras” en el cual se determina si se están visualizando todos los tipos de estructuras que la aplicación puede representar, es decir, determina si existe al menos una partícula, un polímero y una superficie.

```
    boolean todas=false;
    if(ExisteAlmenosUnaParticula(entrada)){
        if(ExisteAlmenosUnaSuperficie(entrada)){
            if(ExisteAlmenosUnPolimero(entrada)){
                todas=true;
            }
        }
    }
    return todas;
}
```

Ilustración 21. Método para determinar si se están visualizando todos los tipos de estructuras.

A continuación se muestra el grafo dirigido de flujo del método "existenTodosLosTiposDeEstructuras".

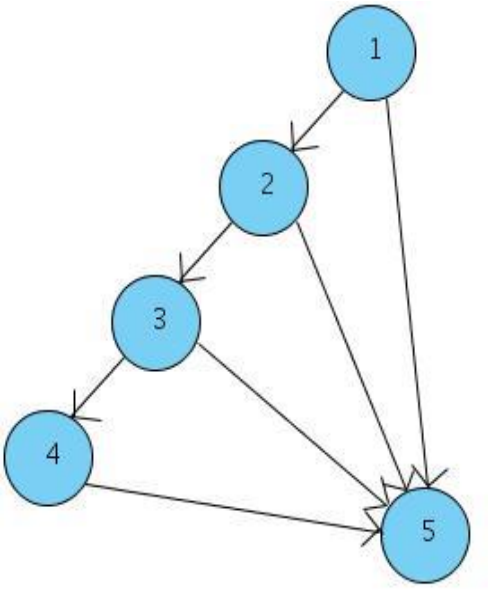


Ilustración ¡Error! Secuencia no especificada..

La complejidad ciclomática es una métrica de software que proporciona una medida cuantitativa de la complejidad lógica de un programa. Cuando se emplea en el contexto del método de prueba de la ruta básica, el valor calculado mediante la complejidad ciclomática define el número de rutas independientes en el conjunto básico de un programa, y proporciona un límite superior para el número de pruebas que deben aplicarse para asegurar que todas las instrucciones se hayan ejecutado por lo menos una vez. (Pressman, 2005). La complejidad ciclomática, V(G), se calcula de una de tres maneras:

1. El número de regiones corresponde a la complejidad ciclomática.
2. V(G) = E-N+2, donde E es el número de aristas y N el número de nodos.
3. V(G) = P+1, donde P es el número de nodos predicado (son aquellos de los que salen varios caminos).

Usando como guía la figura anterior, se calculó la complejidad ciclomática empleando los tres algoritmos indicados para demostrar que la respuesta es la misma.

1. Existen 4 regiones.
2. V(G) = 7 – 5 + 2 = 4
3. V(G) = 3 + 1 = 4

La complejidad ciclomática es igual a 4, lo cual tiene como significado que existen 4 posibles caminos linealmente independientes, representando el mínimo número de casos de prueba para el procedimiento tratado. A continuación se muestran los caminos existentes:

Número	Caminos
1	1-5
2	1-2-5
3	1-2-3-5
4	1-2-3-4-5

Tabla 13. Caminos por donde el flujo puede circular.

El siguiente paso es ejecutar los casos de pruebas para cada camino y se compara con los resultados esperados, teniendo en cuenta que las instrucciones se hayan ejecutado por lo menos una vez. A continuación se muestra el caso de prueba para el camino básico 1, en el Anexo 3 se muestran los restantes casos de prueba.

Caso de prueba para el camino básico 1	
Descripción	Se comprueba si en la cadena pasada por parámetro existe al menos una estructura de tipo partícula.
Condiciones de ejecución	Que se haya ejecutado al menos una simulación.
Entrada	entrada
Resultado esperado	Que la aplicación siga ejecutando el resto del código.

Tabla 14. Caso de prueba para el camino básico 1.

3.4.2 Prueba de caja negra

Las pruebas de caja negra se centran en los requisitos funcionales del software, permite al ingeniero de software obtener conjuntos de condiciones de entrada que ejerciten completamente todos los requisitos funcionales de un programa. (Pressman, 2005).

La prueba de caja negra intenta encontrar errores de las siguientes categorías (Pressman, 2005):

1. funciones incorrectas o ausentes.
2. errores de interfaz.
3. errores en estructuras de datos o en accesos a bases de datos externas.
4. errores de rendimiento.
5. errores de inicialización y terminación.

3.4.3 Casos de prueba

Un caso de prueba es una serie de pruebas de entrada, condiciones de ejecución y resultados esperados desarrollados para un objetivo en particular, tal como verificar el cumplimiento de un requisito en específico (RUP, 2008).

A continuación se muestran ejemplos de algunos casos de prueba para la aplicación.

Caso de Prueba de caja negra	
Código: 6	**HU:** 1
Nombre: Visualizar partículas.	
Descripción: El sistema debe mostrar en pantalla un sistema de partículas, debe permitirle al usuario escoger entre dos vistas (partículas o campo vectorial) y debe mostrar el movimiento de las partículas si el usuario selecciona esa opción. **Condiciones de ejecución:** Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Seleccionar alguna de las opciones del menú Particula. **Resultados esperados:** La herramienta debe mostrar en pantalla un sistema de partículas.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 15. Caso de prueba para la HU Visualizar partículas.

Caso de Prueba de caja negra	
Código: 6	**HU:** 6
Nombre: Cambiar ángulo de la cámara. **Descripción:** La herramienta debe mostrar una vista de las partículas en el plano-xy, plano-xz y plano-yz.	
Condiciones de ejecución: Se debe de ejecutar al menos una simulación ante de seleccionar esta opción.	

Caso de Prueba de caja negra
Entrada/Pasos de ejecución: Seleccionar alguna de las opciones del menú Vista.
Resultados esperados: El usuario puede observar las estructuras desde diferentes ángulos. **Evaluación de la prueba:** Prueba Satisfactoria.

Tabla 16. Caso de prueba para la HU Cambiar ángulo de la cámara.

El resto de los casos de pruebas se encuentran en el **Anexo 3**.

3.4.4 Resultados de las pruebas de caja negra

Se realizaron 2 iteraciones en las que se detectaron 6 no conformidades (NC, en adelante) que fueron resueltas de manera rápida. En la primera iteración se detectaron 6 NC dentro de las cuales 3 fueron de aplicación, 2 de validaciones y 1 de ortografía. En la segunda iteración no se detectaron NC (ver ilustración 22).

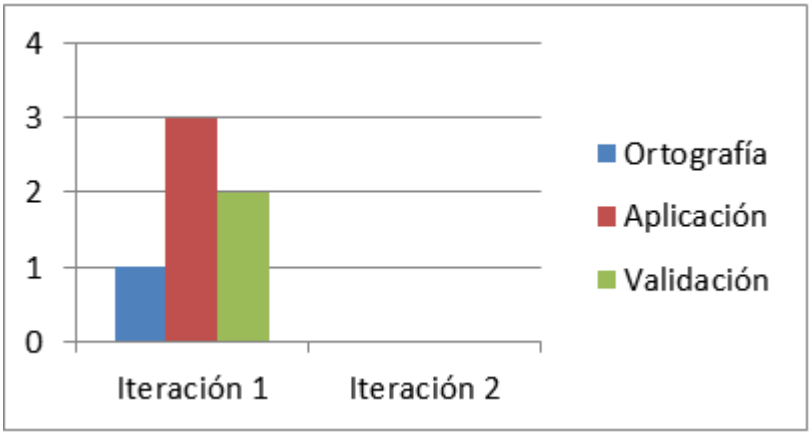


Ilustración 22. Gráfica obtenida como resultado de aplicar las pruebas de caja negra.

3.4.5 Pruebas de rendimiento realizadas a la aplicación

Se realizaron pruebas de rendimiento a la aplicación para el caso del envío de 1000 partículas y para el caso de 10 000 partículas. En el primer caso se obtuvo que el consumo de memoria RAM fue de 430 mb, el tiempo de espera fue de 50.8 ms y el tiempo de respuesta fue de 38.4 ms. En el segundo caso se obtuvo un consumo de memoria RAM de 703 mb, el tiempo de espera fue de 90.7 ms y el tiempo de respuesta fue de 65.1 ms. Para realizar estas pruebas se realizaron 1000

envíos en cada caso por separado y para obtener los resultados se realizó un promedio de los valores obtenidos. En general los resultados se clasifican como buenos, ya que los tiempos se encuentran alrededor de los 100 ms comparable con los 16 Hz para uso en PC.

3.4.6 Validación de la herramienta.

Para validar que el proceso de visualización dinámica cumpliera con todos los requisitos así como la existencia de una buena calidad visual se hizo uso de la técnica Juicio de Expertos.

El juicio de expertos se define como una opinión informada de personas con trayectoria en el tema, que son reconocidas por otros como expertos cualificados en éste, y que pueden dar información, evidencia, juicios y valoraciones (Escobar y Cuervo, 2008). La cantidad de expertos seleccionada para la evaluación fue de cinco personas, teniendo en cuenta que el proceso de difusión en microfluidos con componentes biológicos complejos es un proceso de larga duración.

Esta técnica se utilizó mediante la Agregación Individual, que consiste en obtener la información de manera individual de cada uno de ellos, sin la exigencia de que se pongan en contacto (Cabero y Llorente, 2013).

El procedimiento utilizado para la selección de los expertos fue el biograma. El mismo consiste en elaborar una biografía del experto en función de sus respuestas sobre aspectos de su trayectoria como, por ejemplo, años de experiencia y formación, investigaciones o acciones formativas, conocimiento del objeto de estudio, a partir de los cuales se infiere su adecuación y pertinencia para su actividad de experto (Cabero y Llorente, 2013).

En la etapa de selección de los expertos fueron seleccionados cinco expertos de la UCI, tres de ellos pertenecientes al Grupo de Matemática y Física Computacionales y dos pertenecientes al centro CEIGE de la facultad 3 (en los anexos 5, 6, 7, 8 y 9 se encuentran las síntesis biográficas que validan su experticia).

Los expertos son:

- Dr. Jorge Gulín González
- Lic. Edisel Navas Conyedo.
- MsC. Yailen Costa Marrero.
- MsC Handy Hernández Dalmau.
- MsC Julio Cesar Diaz Vera.

Mediante una entrevista individual se les presentó un modelo con una explicación breve sobre los objetivos del trabajo y los resultados que se deseaban obtener. Se les presentó los aspectos a

valorar a través de una tabla Aspectos / Rangos de Valoración y un vídeo con las imágenes de un proceso de difusión en un microfluidos con componentes biológicos complejos. Los aspectos a evaluar son la visualización de las estructuras bases: partículas, polímeros y superficies y la interacción entre ellas. Los rangos de valoración son: Muy Mala, Mala, Regular, Buena y Muy Buena. Ver tabla 17.

Experto	Visualización de partículas	Visualización de polímeros	Visualización de superficies	Interacción entre las estructuras
Dr. Jorge Gulín González	Muy Buena	Buena	Muy Buena	Muy Buena
Lic. Edisel Navas Conyedo	Buena	Muy Buena	Muy Buena	Buena
MsC. Yailen Costa Marrero	Muy Buena	Muy Buena	Buena	Muy Buena
MsC. Handy Hernández Dalmau	Buena	Muy Buena	Muy Buena	Buena
MsC. Julio Cesar Diaz Vera	Muy Buena	Muy Buena	Muy Buena	Muy Buena

Tabla 17. Valoración de los expertos.

Con las valoraciones obtenidas por parte de los expertos se puede llegar a la conclusión de que la visualización de las partículas, polímeros y superficies tiene una buena calidad y que la interacción entre las estructuras tiene una gran semejanza a la interacción que ocurre en los procesos de difusión en microfluidos reales.

3.5 Conclusiones parciales

Se describieron las técnicas de validación de requisitos y las pruebas efectuadas al diseño y las funcionalidades del sistema propuesto. La aplicación de técnicas de validación de requisitos permitió comprobar que los requisitos obtenidos están en correspondencia con las necesidades del cliente. Al aplicar las métricas TOC y RC se pudo comprobar el correcto diseño de las clases. Las pruebas de caja blanca realizadas a los métodos del sistema mediante la técnica “camino básico” arrojaron los resultados esperados en cada caso. Las pruebas de caja negra aplicadas a las HU permitieron detectar un conjunto de NC que fueron resueltas en una iteración, obteniéndose la solución que el cliente deseaba.

CONCLUSIONES GENERALES

El uso de tecnologías guiadas por la filosofía del software libre favoreció en la búsqueda de la documentación y las herramientas que guiaron la realización de este trabajo, además de contribuir a la soberanía informática establecida por la universidad y el país.

La investigación evidenció la necesidad de incorporar un nuevo software de visualización a los ya existentes en el proyecto CCSD debido a que los mismos no cumplen con los requerimientos que se necesitan para que se produzcan visualizaciones dinámicas.

La validación del sistema haciendo uso de métricas, arrojó una solución estable, correcta y sin ambigüedades sobre la base de los requisitos pactados con los clientes. Además garantiza robustez y flexibilidad a cambios, teniendo en cuenta los resultados de las pruebas de caja negra y caja blanca y acorde al diseño propuesto.

Como resultado general del trabajo realizado se le dio cumplimiento al objetivo general propuesto, implementándose la herramienta de visualización dinámica de simulaciones de la difusión en microfluidos con componentes biológicos en la Universidad de las Ciencias Informáticas.

RECOMENDACIONES

SE RECOMIENDA:

- Realizar un estudio de optimización del Hilo Parser (hilo donde se transforman las cadenas recibidas por el socket a estructuras que serán visualizadas), utilizando técnicas de programación paralela para reducir los tiempos de respuesta.
- Incorporar nuevas funcionalidades que mejoren la experiencia del usuario para la visualización de procesos de difusión en microfluidos con componentes biológicos complejos.
- Analizar la viabilidad del uso de otras formas de comunicación entre el simulador y el visualizador, ejemplo: memoria compartida (Shmem).
- Estandarizar los datos exportados para que puedan ser usados posteriormente en otras herramientas de visualización.
- Incluir soporte de internacionalización a la aplicación para su uso por hablantes de otros idiomas.

BIBLIOGRAFÍA Y REFERENCIAS

A computational study of molecular diffusion and dynamic flow through carbon nanotubes. **Mao, Zugang y Sinnott, Susan B. 2000.** [ed.] American Chemical Society. 19, 18 de Mayo de 2000, The Journal of Physical Chemistry B, Vol. 104, págs. 4618-4624.

Alegsa, Leandro. ¿Cuál es la definición de Usuario? [En línea] [Citado el: 23 de Diciembre de 2014.] http://www.alegsa.com.ar/Dic/usuario.php.

Arnold, Ken y Gosling, James. 1997. *Lenguaje de programación Java.* Madrid : s.n., 1997.

Arquitectura de Software. **Cervantes, Humberto. 2010.** 27, Febrero de 2010, SOFTWARE GURU.

Beck, K. 2000. *Extreme Programming Explained. Embrace Change.* s.l. : Pearson Education, 2000.

Bienvenido a NetBeans y www.netbeans.org, Portal del IDE Java de Código Abierto. [En línea] [Citado el: 27 de Enero de 2015.] https://netbeans.org/index_es.html.

Calabria, Luis y Píriz, Pablo. 2003. Metodología XP. [En línea] Octubre de 2003. [Citado el: 10 de Enero de 2015.] http://fi.ort.edu.uy/innovaportal/file/2021/1/metodologia_xp.pdf.

Carrillo, I. P. 2005. *Metodología de Desarrollo de software.* 2005.

Casas, Sandra y Reinaga, Héctor. 2008. *Identificación y Modelado de Aspectos Tempranos dirigido por Tarjetas de Responsabilidades y Colaboraciones.* 2008.

Castillo, Oswaldo, Figueroa, Daniel y Sevilla, Héctor. *Fases.* [En línea] [Citado el: 18 de Abril de 2015.] http://programacionextrema.tripod.com/fases.htm.

Cruz, Josué. 2012. Visualización dinámica con la familia de microcontrolador 80c51. [En línea] 2012. [Citado el: 15 de Octubre de 2014.] http://www.buenastareas.com/ensayos/Visualizacion-Dinamica-Con-La-Familia-De/3688476.html.

Cumsille, Javiera Alejandra. 2010. Visualizador y Evaluador de Mallas Geométricas Mixtas 3D. *Memoria para Optar al Título de Ingeniero Civíl en Computación.* [En línea] 2010. [Citado el: 3 de Noviembre de 2014.] http://repositorio.uchile.cl/tesis/uchile/2011/cf-mascaro_jc/pdfAmont/cf-mascaro_jc.pdf.

Chidamber, S. R. y Kemerer, C. F. 1994. *A Metrics Suite for Object-Oriented Design, IEEE Trans. Software Engineering.* 1994. págs. 476-493. Vol. 20.

Ejiogo, O. 1991. *Software Engineering whit Formal Metrics.* s.l. : QED Technical Publishing Group, 1991.

Escobar, Cuervo. 2008. *Validez de contenido y Juicio de Expertos: una aproximación a su utilización.* 2008.

García, Javier, Rodríguez, José y Sarriegui, José. 1998. *Aprenda C++ como si estuviera en primero.* San Sebastián : s.n., 1998.

Garrido, Antonio. 2006. *Fundamentos de Programación en C++.* Primera Edición. Madrid : Grefol, 2006. 84-96477-09-6.

Jeffries, R., Anderson, A. y Hendrickson, C. 2000. *Una explicación de la programación extrema.* s.l. : Addison Wesley, 2000.

Joskowicz, Jose. 2008. Reglas y Prácticas en eXtreme Programming. [En línea] 2008. [Citado el: 2 de Mayo de 2015.] http://iie.fing.edu.uy/~josej/docs/XP%20-%20Jose%20Joskowicz.pdf.

Jurado, Edith. 2007. Una introducción a la difusión anómala. *Tesis para obtener el título de Licenciada en Física y Matemáticas.* [En línea] 2007. [Citado el: 17 de Mayo de 2015.] http://www.repositoriodigital.ipn.mx/bitstream/handle/123456789/5908/JURADO%20GALICIA%20EDITH%20Tesis%202007.pdf?sequence=1.

Kernighan, Brian W. y Ritchie, Dennis M. 2000. *El Lenguaje de Programación C.* Segunda. México : Prentice Hall Hispanoamericana, 2000. 968-880-205-0.

Laguna, Miguel A. Microsoft PowerPoint - 2-requisitos.ppt. [En línea] [Citado el: 18 de Marzo de 2015.] http://www.infor.uva.es/~mlaguna/is1/apuntes/2-requisitos.pdf.

LARMAN, Craig. 2003. *UML y Patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* México : Prentice Hall, 2003. pág. 520.

Letelier, Patricio y Penadés, Carmen. Ciclo de vida de un proyecto XP. [En línea] [Citado el: 18 de Diciembre de 2014.] http://oness.sourceforge.net/proyecto/html/ch05s02.html.

Lorenz, Mark y Kidd, Jeff. 1994. *Object-Oriented Software Metrics.* Nueva Jersey : Englewood Cliffs, 1994.

Mao, Zugang y Sinnott, Susan B. 2000. [ed.] American Chemical Society. 19, 18 de Mayo de 2000, The Journal of Physical Chemistry B, Vol. 104, págs. 4618-4624.

Martin y Serrano. 2014. Definición y propiedades de un sistema de partículas. [En línea] 2014. [Citado el: 15 de Noviembre de 2014.] http://laplace.us.es/wiki/index.php/Definici%C3%B3n_y_propiedades_de_un_sistema_de_part%C3%ADculas.

NetBeans IDE. 2013. Oracle Corporation. [En línea] 2013. [Citado el: 5 de Diciembre de 2014.] https://netbeans.org/features/.

Palacio, Juan. 2006. Gestión de proyectos ágil: conceptos básicos. [En línea] 14 de Septiembre de 2006. http://www.navegapolis.net/files/s/NST-003_01.pdf.

Palos, Juan Antonio. 2011. Modelado Gráfico con Java 3D. [En línea] 2011. [Citado el: 17 de Mayo de 2015.] https://code.google.com/p/minipaint-mconde/downloads/detail?name=Modelado%20Gr%C3%A1fico%20%28Java%203D%29%20%20-%20Juan%20Antonio%20Palos.pdf.

Piattini Velthuis, Mario G. 1996. *Análisis y Diseño Detallado de Aplicaciones Informáticas de Gestión Rama.* Madrid : RA-MA EDITORIAL, 1996.

Pressman, Roger S. 2005. *Ingeniería del Software. Un enfoque práctico.* Sexta edición. 2005. 0072853182.

Quesada, Diego. 2009. Tecnologias Java. [En línea] 7 de Enero de 2009. [Citado el: 23 de Mayo de 2015.] http://es.slideshare.net/quesada_diego/tecnologias-java-presentation.

revista-jogl_unsxx_.pdf. [En línea] [Citado el: 17 de Octubre de 2014.] https://dilancreativo.files.wordpress.com/2008/12/revista-jogl_unsxx_.pdf.

Schwaber, Ken. 2013. Scrum Manager. [En línea] 2013. [Citado el: 11 de Octubre de 2014.] http://www.scrummanager.net/.

Shannon, R. E. 1975. *Systems Simulation: The Art and Science.* Englewood Cliffs : Prentice-Hall, 1975.

TANENBAUM, ANDREW S. 2009. *Sistemas Operativos Modernos.* México : Pearson Prentice Hall, 2009. 978-607-442-046-3.

Terrance, Davis. 2004. *Learning Java Bindings for OpenGL (JOGL).* 2004.

Textos Científicos. *Clasificación de los polímeros.* [En línea] [Citado el: 27 de Febrero de 2015.] http://www.textoscientificos.com/polimeros/clasificacion.

The expert´s judgment application as a technic evaluate information and communication technology. **CABERO, Almenara J. y LLORENTE, Cejudo M. 2013.** 2013, Revista de TIC en Educación, págs. 11-22.

The random walk's guide to anomalous diffusion: a fractional dynamics approach. **Metzler, R. y Klafter, J. 2000.** 2000, Physics Reports, págs. 1-77.

University, Edinburgh Napier. Game Technology Research and Development. [En línea] [Citado el: 23 de Noviembre de 2014.] http://games.soc.napier.ac.uk/.

Vincent, María Cinta, Alvarez, Silvia y Zaragoza, José Luis. 2006. *Ciencia Y Tecnología de Polímeros.* s.l. : UPV, 2006. pág. 127. 8497059646, 9788497059640.

Visual Paradigm. 2000. Visual Paradigm International. *Visual Paradigm.* [En línea] 2000. [Citado el: 4 de Diciembre de 2015.] http://www.visual-paradigm.com.

Walton. 2001. LOS SOCKETS. [En línea] 2001. [Citado el: 3 de Mayo de 2015.] https://sistemas.uniandes.edu.co/~isis1301/dokuwiki/lib/exe/fetch.php?media=recursos:sockets.pdf.

Zhang, Y., y otros. 2000. *Una explicación de la programación extrema.* s.l. : Addison Wesley, 2000.

GLOSARIO

Animación: Es el método para crear la ilusión de movimiento a través de la reproducción continua de imágenes estáticas. También podría entenderse como un conjunto de interpolaciones creadas por un software a través de cuadros "claves".

API: Interfaz de programación de aplicaciones, es el conjunto de funciones y procedimientos que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Cliente: Sistema informático (ordenador) que solicita ciertos servicios y recursos de otro ordenador (denominado servidor), al que está conectado en red.

GUI: (Graphical User Interface/Interfaz Gráfica para el Usuario) El GUI, o Interfaz Gráfica para el Usuario, es una forma de interacción entre el usuario y la computadora. En lugar de escribir comandos como se hace tradicionalmente en el DOS o UNIX el usuario utiliza un mouse o algún otro periférico para seleccionar íconos y menús. El Mac OS y el Windows son sistemas operativos con GUI.

JVM: Es un máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

Luz ambiental/ Ambient light: Es una luz artificial que emite luz de forma global y pareja una escena 3D. Permitiendo ver los objetos mientras no existían luces creados por el usuario. La mayoría de las veces es representada por innumerables reflejos en todas las superficies existentes.

Mesh/Malla Poligonal: Es una forma 3D usada en computación gráfica basada en polígonos, los polígonos son compuestos por vértices, ejes y caras, en inglés, vertex, edge, faces. Las caras comúnmente tienen forma de triángulo o cuadriláteros, la mayoría de los programas modeladores 3D las utiliza para visualizar o renderizar una superficie. A mayor cantidad de polígonos, mayor precisión. Algunos de los formatos de mallas poligonales son stl, obj, vrml, ply, x3d, etc. Un ejemplo de programas que manejan mallas poligonales o mesh son 3DsMax, Maya entre otros.

Modelo 3D: un modelo en 3D es un "mundo conceptual en tres dimensiones".

- Desde un punto de vista técnico, es un grupo de fórmulas matemáticas que describen un "mundo" en tres dimensiones.
- Desde un punto de vista visual, un modelo en 3D es una representación esquemática visible a través de un conjunto de objetos, elementos y propiedades que, una vez procesados (renderización), se convertirán en una imagen en 3D o una animación 3d.

Por lo general, el modelo visual suele ser el modelo 3d que las diseñadores manejan, dejando las fórmulas a procesos computacionales. Esto es así, porque lo que el modelo en 3d visual representa se acerca más a la imagen en 3D final que se mostrará al renderizarse.

Parámetro: Es un dato variable que indica a un programa informático cómo debe comportarse en cada situación.

Puerto de comunicaciones/Puerto: Punto de acceso a un ordenador o medio a través del cual tienen lugar las transferencias de información (entradas / salidas), del ordenador con el exterior y viceversa (vía TCP/IP).

PNG: Es un formato gráfico basado en un algoritmo de compresión sin pérdida para bitmaps no sujeto a patentes. Este formato fue desarrollado en buena parte para solventar las deficiencias del formato GIF y permite almacenar imágenes con una mayor profundidad de contraste y otros importantes datos.

Protocolo: Es el sistema, reglas y conjunto de normas que establece, gobierna y permite la comunicación entre dispositivos informáticos (la transferencia de datos).

Red: Grupo de ordenadores o dispositivos informáticos conectados entre sí a través de cable, línea telefónica, ondas electromagnéticas (microondas, satélite, etc), con la finalidad de comunicarse y compartir recursos entre ellos.

Renderización: La renderización es el proceso de generar una imagen (imagen en 3D o una animación en 3D) a partir de un modelo, usando una aplicación de computadora. El modelo es una descripción en tres dimensiones de objetos en un lenguaje o estructura de datos estrictamente definidos. El modelo debería contener geometría, punto de vista, textura e información de iluminación. La imagen resultado de la renderización es una imagen digital (raster).

Servidor: Sistema informático (ordenador) que presta ciertos servicios y recursos (de comunicación, aplicaciones, ficheros, etc.) a otros ordenadores (denominados clientes), los cuales están conectados en red a él.

3D: Tridimensional, comprende las dimensiones ancho, largo y profundo, creando una visión cercana a la realidad.

ANEXOS

Anexo1.

Historia de Usuario	
No.: 3	**Nombre:** Visualizar polímeros.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 1
Descripción: El sistema debe mostrar en pantalla un polímero.	
Información adicional (Observaciones): Da cumplimiento al RF3 ”Visualizar polímeros”.	

Tabla 18. HU 3.

Historia de Usuario	
No.: 4	**Nombre:** Recibir datos del simulador.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 1 semana	**Iteración Asignada:** 2
Descripción: El sistema debe ser capaz de recibir y entender los datos que son enviados por el simulador.	
Información adicional (Observaciones): Da cumplimiento al RF4 ”Recibir datos del simulador”.	

Tabla 19. HU 4.

Historia de Usuario	
No.: 5	**Nombre:** Establecer área límite del escenario gráfico.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 2
Descripción: El sistema debe ser capaz de establecer un área donde las estructuras van a ser visualizadas, fuera de esta región no deben existir imágenes.	
Información adicional (Observaciones): Da cumplimiento al RF5 ”Establecer área límite del escenario gráfico”.	

Tabla 20. HU 5.

Historia de Usuario	
No.: 6	**Nombre:** Cambiar ángulo de la cámara.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 2
Descripción: El usuario debe tener la opción de visualizar las estructuras desde los planos xy, xz y yz.	
Información adicional (Observaciones): Da cumplimiento al RF6 ”Cambiar ángulo de la cámara”.	

Tabla 21. HU 6.

Historia de Usuario	
No.: 7	**Nombre:** Reiniciar la vista.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 1 semana	**Iteración Asignada:** 3
Descripción: El sistema debe permitir que la vista sea reiniciada cuando el usuario lo desee para facilitarle el entendimiento de la simulación en curso.	
Información adicional (Observaciones): Da cumplimiento al RF7 ”Reiniciar la vista”.	

Historia de Usuario	
No.: 8	**Nombre:** Habilitar navegación.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 1 semana	**Iteración Asignada:** 3
Descripción: El usuario debe ser capaz de habilitar y deshabilitar las siguientes opciones de navegación: el movimiento por cursores, el zoom, la rotación y la traslación.	
Información adicional (Observaciones): Da cumplimiento al RF8 ”Habilitar navegación”.	

Tabla 23. HU 8.

Historia de Usuario	
No.: 9	**Nombre:** Mostrar el diagrama de comportamiento de la simulación.
Usuario: Especialista en simulación.	
Prioridad en el Negocio: Alta	**Nivel de Complejidad:** Alta
Estimación: 2 semanas	**Iteración Asignada:** 3
Descripción: El sistema cada vez que se refresque la simulación debe mostrar el diagrama de comportamiento de la misma, de tal manera que permita que el usuario realice análisis.	
Información adicional (Observaciones): Da cumplimiento al RF9 ”Mostrar el diagrama de comportamiento de la simulación”.	

Anexo 2.

Tarjetas CRC	
Clase: Color.	
Responsabilidades: devolverColor	**Colaboraciones:** Particula Polimero Superficie

Tabla 26. Tarjeta CRC de la clase Color.

Tarjetas CRC	
Clase: Particula.	
Responsabilidades: dibujarParticulas crearUnaParticula transformarParticulasaJava	**Colaboraciones:** Visualizador Color

Tabla 25. Tarjeta CRC de la clase Partícula.

Tarjetas CRC	
Clase: Superficie.	
Responsabilidades: pintarTriangulos puntosTriangulo transformarPuntosSuperficieaJava	**Colaboraciones:** Color Visualizador Matriz

Tabla 27. Tarjeta CRC de la clase Superficie.

Tarjetas CRC	
Clase: Polimero.	
Responsabilidades: crearUnaParticula pintarLineasPolimero transformarPolimeroaJava	**Colaboraciones:** Particula Visualizador Color

Tabla 29. Tarjeta CRC de la clase Polimero.

Tarjetas CRC	
Clase: Imagenes.	
Responsabilidades: setImg paint	**Colaboraciones:** Visualizador HiloSocket

Tabla 28. Tarjeta CRC de la clase Imagenes.

Tarjetas CRC	
Clase: HiloSocket.	
Responsabilidades: run	**Colaboraciones:** Visualizador Imagenes

Tabla 30. Tarjeta CRC de la clase HiloSocket.

Tarjetas CRC	
Clase: HiloServer.	
Responsabilidades: run	**Colaboraciones:** Visualizador

Tabla 31. Tarjeta CRC de la clase HiloServer.

Anexo 3.

Caso de prueba para el camino básico 2	
Descripción	Se comprueba si en la cadena pasada por parámetro existe al menos una estructura de tipo superficie.
Condiciones de ejecución	Que se haya ejecutado al menos una simulación y exista al menos una partícula.
Entrada	entrada
Resultado esperado	Que la aplicación siga ejecutando el resto del código.

Tabla 32. Caso de prueba para el camino básico 2.

Caso de prueba para el camino básico 3	
Descripción	Se comprueba si en la cadena pasada por parámetro existe al menos una estructura de tipo polímero.
Condiciones de ejecución	Que se haya ejecutado al menos una simulación, exista al menos una partícula y exista al menos una estructura de tipo superficie.
Entrada	entrada
Resultado esperado	Que la aplicación siga ejecutando el resto del código.

Tabla 33. Caso de prueba para el camino básico 3.

Caso de prueba para el camino básico 4	
Descripción	Se le asigna el valor verdadero a la variable booleana que indica si se está visualizando al menos una estructura de cada tipo.
Condiciones de ejecución	Que se haya ejecutado al menos una simulación, exista al menos una partícula, exista al menos una estructura de tipo superficie y por último que exista a lo sumo una estructura de tipo polímero.
Entrada	entrada
Resultado esperado	Que la aplicación siga ejecutando el resto del código.

Tabla 34. Caso de prueba para el camino básico 4.

Anexo 4.

Caso de Prueba de caja negra	
Código: 2	**HU:** 2
Nombre: Visualizar superficies.	
Descripción: El sistema debe mostrar en pantalla una superficie, y debe permitirle al usuario poder cambiar su color y escoger entre tres modos de vista: superficie, triángulos y puntos. **Condiciones de ejecución:** Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Seleccionar alguna de las opciones del menú Superficie. **Resultados esperados:** La herramienta debe mostrar en pantalla una superficie.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 35. Caso de Prueba de caja negra HU 2.

Caso de Prueba de caja negra	
Código: 3	**HU:** 3
Nombre: Visualizar polímeros. **Descripción:** El sistema debe mostrar en pantalla un polímero.	
Condiciones de ejecución: Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Recibir el dato que envía el simulador con las características del polímero.	
Resultados esperados: La herramienta debe mostrar en pantalla un polímero.	

Caso de Prueba de caja negra
Evaluación de la prueba: Prueba Satisfactoria.

Tabla 36. Caso de Prueba de caja negra HU 3.

Caso de Prueba de caja negra	
Código: 4	**HU:** 4
Nombre: Recibir los datos del simulador.	
Descripción: El sistema debe capturar todos los datos que son enviados por el simulador. **Condiciones de ejecución:** Se debe estar ejecutando alguna simulación. **Entrada/Pasos de ejecución:** Recibir los datos que envía el simulador con las características de las estructuras.	
Resultados esperados: La herramienta debe refrescar la imagen en pantalla de una manera dinámica. **Evaluación de la prueba:** Prueba Satisfactoria.	

Tabla 37. Caso de Prueba de caja negra HU 4.

Caso de Prueba de caja negra	
Código: 5	**HU:** 5
Nombre: Establecer área límite del escenario gráfico.	
Descripción: El sistema debe establecer un área de visualización, fuera de ella no pueden haber imágenes.	

Caso de Prueba de caja negra
Condiciones de ejecución: Se debe estar ejecutando alguna simulación.
Entrada/Pasos de ejecución: Recibir los datos que envía el simulador con las características de las estructuras y representarlas en el área de visualización.
Resultados esperados: La herramienta debe mostrar a todas las estructuras dentro del área predefinida.
Evaluación de la prueba: Prueba Satisfactoria.

Tabla 38. Caso de Prueba de caja negra HU 5.

Caso de Prueba de caja negra	
Código: 6	**HU:** 6
Nombre: Cambiar ángulo de la cámara.	
Descripción: El sistema debe ser capaz de cambiar el ángulo de la cámara, mostrando la vista en los planos xy, yz y xz.	
Condiciones de ejecución: Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Detectar los eventos del mouse y del teclado que aparecen en el momento del cambio.	
Resultados esperados: La herramienta debe mostrar a todas las estructuras desde diferente puntos de vista.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 39. Caso de Prueba de caja negra HU 6.

Caso de Prueba de caja negra	
Código: 7	**HU:** 7
Nombre: Reiniciar la vista.	
Descripción: El sistema debe ser capaz de retornar la vista hasta la posición inicial que ocupaba en el momento del inicio de la simulación.	
Condiciones de ejecución: Se debe estar ejecutando alguna simulación. **Entrada/Pasos de ejecución:** Detectar los últimos datos que fueron enviados por el simulador. **Resultados esperados:** La herramienta debe mostrar a todas las estructuras como si no se hubieran rotado o trasladado.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 40. Caso de Prueba de caja negra HU 7.

Caso de Prueba de caja negra	
Código: 8	**HU:** 8
Nombre: Habilitar navegación. **Descripción:** El sistema debe ser capaz de activar o desactivar la navegación por teclado, la rotación, la traslación y el zoom.	
Condiciones de ejecución: Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Detectar los eventos del mouse que aparecen sobre los chekbox del menú navegación.	

Caso de Prueba de caja negra	
Resultados esperados: La herramienta debe permitir que las estructuras sean trasladadas, rotadas y que se puedan acercar y alejar.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 41. Caso de Prueba de caja negra HU 8.

Caso de Prueba de caja negra	
Código: 9	**HU:** 9
Nombre: Mostrar el diagrama de comportamiento de la simulación.	
Descripción: El sistema debe ser capaz de mostrar en una ventana el diagrama de comportamiento de la simulación, este es enviado por el simulador.	
Condiciones de ejecución: Se debe estar ejecutando alguna simulación.	
Entrada/Pasos de ejecución: Detectar los datos de las imágenes que son enviados a través del socket.	
Resultados esperados: La herramienta debe mostrar un diagrama de comportamiento actualizado cada vez que se refresque la vista.	
Evaluación de la prueba: Prueba Satisfactoria.	

Tabla 42. Caso de Prueba de caja negra HU 9.

Anexo 5.

Síntesis biográfica de Dr. Jorge Gulín González

Jorge Gulín González se graduó en el año 1995 en la Universidad de La Habana de Licenciatura en Ciencias Físicas. Del 1996 al 2000 realizó su doctorado en Ciencias Físicas. Desde el año 1996 hasta el 1997 se desempeñó como investigador en adiestramiento en el departamento de Física de la Cuidad Universitaria José A. Echeverría (CUJAE). En el 2002impartió clases en la CUJAE como Profesor Asistente. Desde el 2008 hasta la actualidad es profesor titular en la Universidad de las

Ciencias Informáticas (UCI). Del 2008 hasta el 2013 fue Director de Investigaciones en la UCI. Sus líneas de investigación son Física computacional, Física-Química, materia condensada, Bioinformática, materiales nanoporosos, catalizadores sólidos y difusión anómala. En el 2002 recibió el Premio Nacional del CITMA en la modalidad de Investigador Joven. En el 2004 se le otorga la membresía de la fundación A.v.Humboldt's de University of Leipzig (Alemania). En los años 2003 y 2009 fue premio de la Academia de Ciencias de Cuba. Desde el 2008 hasta el 2013 se desempeñó como asistente y editor de la Revista Cubana de Ciencias Informáticas. Es actualmente miembro del Consejo Científico de la CUJAE y la UCI, miembro de la Sociedad Cubana de Física y de la Sociedad Cubana de Matemática y Computación. Es miembro del Claustro de la Universidad de Sassari en Italia, también del claustro para doctorados en Bioinformática de la Universidad de La Habana. Fue presidente del comité científico de los congresos: COMPUMAT 2013, UCIENCIA 2012 y UCIENCIA 2013.Actualmente cuenta con más de 40 publicaciones.

Anexo 6. Síntesis biográfica de Lic. Edisel Navas Conyedo

Edisel Navas Conyedo se graduó en el año 2003 con título de oro en la Universidad de La Habana de la carrera Licenciatura en Física. Desde el año 2012 es estudiante de doctorado en la especialidad de Física. Del año 2003 al 2004 se desempeñó como Profesor Adiestrado en el Instituto de Materiales y Reactivos (IMRE) de la Universidad de La Habana. Desde 2004 hasta 2005 como Profesor Adiestrado impartió clases de Mecánica Teórica I y II y Física General I y II en la Facultad de Matemática Física y Computación y la Facultad de Ingeniería Mecánica respectivamente. En la Universidad de las Ciencias Informáticas desde el año 2005 hasta el 2007 impartió las asignaturas Física General I y II como Profesor Instructor. Desde el 2007 hasta el 2013 se desempeñó como Profesor Asistente en la UCI impartiendo clases de Física y Matemática IV. Desde 2013 hasta la actualidad es investigador y Profesor Auxiliar perteneciente al Centro de Estudios de Matemática Computacional y Coordinador del grupo de Matemática y Física Computacionales de la UCI. En el 2012 fue invitado como joven investigador al Instituto de Sistemas Complejos y Simulaciones Avanzadas de Jülich, Alemania. Ha cursado numerosos estudios de postgrado y ha participado en eventos como el XIII Symposio y XI Congreso de la Sociedad Cubana de Física, el Congreso internacional COMPUMAT 2013, XVIII Escuela de Verano de Ciencias y Tecnologías de los Materiales, el IV Simposio de Foto-Biología-Física-Química Fotociencias 2008, el XI Simposio y IX Congreso de la Sociedad Cubana de Física, V Taller de Tecnologías Láser, TecnoLáser 2007, XIV Escuela de Verano de Ciencias y Tecnologías de los Materiales, el IV Taller de Tecnologías Láser, TecnoLáser 2005, III Simposio de Fotobiología, Fotofísica y Fotoquímica, FOTOCIENCIAS 2005, a la Escuela y Conferencia sobre Física Estadística y Aplicaciones Interdisciplinarias, V Escuela Regional de Cristalografía y Difracción, entre otros. Actualmente cuenta con 10 publicaciones y es jefe del proyecto Caracterización Computacional de Sistemas Difusivos.

Anexo 7. Síntesis biográfica de MSc. Yailen Costa Marrero

Yailen Costa Marrero se graduó en el año 2003 en la Universidad de La Habana de la carrera Licenciatura en Física. En el año 2014 terminó la maestría en la especialidad Ciencia y Tecnología de Materiales-Física. Del año 2004 al 2005 impartió clases como Profesor Adiestrado de Física General I y II en las facultades de Ingeniería Mecánica y Ciencias Empresariales en la Universidad Central de las Villas. Como Profesor Instructor impartió las asignaturas Física I y II en la Facultad 4 de la Universidad de las Ciencias Informáticas desde 2005 hasta 2007. Desde el 2007 hasta el 2012 impartió clases de Física I y II como Profesor Asistente en la Universidad de las Ciencias Informáticas. Actualmente es Asesora del Departamento Docente Metodológico Central de Ciencias Básicas de la UCI e imparte clases de las asignaturas Física y Fundamentos Físicos. Tiene diversos cursos de postgrado en la especialidad de Física. Ha participado en eventos tales como el XIII Simposio y XI Congreso de la Sociedad Cubana de Física, el VI Taller de Física y Matemática Computacionales de la VI Conferencia Científica de la Universidad de las Ciencias

Informáticas UCIENCIA, la XIV Escuela Internacional de Ciencia y Tecnología de Materiales, el XXVII Congreso Latinoamericano de Química y VI Congreso Internacional de Química e Ingeniería Química, el III Simposio de Fotobiología, Fotofísica y Fotoquímica, FOTOCIENCIAS 2005, V Escuela Regional de Cristalografía y Difracción, entre otros. Actualmente cuenta con 6 publicaciones y ha formado parte de los proyectos “Proyecto de Innovación Pedagógica, Física” y “Caracterización Computacional de Sistemas Difusivos”.