# Universidad de las Ciencias Informáticas

## Facultad 8


# Análisis y Diseño del primer ciclo de desarrollo del Simulador FisMat del proyecto Alfaomega

Trabajo de diploma para optar por el título de Ingeniero en Ciencias Informáticas

Autor: Viviana Font Merelles.

Tutor: Ing. Isyed De La Caridad Rodríguez Trujillo.

Ciudad de La Habana

2010

"Año 52 de la Revolución"

**Declaración de autoría**

Por este medio declaro ser autora 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 _____.

| | |
|---|---|
| Viviana Font Merelles | Isyed de la Caridad Rodríguez Trujillo |
| _____ | _____ |
| Firma del Autor | Firma del Tutor |

*"Los hombres necesitan quien les mueva a menudo la compasión en el pecho y las lágrimas en los ojos.; y les haga el supremo bien de sentirse generosos."*

*José Martí.*

# *Agradecimientos*

A mi mami y a mi papi que más que por mí, siento que todo lo hago por ellos y para ellos, porque se lo merecen después de todo lo que me han dado.

A mi hermanita del alma, por estar siempre ahí.

A mis abuelas Nelthys y Ñica, que se han preocupado demasiado por mí y siempre me han consentido en todo, nunca he dejado de pensar en ellas.

A mi abuelo Guille, por sacarme de los líos siempre que lo he necesitado.

A mi abuelo Mario, que sé que está muy orgulloso, donde quiera que esté.

A mi tío Carlos por preocuparse siempre porque me alimentara.

A mi primo Carlitos, por servirme de inspiración para cumplir este sueño que sé que también hubiera sido el de él.

A mi tío Eduardito, a mi primito Kevin y al que está por llegar, por estar siempre pendientes.

A mi tía Lucy y a mi primo Pavelín por confiar en mí todo el tiempo.

A Reynaldo, por apoyarme desde lejos.

A Maritza, Neima, Ivette, Katia y Lili, loquillas ustedes hacen que no me quiera ir de aquí. Gracias por su apoyo en este cortísimo tiempo en que nos hemos conocido, que lástima no haberlas conocido antes.

A mi casi compañera de tesis, Lianne, por apoyarme tanto en nuestras travesías por las pruebas de nivel.

A Yssel y Enelys por ser mis amigas desde pequeñitas, por tantos años compartiendo alegrías, discusiones, travesuras en bicicleta, disco, chismes, historias, secretos, en fin, miles de cosas que las hacen ser especiales.

A Talía y Made, por aguantarme 3 años, aún cuando no las soportaba. Gracias por su inmenso cariño y por acompañarme en las buenas y en las malas.

A mis amigas Yudianna y Sindy, aunque no estén aquí las tengo siempre presentes. Gracias por tantas ocupaciones para conmigo y darme cariño, aunque sea de lejos aquí llega.

A mis eternos compañeros Erianne, Yusle, Abel y Gilbe por tantas travesuras y por soportarme todo este tiempo, gracias por su paciencia.

A Carlos, mi tati, por creer en mí, aún cuando yo no lo hacía y por alentarme cuando más lo necesité.

A mis suegros Iraida y Carlos, y a mis cuñados Jose y Leidys, por quererme como a una más de la familia.

A mi cuñi Rodney, por su sinceridad y sabias palabras.

A Rita, Jesús, Walber y Wilkie, por quererme aún a pesar del tiempo.

A Ivis, Maykel y Luis, por su preocupación y cariño.

A todos los profesores que han formado en mí los valores necesarios para ser una buena profesional.

A mi tutora, por exigirme ser mejor cada día.

A los profesores del proyecto Alfaomega, por socorrerme siempre que estuve estancada. Gracias por su sabiduría.

A la vida por haberme conducido hasta aquí.

A todas las personas que colaboraron aportando su granito de arena y a las que han comprendido que el sacrificio vivido fue de gran importancia porque hoy se recogen los frutos. A los que de una forma u otra dejaron en mí una huella.

Muchas gracias.

# *Dedicatoria*

A mis padres que lo han dado todo para verme graduada.

A mi hermana que la adoro, espero siempre ser para ti más que una hermana, una amiga.

A mi primo querido, estés donde estés, sé que hubieras estado orgulloso de ver en lo que me he convertido y todo lo que he sido capaz de hacer.

A mis abuelos y tíos que los quiero mucho.

# *Resumen*

La simulación se ha desarrollado crecientemente a nivel mundial y se han podido observar las mejoras que ofrece esta técnica en diferentes ramas como la educación, la salud y la industria, por solo mencionar algunas. A la facultad 8 de la Universidad de las Ciencias Informáticas se le encomendó la tarea de crear una serie de contenidos educativos, con el propósito de facilitar la posterior construcción de una plataforma de aprendizaje online que consolide el sistema educacional en México. En estas condiciones surge la idea de crear un simulador de fenómenos físicos que constituya una herramienta interactiva y amigable para el estudiante y el profesor. Para ello, se hace necesario realizar el análisis y diseño del simulador FisMat que se desarrolla, permitiendo un mejor entendimiento del sistema que se desea implementar. Para realizar tal tarea se utiliza la metodología RUP (Proceso Unificado de Desarrollo) debido a que el contratista se encuentra en otro país y no existe una comunicación constante con el mismo, pues esta brinda la posibilidad de documentar todo el avance del proceso para que el cliente puede mantener un control estricto sobre el mismo. Además, se utiliza UML como lenguaje de modelado y la herramienta Visual Paradigm como apoyo para realizar los diagramas y modelos pertinentes al simulador. De esta manera se obtiene el diagrama de casos de uso del sistema, un listado de los requisitos funcionales del simulador, la descripción de los casos de uso y el análisis y diseño perteneciente al módulo Instrumentos del simulador.

**PALABRAS CLAVE**

Simulador, Análisis y Diseño.

# Índice

**Índice de Tablas**

**Índice de Figuras**

## Introducción:

La Universidad de las Ciencias Informáticas (UCI), fomenta el desarrollo de la industria del software en Cuba, mediante proyectos productivos. Alfaomega es un proyecto que tiene como objetivo realizar una plataforma de aprendizaje online, que brinde la posibilidad de prepararse y estudiar para su formación a estudiantes y profesores. Para ello es necesario elaborar varios contenidos educativos, entre los que se encuentra el simulador FisMat.

FisMat es un software que permite realizar la simulación de determinados fenómenos físicos, descritos por un modelo matemático. Actualmente existe un simulador que no cumple con todas las expectativas, pues tiene funcionalidades limitadas. Su diseño gráfico es pobre, al igual que la cantidad de información que aporta. Además, está realizado en lenguaje Basic, lo cual limita su funcionamiento multiplataforma y dificulta el proceso de instalación y distribución, por lo que se hace difícil su mantenimiento. Tampoco se documentó la ingeniería de la aplicación. Toda esto provoca que el simulador existente no refleje los elementos fundamentales del contenido educativo. Se hace necesario comprender bien los procesos que intervienen en el producto a realizar, para modelar un simulador que presente las principales características de estas aplicaciones y que corresponda con el contenido que se quiere reproducir.

De aquí la necesidad de realizar el análisis y diseño del nuevo simulador al ser de vital importancia para la implementación del mismo. De esta manera el **problema a resolver** consiste en: ¿Cómo modelar el simulador FisMat de manera que facilite su desarrollo?

El **objetivo general** del presente trabajo no es más que: Realizar el análisis y diseño del primer ciclo de desarrollo del simulador FisMat del proyecto Alfaomega.

Para desarrollar el objetivo general se plantean los siguientes **objetivos específicos**:

- Aplicar la ingeniería de requisitos.
- Realizar la captura de requisitos del simulador FisMat.
- Especificar los requisitos funcionales y no funcionales del simulador.
- Agrupar los requisitos según sus similitudes en casos de usos.
- Modelar una propuesta de módulos a realizar.
- Describir los casos de usos identificados en cada módulo.

- Realizar el análisis y diseño de un módulo del simulador.

El **objeto de estudio** de este lo constituyen los procesos asociados al desarrollo de simuladores y el **campo de acción** se encuentra enmarcado en el modelado del simulador FisMat del proyecto Alfaomega.

Por tanto la idea a defender es: Modelar el simulador FisMat aplicando la Ingeniería de Requisitos y el análisis y diseño de un módulo, para facilitar el desarrollo de una herramienta que permita simular adecuadamente los contenidos educativos.

Para dar cumplimiento a los objetivos antes expuestos se ha propuesto la realización de las siguientes tareas de investigación:

- Analizar las características de las principales metodologías de desarrollo.
- Examinar la solución existente del FisMat actual, para definir los problemas que presenta y los perfeccionamientos que necesita.
- Describir los Casos de Uso según la metodología seleccionada.
- Capturar los requisitos funcionales del simulador.
- Agrupar los requisitos según sus similitudes en casos de usos y modelar una propuesta de módulos a realizar.
- Realizar el diagrama de Casos de Uso.
- Realizar el diagrama de clases del análisis y del diseño de un módulo.
- Realizar los diagramas de secuencia de un módulo.

Con el desarrollo de lo planteado anteriormente se esperan obtener los siguientes resultados:

- Listado de requisitos funcionales del sistema.
- Diagrama de casos de uso.
- Descripción de los casos de usos que contiene la propuesta.
- Análisis y diseño de un módulo.

Con el fin de resolver y dar cumplimiento a los objetivos y las tareas propuestas se han utilizado los siguientes métodos científicos, categorizados como métodos teóricos:

Se utiliza el **método dialéctico** como método rector, está presente en la realización de todas las tareas, pues este permite abordar el objeto de investigación considerándolo en su desarrollo y teniendo en cuenta la interrelación entre todos sus componentes. Busca las contradicciones existentes y explica los cambios cualitativos que se producen en el sistema y dan paso a un nuevo objeto. Como parte del método dialéctico se utilizarán la inducción y la deducción; y el análisis y la síntesis como procedimientos para comprender la esencia del fenómeno que se investiga.

Además, se hace uso del **método de la modelación**, específicamente el enfoque de sistema, que permite crear abstracciones con el objetivo de explicar la realidad. El lenguaje de modelado UML permitirá reflejar la estructura, relaciones internas y características de la solución a través de diagramas.

El **método histórico – lógico** permite abordar el objeto desde el punto de vista diacrónico y sincrónico. Permite, además, comprender los fundamentos teóricos que sustentan la propuesta e identificar la pertinencia de cada uno de ellos para la realización del análisis y del diseño. Se evidencia mayormente en el momento de estudiar las metodologías, herramientas y lenguajes de modelado, así como en la investigación del modelado de simuladores.

El presente trabajo se apoya, también, en el **método sistémico** que permite abordar todos los componentes del diseño en su interrelación, y establecer relaciones de subordinación y coordinación entre los mismos. Se pone de manifiesto, principalmente, en la realización del estudio del arte de las principales metodologías de desarrollo que permiten realizar una correcta captura de requisitos.

Y finalmente el método empírico de la **observación** se utiliza para constatar las deficiencias del simulador vigente a partir de indicadores preestablecidos, específicamente a la hora de analizar la solución existente del FisMat actual, para definir los problemas que presenta y los perfeccionamientos que necesita.

El documento estará conformado por 3 capítulos estructurados de la siguiente manera:

Capítulo 1: “Fundamentación Teórica”: Se abordará el estudio del arte del tema que se investiga. Se destacarán las metodologías de desarrollo de software más conocidas actualmente, se realizará un estudio comparativo entre ellas, y se propondrán además: la herramienta, el lenguaje de modelado y la metodología que se utilizarán para el modelado.

Capítulo 2: “Descripción del Sistema”: Se describirán los conceptos de Modelo de Negocio y Modelo de Dominio. Se elaborará una descripción de los conceptos que están presentes en el simulador y se realizará un diagrama representativo de los mismos. Además se especificarán los requisitos funcionales y no funcionales del simulador y contará, también, con las descripciones de los casos de uso del sistema.

Capítulo 3: “Análisis y Diseño del Sistema”: Estará constituido por los diagramas de clases del análisis y los diagramas de clases del diseño de uno de los módulos del simulador, así como los diagramas de interacción correspondientes a cada caso de uso.

## Capítulo 1: "Fundamentación Teórica"

### 1. Introducción:

La simulación es muy utilizada actualmente en muchos campos, es una herramienta que permite representar un fenómeno mediante otro que lo hace mucho más simple y entendible.

La simulación no solamente se enfoca en el modelado de sistemas de alto nivel, pues ha evolucionado hasta el punto de que se centra más en la creación de prototipos de software, consiguiendo un ahorro de tiempo y dinero a las empresas en su desarrollo.

El principal objetivo de este capítulo es ubicar al lector en el tema del trabajo, abordándose en él conceptos asociados con el contenido del mismo. También se hace referencia a los principales programas de simulación existentes, brindando sus características generales. Además se dan a conocer algunas metodologías y herramientas para el desarrollo de software.

### 1.1 ¿Qué es un simulador?

Un simulador es un sistema de software que imita tanto el comportamiento de un sistema del mundo real, como los procesos de entrada que manejan o controlan el sistema simulado. Las simulaciones pueden usarse para obtener conocimiento acerca de sistemas existentes, para predecir su comportamiento y para propósitos de enseñanza (Cuéllar-Vázquez, y otros).

Una de las características claves de la simulación es la habilidad de modelar el comportamiento del sistema considerando el progreso del tiempo. Las características inherentes a las aplicaciones que simulan algún sistema deben ser consideradas desde la etapa de análisis para reflejar los requerimientos asociados durante la etapa de diseño del sistema (Cuéllar-Vázquez, y otros).

Tal y como se ha hecho referencia en diversas publicaciones, dentro de diferentes clasificaciones de software educativo, los simuladores constituyen una poderosa

herramienta para lograr el cumplimiento de diversos objetivos de enseñanza en varias disciplinas.

A pesar de que la simulación como técnica de enseñanza - aprendizaje surge antes de la aparición de la primera computadora electrónica y que su valor es reconocido no sólo en el campo del software educativo, la simulación por computadoras ofrece grandes ventajas en el proceso docente, pues a través de la misma se puede lograr imitar, reproducir, replicar, con un alto grado de similitud procesos, objetos y fenómenos del mundo real, algunos de los cuales no podrían ser presentados a los alumnos y otros, mucho menos permitirían interactuar con ellos.

### 1.1.1 Tipos de simuladores

Según Alessi y Trollip (Alessi, y otros, 1985) los simuladores pueden ser divididos en cuatro grupos o categorías las cuales son:

- Simuladores físicos.
- Simuladores procedimentales.
- Simuladores situacionales.
- Simuladores de procesos.

En la primera categoría, o sea, los simuladores físicos, se encuentran aquellos en que un objeto físico es presentado a través de la pantalla para que el estudiante pueda usarlo o aprender de él.

Aunque las simulaciones físicas son muy usuales, generalmente juegan un papel secundario y existen para su empleo en simulaciones procedimentales.

Los simuladores procedimentales, más que ser usados para que el estudiante aprenda sobre un objeto físico, proporcionan una vía para que el alumno adquiera los conocimientos y habilidades necesarias para aprender a usarlo.

La tercera categoría, en este caso los simuladores situacionales, son empleados para reflejar las actitudes y el comportamiento del ser humano ante diferentes situaciones y explorar los efectos de diferentes tratamientos de una situación y les posibilitan al estudiante jugar diferentes papeles en la propia simulación por lo que en casi toda

simulación situacional el alumno es una parte integrante de la misma y juega uno de los papeles principales.

Por último, las simulaciones de procesos son sistemas en los que el estudiante, a diferencia de las simulaciones situacionales, no juega un papel activo, ni constantemente manipula como en las simulaciones físicas o procedimentales sino que el alumno solamente se limita a seleccionar diferentes valores para cada uno de los parámetros que se contemplan en la simulación y luego observa cómo ocurre el proceso.

### 1.1.2 Ejemplos de Simuladores de fenómenos físicos

#### 1.1.2.1 Phet:

Es un interesante conjunto de simuladores didácticos e interactivos diseñados para enseñar los conceptos básicos de diferentes fenómenos físicos. Con Phet se puede experimentar con la gravedad, tiros parabólicos, señales de radio y efectos electromagnéticos, construir sencillos circuitos eléctricos, representar ecuaciones en gráficas, experimentar con señales láser, entre otras posibilidades. Cada simulador de Phet incluye los controles necesarios para configurar los parámetros básicos del fenómeno que estudia.

#### 1.1.2.2 Phun:

Creado por Emil Ernerfeldt, un estudiante sueco y se llama Phun (juego de palabras con fun, «divertido»). Incluye gravedad, fluidos y muchas más opciones para crear pequeños mundos físicos virtuales. Es un programa que se puede descargar libremente para usos no comerciales. Funciona en Windows y Linux, próximamente estará disponible para su uso en otros sistemas operativos.

#### 1.1.2.3 Interactive Physics:

Interactive Physics es como una pizarra electrónica, donde se ven imágenes y gráficos que cobran vida, facilitando a los alumnos el entendimiento de los fenómenos físicos. Con este programa el profesor puede simular sus experimentos de física en la PC. Por

ejemplo se pueden variar los parámetros de fricción, gravedad, tiempo, fuerza, velocidad inicial, etc. y ver las variables en forma instantáneas. Se pueden demostrar principios básicos de la física, como así también, explorar nuevas situaciones. Es una de las herramientas ideales para la creación de experiencias educativas y experimentos.

Con las herramientas provistas se pueden abordar estudios de estabilidad, caída libre, velocidad con respecto a distintos sistemas de referencia, aceleración, primera y segunda ley de Newton, distinción entre masa y peso, cinemática y dinámica rotacional, colisiones, trabajo y potencia, etc. También se pueden crear nuevas experiencias definiendo los entornos y las condiciones iniciales.

## 1.2 Metodologías de Desarrollo de Software:

### 1.2.1 Rational Unified Process (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process (Proceso Unificado de Desarrollo), divide en 4 fases el desarrollo del software:

- Conceptualización (Concepción o Inicio): Se describe el negocio y se delimita el proyecto describiendo sus alcances con la identificación de los casos de uso del sistema.
- Elaboración: Se define la arquitectura del sistema y se obtiene una aplicación ejecutable que responde a los casos de uso que la comprometen. A pesar de que se desarrolla a profundidad una parte del sistema, las decisiones sobre la arquitectura se hacen sobre la base de la comprensión del sistema completo y los requerimientos (funcionales y no funcionales) identificados de acuerdo al alcance definido.
- Construcción: Se obtiene un producto listo para su utilización que esté documentado y tenga un manual de usuario. Se obtienen una o varias versiones (release) del producto que han pasado las pruebas, estas versiones se disponen a la consideración de un subconjunto de usuarios.
- Transición: El release ya está listo para su instalación en las condiciones reales. Puede implicar reparación de errores.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, lo cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los objetivos de una

iteración se establecen en función de la evaluación de las iteraciones precedentes (Sánchez, 2004).

Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas: Flujos de Ingeniería y Flujos de Apoyo.

#### 1.2.1.1 Flujos de Ingeniería (Software, 2009):

- Modelamiento del negocio: Describe los procesos de negocio, identificando quiénes participan y las actividades que requieren automatización.
- Requerimientos: Define qué es lo que el sistema debe hacer, para lo cual se identifican las funcionalidades requeridas y las restricciones que se imponen.
- Análisis y diseño: Describe cómo el sistema será realizado a partir de la funcionalidad prevista y las restricciones impuestas (requerimientos), por lo que indica con precisión lo que se debe programar.
- Implementación: Define cómo se organizan las clases y objetos en componentes, cuáles nodos se utilizarán y la ubicación en ellos de los componentes y la estructura de capas de la aplicación.
- Prueba (Testeo): Busca los defectos a los largo del ciclo de vida.
- Instalación o despliegue: Produce versiones (release) del producto y realiza actividades (empaque, instalación, asistencia a usuarios, etc.) para entregar el software a los usuarios finales.

#### 1.2.1.2 Flujos de Apoyo (Software, 2009):

- Administración del proyecto: Involucra actividades con las que se busca producir un producto que satisfaga las necesidades de los clientes.
- Administración de configuración y cambios: Describe cómo controlar los elementos producidos por todos los integrantes del equipo de proyecto en cuanto a: utilización/actualización concurrente de elementos, control de versiones, etc.
- Ambiente: Contiene actividades que describen los procesos y herramientas que soportarán el equipo de trabajo del proyecto; así como el procedimiento para implementar el proceso en una organización.

Como RUP es un proceso, en su modelación define como sus principales elementos (Computación, 2009):

- Trabajadores: Define el comportamiento y responsabilidades (rol) de un individuo, grupo de individuos, sistema automatizado o máquina, que trabajan en conjunto como un equipo. Ellos realizan las actividades y son propietarios de elementos.
- Actividades: Es una tarea que tiene un propósito claro, es realizado por un trabajador y manipula elementos.
- Artefactos: Son productos tangibles del proyecto que son producidos, modificados y usados por las actividades.
- Flujo de actividades: Es una secuencia de actividades realizadas por trabajadores y produce un resultado de valor observable.

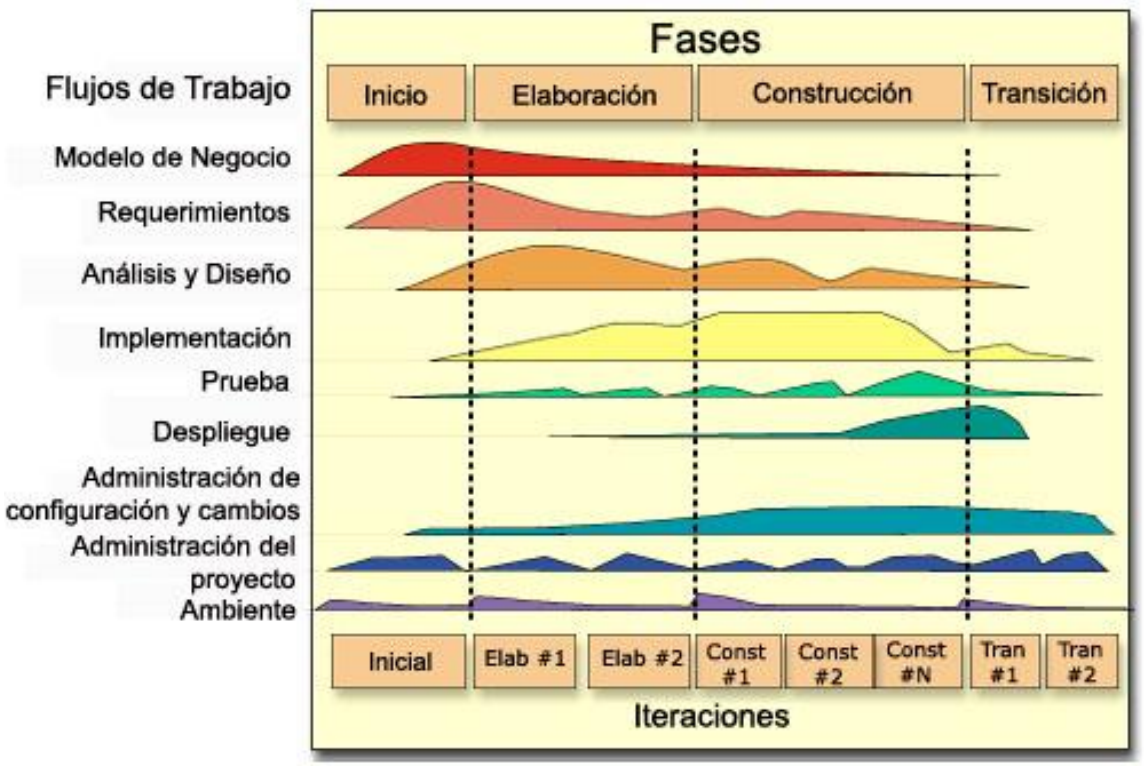



**Figura 1: Fases e Iteraciones de la Metodología RUP**

### 1.2.1.3 El Ciclo de Vida de RUP se caracteriza por (Software, 2009):

- Dirigido por casos de uso: Los casos de uso reflejan lo que los usuarios futuros necesitan y desean, lo cual se capta cuando se modela el negocio y se representa a través de los requerimientos. A partir de aquí los casos de uso guían el proceso de desarrollo ya que los modelos que se obtienen, como resultado de los

diferentes flujos de trabajo, representan la realización de los casos de uso (cómo se llevan a cabo).

- Centrado en la arquitectura: La arquitectura muestra la visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. RUP se desarrolla mediante iteraciones, comenzando por los casos de uso relevantes desde el punto de vista de la arquitectura. El modelo de arquitectura se representa a través de vistas en las que se incluyen los diagramas de UML (Lenguaje Unificado de Modelado).
- Iterativo e Incremental: Puede sugerir que los flujos de trabajo se desarrollan en cascada, la "lectura" de este gráfico tiene que ser vertical y horizontal. RUP propone que cada fase se desarrolle en iteraciones. Una iteración involucra actividades de todos los flujos de trabajo, aunque desarrolla fundamentalmente algunos más que otros. Por ejemplo, una iteración de elaboración centra su atención en el análisis y diseño, aunque refina los requerimientos y obtiene un producto con un determinado nivel, pero que irá creciendo incrementalmente en cada iteración. Es práctico dividir el trabajo en partes más pequeñas o mini proyectos. Cada mini proyecto es una iteración que resulta en un incremento. Las iteraciones hacen referencia a pasos en los flujos de trabajo, y los incrementos, al crecimiento del producto. Cada iteración se realiza de forma planificada es por eso que se dice que son mini proyectos.

Una particularidad de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software (Sánchez, 2004).

### 1.2.2 Programación Extrema (XP)

La Programación Extrema (XP) o Extreme Programming es otra de las metodologías de desarrollo de software que existen en la actualidad. Mientras que RUP intenta reducir la complejidad del software por medio de estructura y la preparación de las tareas pendientes en función de los objetivos de la fase y actividad actual, XP, como toda

metodología ágil, lo intenta por medio de un trabajo orientado directamente al objetivo, basado en las relaciones interpersonales y la velocidad de reacción (Molpeceres, 2002).

XP se basa en la retroalimentación 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. Se define como: especialmente adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

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.

El ciclo de vida ideal de XP consta de seis fases: Exploración, Planificación de la Entrega (Release), Iteraciones, Producción, Mantenimiento y Muerte del Proyecto (Rodríguez Trujillo, 2008).

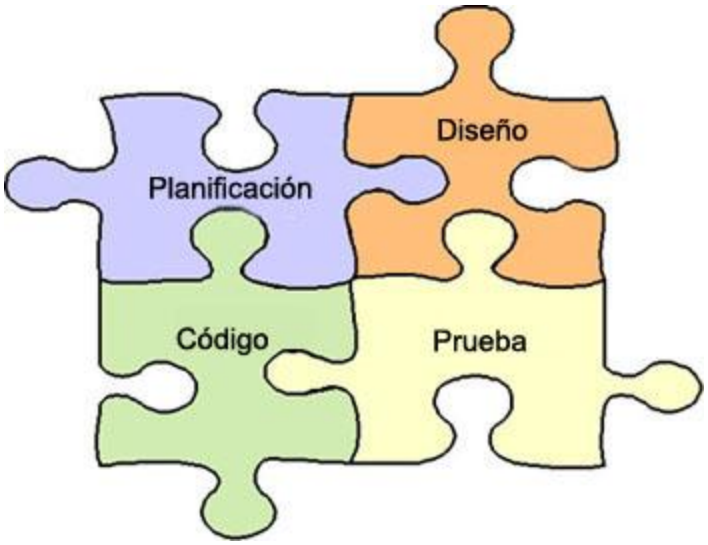



**Figura 2: Metodología XP**

#### 1.2.2.1 ¿En qué consiste XP? Sus objetivos:

Los objetivos de XP son muy simples: la satisfacción del cliente. Esta metodología trata de dar al cliente el software que él necesita y cuando lo necesita. Por tanto, debemos responder muy rápido a las necesidades del cliente, incluso cuando los cambios sean al final de ciclo de la programación.

El segundo objetivo es potenciar al máximo el trabajo en grupo. Tanto los jefes de proyecto, los clientes y desarrolladores, son parte del equipo y están involucrados en el desarrollo del software (Calero Solís, 2003).

#### 1.2.2.2 Principales Características:

La metodología se basa en:

- Pruebas Unitarias: Se basa en las pruebas realizadas a los principales procesos, de tal manera que, adelantándose en algo hacia el futuro, se puedan hacer pruebas de las fallas que pudieran ocurrir. Es como si se adelanta en la obtención de los posibles errores.
- Re-fabricación: Se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.
- Programación en pares: Una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa (Sánchez, 2004).

Lo fundamental en este tipo de metodología es:

- La comunicación, entre los usuarios y los desarrolladores
- La simplicidad, al desarrollar y codificar los módulos del sistema
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales (Sánchez, 2004).

## 1.3 Lenguaje de Modelado Unificado (UML)

Las ramas más antiguas de la ingeniería han encontrado útil, desde hace mucho tiempo, representar los diseños mediante dibujos. Desde los inicios del software, los programadores han encapsulado sus conceptos en diversos tipos de dibujos o, más ampliamente, de modelos (Jacobson, y otros, 2004).

Los modelos de ingeniería tienen como propósito ayudar a resolver un problema complejo, comunicar ideas acerca de un problema o solución y guiar la implementación.

Para que un modelo sea eficaz debe satisfacer las siguientes características:

- Abstracto: Debe enfatizar los elementos importantes y ocultar los irrelevantes.
- Comprensible: Debe ser fácil de entender para los observadores.
- Preciso: Representar de forma fiel el sistema que modela.
- Predictivo: Se puede usar para deducir conclusiones sobre el sistema que modela.
- Barato: Mucho más barato y sencillo de construir que el sistema que modela.

Cuando se dice que un modelo debe ser “Barato” no se refiere solamente a costo monetario sino también a costo de tiempo y esfuerzo por parte del equipo de trabajo.

El Lenguaje Unificado de Modelado (UML) es un lenguaje estándar de modelado para software – un lenguaje para la visualización, especificación, construcción y documentación de los artefactos de sistemas en los que el software juega un papel importante. Básicamente, UML permite a los desarrolladores visualizar los resultados de su trabajo en esquemas o diagramas estandarizados (Jacobson, y otros, 2004).

UML es un lenguaje de modelado de propósito general que pueden usar todos los modeladores. No tiene propietario y está basado en el común acuerdo de gran parte de la comunidad informática (Rumbaugh, y otros, 2000).

Es importante destacar que UML es un lenguaje de modelado, no un método o un proceso, se emplea para definir, detallar y documentar los artefactos de un sistema de software. En la última versión se adicionaron diversas novedades que resuelven carencias desde el punto de vista práctico, fundamentalmente. Entre los diagramas que propone UML para modelar un sistema se encuentran (Rumbaugh, y otros, 2000):

- Diagramas de estructura (clases, componentes, objetos, despliegue, paquetes).

- Diagramas de comportamiento (actividades, casos de uso, estado).
- Diagramas de interacción (secuencia, comunicación).

## 1.4 Herramientas Case

CASE es una sigla, que corresponde a las iniciales de: Computer Aided Software Engineering; y en su traducción al Español significa Ingeniería de Software Asistida por Computación.

El concepto de CASE es muy amplio; y una buena definición genérica, que pueda abarcar esa amplitud de conceptos, sería la de considerar a la Ingeniería de Software Asistida por Computación (CASE), como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación. Estas herramientas permiten organizar y manejar la información de un proyecto informático. Permitiéndoles a los participantes de un proyecto, que los sistemas (especialmente los complejos), se tornen más flexibles, más comprensibles y además mejorar la comunicación entre los participantes (Périssé, 2001).

A continuación se describen los principales componentes de las herramientas CASE más utilizadas y sus funcionalidades:

### 1.4.1 Rational Rose

Rational Rose es una herramienta de diseño de software destinada a modelado visual. Proporciona un lenguaje común de modelado para el equipo que facilita la creación de software de calidad más rápidamente.

Dos rasgos importantes del Rational Rose son: su habilidad para ofrecer desarrollo iterativo e ingeniería bidireccional. El Rational Rose permite a los diseñadores aprovecharse del desarrollo iterativo (también llamado desarrollo evolutivo) porque la nueva aplicación puede ser creada por etapas con la salida de una iteración como entrada de la próxima.

Además, Rose permite la generación de código a partir de un diseño en UML en lenguajes como C++, VisualBasic, Java, Ada, genera IDL's para aplicaciones CORBA. Soporta realizar ingeniería inversa por lo que se puede obtener un diseño a partir del código de un programa. Está disponible en la plataforma Windows: en Microsoft Windows NT 4.0, Windows 95, o Windows 98; y la licencia es exclusivamente propietaria (García Montero, 2008).

### 1.4.2 Visual Paradigm

El Visual Paradigm es una suite completa de herramientas CASE que da soporte al modelado visual con UML 2.0, ofreciendo distintas perspectivas del sistema. Independiente de la plataforma y dotada de una buena cantidad de productos o módulos para facilitar el trabajo durante la confección de un software así como garantizar la calidad del producto final. También es importante destacar que tiene versiones con licencia libre para el uso de la misma (García Montero, 2008).

Posee entre sus principales características las siguientes:

- Es profesional: Brinda la posibilidad de crear un conjunto bastante amplio de artefactos utilizados con mucha frecuencia durante la confección de un Software. Todos estos, cumpliendo con el Standard UML 2.0.
- Es amigable: Puede ser utilizado en varios idiomas, sus componentes se encuentran relacionados, por lo que se hace muy fácil la creación de cualquier tipo de diagrama, ya que cada componente utilizado en el diagrama que se esté creando, sugiere nuevos posibles componentes a utilizar, por lo que ya no es necesario localizarlos en la barra donde pueden aparecer un número grande de componentes.
- Brinda un número considerable de estereotipos a utilizar, lo que permite un mayor entendimiento de los diagramas.
- Facilidades para redactar especificaciones de casos de uso: Es posible crear plantillas para las especificaciones de casos de uso y describirlos, por lo que no se necesita de una herramienta externa como editor de texto.
- Generación de código e ingeniería inversa: Brinda la posibilidad de generar código a partir de los diagramas, para plataformas como .Net, Java y PHP, así cómo obtener diagramas a partir del código.
- Integración con distintos Ambientes de Desarrollo Integrados (IDE): Se integra fácilmente con varios IDEs, entre ellos el de Visual Studio y el Eclipse.

- Interoperabilidad con otras aplicaciones: Brinda la posibilidad de intercambiar información mediante la importación y exportación de ficheros con aplicaciones como por ejemplo Visio y Rational Rose. Además permite importar y exportar XML y XMI.
- Generación de código ORM: Permite generar a partir de un Diagrama de Entidad Relación una Base de Datos Relacional y el código necesario para acceder a esta base de datos utilizando Java, PHP, C# o Enterprise Object Framework.
- Generación de documentación: Brinda la posibilidad de documentar todo el trabajo sin necesidad de utilizar herramientas externas.
- Disponibilidad en múltiples plataformas: Microsoft Windows (98, 2000, XP, o Vista), Linux, Mac OS X, Solaris o Java (García Montero, 2008).

## 1.5 Metodología y herramientas

El proyecto Alfaomega es de gran tamaño y complejo, aunque el grupo de desarrollo de simuladores es pequeño. En el mismo, se realizó un contrato con una fecha límite, donde el cliente se encuentra en otro país, por lo que no puede estar en constante contacto con el equipo de desarrolladores, sino que solo interactúa y sigue el avance del proyecto a través de reuniones que son planificadas, por ello es necesario una mayor documentación para que exista un control de todo el proceso de avance. Por tanto la metodología más apropiada según las características que han sido descritas y las particularidades del proyecto, es la tradicional. Dentro de la tradicional se centra la atención en una las más importantes metodologías de desarrollo, el Proceso Unificado de Modelado (RUP), ya que cumple con las condiciones planteadas y muestra una correcta guía para la especificación de requisitos, y aporta gran documentación favorable para el tipo de contacto que se tiene con el cliente, pues XP no se ajusta debido a que involucra al cliente desde el principio hasta el final de cada ciclo y trae consigo una escasa documentación.

Lograr una comunicación efectiva entre los usuarios y el equipo de proyecto con el objetivo de llegar a un entendimiento de lo que hay que hacer, es la clave del éxito en la producción de un software. Durante muchos años muchas aplicaciones han fallado (no se culminaron o no se usaron) porque existieron incongruencias entre lo que el usuario quería, lo que realmente necesitaba, lo que interpretaba cada miembro del equipo de proyecto y lo que realmente se obtiene. Se puede definir que RUP brinda una mejor

posibilidad de realizar una captura de requisitos que concuerde más con las expectativas del cliente y del producto (Rodríguez Trujillo, 2008).

Seleccionar una herramienta Case no es una tarea simple. No existe una herramienta mejor con respecto a otra, pues todas son vulnerables a producir fallas. Visual Paradigm es la herramienta aprobada por el proyecto Alfaomega para generar los artefactos necesarios durante el desarrollo del software. Por ello fue escogida para modelar el simulador, además de ser una herramienta multiplataforma y fácil de usar. También se utiliza el Lenguaje de Modelado Visual (UML), ya que este se integra perfectamente con la herramienta selecciona.

UML es el lenguaje de modelado de sistemas de software más conocido en la actualidad; es el estándar internacional aprobado por la OMG (Object Managment Group/Grupo de Administración de Objetos), consorcio creado en 1989 responsable de la creación, desarrollo y revisión de especificaciones para la industria del software. Constituye un grupo de especificaciones de notación orientadas a Objeto, las cuales están compuestas por distintos diagramas, que representan las diferentes etapas del desarrollo de un proyecto de software. Cuenta con varios tipos de diagramas, los cuales muestran diferentes aspectos de lo que se quiere representar (Software, 2009).

## 1.6 Conclusiones

A partir del estudio realizado sobre los principales simuladores utilizados en la enseñanza de la Física en las escuelas y de las principales metodologías y herramientas utilizadas para el desarrollo de software, se decidió utilizar como metodología de desarrollo la metodología RUP, la herramienta de modelado Visual Paradigm y UML como lenguaje de modelado, persiguiendo con esto, facilitar el desarrollo del Simulador FisMat. También se realizó un estudio de algunos de los simuladores de fenómenos físicos más utilizados actualmente, permitiendo estudiar sus características principales.

## Capítulo 2: "Descripción del Sistema"

### 2. Introducción:

En este capítulo se aborda el funcionamiento general del sistema mediante un modelo de dominio. Además, se identifican los principales requisitos funcionales y no funcionales del simulador. Se describen los actores y los casos de uso del sistema, así como sus relaciones, siendo esto reflejado en el diagrama de casos de uso para mejorar su comprensión.

### 2.1 Modelo de negocio

De acuerdo a RUP, un modelo representa una forma de contemplar el sistema que se modela, según el punto de vista que se elabora. El modelado del negocio es una técnica para comprender los procesos de negocio de la organización (Jacobson, y otros, 2004).

Los objetivos del modelado de negocio son:

- Comprender la estructura y la dinámica de la organización en la cual se va a implantar un sistema.
- Comprender los problemas actuales de la organización e identificar las mejoras potenciales.
- Asegurar que los consumidores, usuarios finales y desarrolladores tengan un entendimiento común de la organización.
- Derivar los requerimientos del sistema que va a soportar la organización.

Si los procesos están claramente definidos y no se van a introducir cambios entonces se justifica la realización de un modelo de negocio. Si se determina que no es necesario un modelo completo del negocio se realizará lo que se conoce como modelo de dominio.

### 2.2 Modelo de Dominio

El Modelo de Dominio (Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema, conceptos del mundo real, no

de los componentes de software. En este modelo no se muestra comportamiento. Las clases conceptuales pueden tener atributos pero no métodos (Pressman, 2002).

El modelo del dominio se describe mediante diagramas UML, específicamente mediante diagramas de clases conceptuales significativas en el dominio del problema, donde se muestran a los clientes, usuarios, revisores y a otros desarrolladores las clases del dominio y cómo se relacionan unas con otras a través de asociaciones (Jacobson, y otros, 2004).

Su principal objetivo es ayudar a los implicados en el desarrollo del producto a utilizar un vocabulario común que posibilite una mejor comprensión entre ellos, para poder entender y describir las clases más importantes dentro del contexto donde se ubica el sistema. Además, permite y facilita el levantamiento de requisitos así como definir los procesos y roles más significativos.

### 2.2.1 ¿Por qué usar modelo de dominio?

El primer paso del modelado del negocio consiste en capturar los procesos de negocio de la organización bajo estudio. La definición del conjunto de procesos del negocio es una tarea crucial, ya que define los límites del proceso de modelado posterior. Teniendo en cuenta lo anteriormente dicho no es posible realizar un modelo de negocio en la presente investigación pues no existe ningún límite en el negocio que se desarrolla. Esto, unido al bajo nivel de estructuración que presenta los procesos y a que no se definen concretamente los procesos del negocio hace posible que se proponga un modelo de dominio, ayudando a los usuarios, clientes, desarrolladores y demás interesados, a utilizar un vocabulario común para poder entender el contexto en que se emplaza el sistema. Con este se logrará un glosario o diccionario de clases que sirve como común denominador a todos los componentes del sistema, incluyendo a las personas involucradas a lo largo del desarrollo. El modelo de dominio puede ser tomado como el punto de partida para el diseño del sistema pues, al programar orientado a objetos, como es el caso, el mapa de conceptos de este modelo constituye una primera versión del sistema imitando así en alguna medida la realidad y cuya función principal es ayudar a comprender el problema a tratar.

### 2.2.2 Conceptos del dominio

Usuario: Persona con conocimientos de Física y Matemática que desea simular un modelo, puede ser un estudiante o un profesor.

Simulación: Representa un modelo de forma virtual obteniendo resultados de su posible funcionamiento en la realidad.

Hoja Lógica: Es una ventana en la que se crea, abre, guarda y compila un modelo matemático.

Modelo Matemático: Tipo de modelo científico, que emplea algún tipo de formulismo matemático para expresar relaciones, proposiciones sustantivas de hechos, variables, parámetros, entidades y relaciones entre variables y/o entidades u operaciones, para estudiar comportamientos de sistemas complejos ante situaciones difíciles de observar en la realidad.

Animación: Área en que se muestran los componentes y herramientas de la simulación.

Administrador de componentes: Es una ventana que tiene un listado de todas las herramientas y componentes de la simulación, así como las gráficas. Permite modificar las propiedades de estos elementos y eliminarlos.

Componentes: Son etiquetas, imágenes, puntos, vectores o medidores.

Herramientas: Están constituidas por los cursores.

Gráfica: Representación visual que muestra el comportamiento en el eje X y Y de las variables definidas en el modelo matemático.

Vector: Representación geométrica de una magnitud (velocidad, aceleración, fuerza) que necesita orientación espacial, punto de aplicación, dirección y sentido para quedar definida: los vectores no son magnitudes escalares.

Medidor: Mide el valor de una variable durante la simulación.

Etiqueta: Marca o señal que se coloca para identificar o representar el valor de una variable durante la simulación.

Imagen: Figura que representa un comportamiento durante la simulación.

Punto: Lugar de una recta, superficie o espacio al que se puede asignar una posición pero que no posee dimensiones.

Cursor: Mide el valor de una variable durante la simulación.

### 2.2.3 Diagrama del modelo de dominio

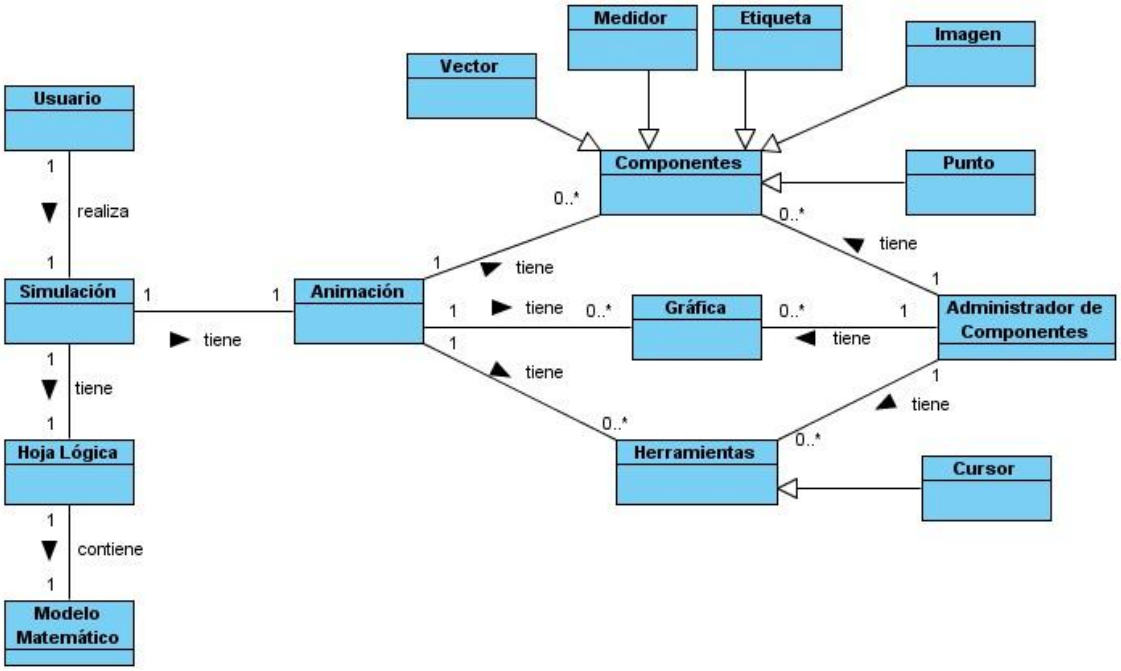



**Figura 3: Modelo de dominio**

### 2.2.4 Descripción del modelo de dominio

El usuario se encarga de realizar una o más simulaciones de su interés. Para la realización de las mismas debe contar con un modelo matemático que represente las operaciones que se desean simular. Para facilitar el proceso de simulación puede apoyarse en el uso de algunos componentes como: vectores, medidores, etiquetas, imágenes y puntos; de herramientas como: cursores; y gráficas que muestren el comportamiento en los ejes X y Y de las variables definidas en el modelo.

## 2.3 Requisitos del sistema

Un requisito (o requerimiento) es una característica de diseño, una propiedad o un comportamiento de un sistema. Los requisitos constituyen la descripción de los deseos o de las necesidades de un producto (Jacobson, y otros, 2004).

### 2.3.1 Requisitos funcionales

Los requisitos funcionales son capacidades o condiciones que el sistema debe cumplir. Estos se mantienen invariables sin importar con que propiedades o cualidades se relacionen (Software, 2009).

RF1 Gestionar Modelo Matemático

RF1.1 Definir un modelo matemático, que describa un fenómeno físico.

RF1.2 Guardar un modelo matemático.

RF1.3 Abrir un modelo matemático previamente definido.

RF1.4 Definir los casos (Establecer entre 1 y 3 valores a las variables del modelo matemático descrito).

RF2 Compilar Modelo Matemático

RF2.1 Interpretar un modelo matemático.

RF2.2 Mostrar los errores que contenga el modelo matemático definido.

RF3 Consultar Ayuda del Intérprete

RF3.1 Mostrar la ayuda del intérprete con la sintaxis y las principales reglas que debe cumplir el modelo matemático definido.

RF3.2 Mostrar listado de palabras reservadas agrupadas por las siguientes categorías: Palabras Reservadas, Funciones Matemáticas, Operadores Aritméticos y de Comparación y otras.

RF4 Gestionar Objeto

RF4.1 Adicionar/Eliminar objetos en el escenario.

RF4.2 Seleccionar un objeto.

RF4.3 Mostrar las propiedades del objeto.

RF4.4 Definir el comportamiento de cada objeto según las variables del modelo matemático definido en el intérprete.

RF4.4.1 Cambiar su posición en el eje X y/o en el eje Y.

RF4.4.2 Modificar sus dimensiones.

RF4.4.3 Modificar su visibilidad.

RF4.5 Mover un objeto previamente adicionado a cualquier posición dentro del escenario.

RF5 Realizar Simulación

RF5.1 Iniciar la simulación.

RF5.2 Pausar la simulación.

RF5.3 Detener la simulación.

RF5.4 Mostrar el comportamiento de los objetos según el modelo matemático descrito.

RF5.5 Permitir la ejecución de 1 a 3 casos.

RF5.5.1 Cambiar entre cualquiera de los casos definidos.

RF5.5.2 Mostrar el comportamiento de los objetos según el caso definido en el modelo matemático descrito.

RF6 Administrar Propiedades de los Componentes y Herramientas

RF6.1 Listar todos los objetos ubicados en el escenario.

RF6.2 Seleccionar un objeto en el Administrador de Componentes.

RF6.3 Mostrar las propiedades de un objeto seleccionado.

RF6.4 Modificar las propiedades de un objeto seleccionado.

RF6.5 Eliminar un objeto seleccionado.

RF6.6 Mostrar la posición (x; y) en el escenario de un objeto seleccionado.

RF7 Adicionar Elemento al Escenario

RF7.1 Permitir adicionar al escenario cualquiera de los siguientes elementos: Cursores, Etiquetas, Puntos, Vectores y Medidores.

RF7.2 Medir, durante la simulación, el valor de las variables definidas en el modelo matemático.

RF7.3 Mostrar de manera puntual o vectorial el comportamiento de las variables definidas.

RF8 Graficar

RF8.1 Graficar el comportamiento de una variable durante la simulación.

RF9 Administrar Elementos del Simulador

RF9.1 Abrir una simulación previamente guardada.

RF9.2 Guardar una simulación creada.

RF9.3 Mostrar/Ocultar el administrador de objetos.

RF9.4 Mostrar la ayuda general del software.

### 2.3.2 Requisitos no funcionales

Los requisitos no funcionales son propiedades o cualidades que el producto debe tener. Son importantes para que clientes y usuarios puedan valorar las características no funcionales del producto (Software, 2009).

- **Requisitos de apariencia o interfaz externa**

- Interfaz sencilla para usuarios finales. Su funcionamiento deberá ser intuitivo, y requerir de información mínima.
- Claridad y buena organización de la información, permitiendo la interpretación correcta e inequívoca de la información.
- Ejecución de acciones de una manera rápida, minimizando los pasos a dar en cada proceso.

- **Requisitos de Seguridad**

- Debe proteger el contenido de los productos educativos que gestione, de forma que restrinja el acceso a usuarios no autorizados.
- Seguridad de acceso y administración de usuarios: otorgamiento de privilegios y roles, asignación de perfiles. Los niveles de acceso están determinados por los diferentes roles válidos dentro de la misma.
- Protección contra la copia ilegal, mediante un procedimiento que impida que funcione en otra computadora que no esté registrada en Alfaomega.

## 2.4 Módulos del sistema

Un módulo de un simulador agrupa e implementa las funcionalidades de los casos de uso relacionados con la misma área temática o el mismo tipo de información. El simulador FisMat cuenta con 4 módulos, descritos a continuación:

**Módulo Intérprete**: Maneja la información relacionada con la Hoja Lógica y el modelo matemático, también se hace cargo del proceso de compilación de dicho modelo. Está constituido por los siguientes casos de uso:

- CU Gestionar Modelo Matemático.
- CU Compilar Modelo Matemático.
- CU Consultar Ayuda del Intérprete.

**Módulo Escenario**: Permite adicionar los componentes y herramientas al escenario de trabajo, además de encargarse del proceso de simulación. Lo conforman los siguientes casos de uso:

- CU Gestionar Objeto.
- CU Realizar Simulación.
- CU Administrar Propiedades de los Componentes y Herramientas.

**Módulo Instrumentos**: Permite adicionar algunos componentes al escenario de trabajo y cubre el proceso de graficar el comportamiento de las variables definidas en el modelo. Está formado por los siguientes casos de uso:

- CU Adicionar Elemento al Escenario.
- CU Graficar.

**Módulo Elementos Generales**: Se encarga de la creación, y cambios que se realizan en las simulaciones. Integrado por el siguiente caso de uso:

- CU Administrar Elementos del Simulador.

## 2.5 Definición y descripción de los casos de uso del sistema

### 2.5.1 Definición de los actores del sistema

El término actor significa el rol que algo o alguien juega cuando interactúa con el sistema. Un candidato a actor del sistema es cualquier individuo, grupo, organización o máquina que interactúa en los casos de uso. De acuerdo con esta idea un actor del sistema representa un tipo particular de usuario del sistema más que un usuario físico, ya que varios usuarios físicos pueden realizar el mismo papel en relación al negocio, o sea, ser instancias de un mismo actor. Una vez que se han identificado los actores del sistema, se tiene identificado el entorno externo del sistema.

| Actor | Descripción |
|---|---|
| Usuario | Persona con conocimientos de Física y Matemática que desea simular un modelo, pueden ser un estudiante o un profesor. |

**Tabla 1: Definición de los actores del sistema**

### 2.5.2 Diagrama de casos de uso del sistema

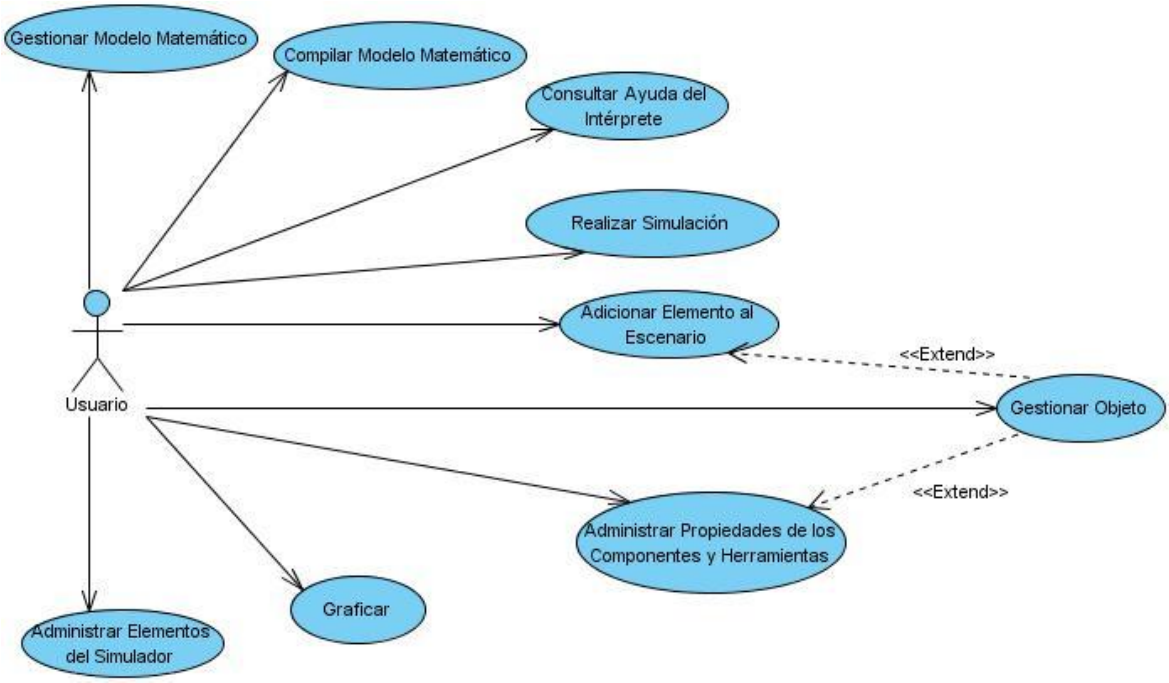



**Figura 4: Diagrama de casos de uso del sistema**

### 2.5.3 Casos de Uso del Sistema

A continuación se presenta para cada caso de uso del sistema, las especificaciones de manera resumida de los eventos más importantes, para mayor comprensión remitirse al expediente del proyecto Alfaomega.

#### 2.5.3.1 Gestionar Modelo Matemático

| Nombre | CU Gestionar Modelo Matemático |
|---|---|
| Objetivo | Crear, guardar y abrir un modelo matemático. |
| Actores | Usuario (Inicia): Tramitar Modelo Matemático. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>El estudiante debe tener los conocimientos necesarios para crear un modelo matemático. |
| Poscondiciones | Se creó, abrió o guardó el Modelo Matemático. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el Usuario realiza una acción sobre la Hoja Lógica. El actor puede crear, abrir y guardar un Modelo Matemático. En caso de que seleccione la opción de crear un Modelo Matemático, el sistema dará la posibilidad de insertar los datos que se necesitan para confeccionar dicho modelo. Si el actor elige la opción de abrir un Modelo Matemático el sistema permitirá seleccionar un modelo previamente guardado. Si el actor elige la opción de guardar un Modelo Matemático, el sistema permitirá escoger el lugar donde guardar el modelo en cuestión. El caso de uso termina. | |

**Tabla 2: Descripción del CU Gestionar Modelo Matemático**

#### 2.5.3.2 Compilar Modelo Matemático

| Nombre | CU Compilar Modelo Matemático |
|---|---|

| | |
|---|---|
| Objetivo | Interpretar un Modelo Matemático y mostrar los errores que contenga dicho modelo. |
| Actores | Usuario (Inicia): Compilar Modelo Matemático. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>El Usuario ha creado o abierto un Modelo Matemático. |
| Poscondiciones | Se interpretó el Modelo Matemático.<br>Se mostró un listado de los errores existentes en el Modelo Matemático, en caso de que existieran. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el Usuario selecciona la opción que le permite interpretar el Modelo Matemático, en la Hoja Lógica. El sistema interpreta el modelo Matemático y se muestra un mensaje que indica el resultado de la interpretación, este mensaje puede indicar los errores, o el estado satisfactorio del proceso. El caso de uso termina. | |

**Tabla 3: Descripción del CU Compilar Modelo Matemático**

#### 2.5.3.3 Consultar Ayuda del Intérprete

| | |
|---|---|
| Nombre | CU Consultar Ayuda del Intérprete |
| Objetivo | Consultar la Ayuda del Intérprete. |
| Actores | Usuario (Inicia): Consultar la Ayuda del Intérprete. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado. |
| Poscondiciones | Se mostró la Ayuda del Simulador al Usuario. |
| Descripción del Caso de Uso | |

| |
|---|
| El caso de uso se inicia cuando el Usuario realiza una acción sobre Hoja Lógica que le permite abrir la ayuda del intérprete. El actor puede seleccionar el contenido sobre el que desea obtener información. La ayuda también contiene la sintaxis y las principales reglas que debe cumplir el modelo matemático definido. Además, muestra un listado de palabras reservadas agrupadas por las siguientes categorías: Palabras Reservadas, Funciones Matemáticas, Operadores Aritméticos y de Comparación y otras. El caso de uso termina. |

**Tabla 4: Descripción del CU Consultar Ayuda del Intérprete**

#### 2.5.3.4 Gestionar Objeto

| | |
|---|---|
| Nombre | CU Gestionar Objeto |
| Objetivo | Incluir, seleccionar, modificar o eliminar un Objeto. |
| Actores | Usuario (Inicia): Incluye, selecciona, modifica o elimina objetos. |
| Complejidad | Media. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Para incluir un Objeto debe estar seleccionado previamente.<br>Para ver las propiedades del Objeto, debe estar seleccionado previamente.<br>Para modificar el Objeto, debe estar seleccionado con anterioridad.<br>Para eliminar el Objeto, debe estar seleccionado.<br>Debe haberse realizado la compilación de manera correcta. |
| Poscondiciones | Se incluyó, seleccionó, modificó o eliminó un Objeto por el actor. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el Usuario selecciona la opción que le permite realizar una acción sobre el Objeto. El actor puede incluir, ver, modificar y eliminar un Objeto. En | |

| |
|---|
| caso de que seleccione la opción de incluir un Objeto, el sistema dará la posibilidad de insertar los datos que se necesitan para definir el comportamiento de cada objeto según las variables del modelo matemático definido en el intérprete. Si el actor elige la opción de seleccionar un Objeto, el sistema mostrará las Propiedades del Objeto en cuestión. Si el actor elige la opción de modificar un Objeto, el sistema mostrará los datos que pueden ser editables dentro de un Objeto. Si el actor elige la opción de eliminar un Objeto, el sistema eliminará el Objeto seleccionado, y una vez realizados los cambios, guardará las modificaciones. El caso de uso termina. |

**Tabla 5: Descripción del CU Gestionar Objeto**

#### 2.5.3.5 Realizar Simulación

| | |
|---|---|
| Nombre | CU Realizar Simulación |
| Objetivo | Mostrar el comportamiento de los Objetos durante la simulación. |
| Actores | Usuario (Inicia): Realizar Simulación. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Debe haberse realizado la compilación de manera correcta. |
| Poscondiciones | Se inició, pausó o detuvo la simulación. |
| Descripción del Caso de Uso | |
| El caso de uso inicia cuando el actor selecciona la opción que le permite realizar un la simulación, el actor puede pausar o detener la simulación. El caso de uso termina. | |

**Tabla 6: Descripción del CU Realizar Simulación**

#### 2.5.3.6 Administrar Propiedades de los Componentes y Herramientas

| | |
|---|---|
| Nombre | CU Administrar Propiedades de los Componentes y Herramientas |
| Objetivo | Seleccionar, eliminar o ver las propiedades de un componente o herramienta. |

| | |
|---|---|
| Actores | Usuario (Inicia): Administrara las Propiedades de los Componentes y Herramientas. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Debe haberse realizado la compilación de manera correcta.<br>Debe haberse añadido un componente o herramienta al escenario de trabajo. |
| Poscondiciones | Se seleccionó o eliminó un objeto o se vieron las propiedades del mismo. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el actor realiza una acción sobre un Objeto, estos se encuentran ubicados en el administrador de componentes desde el mismo momento en que son adicionados al escenario de trabajo. El actor puede ver las propiedades de un Objeto seleccionado, así como eliminar el mismo. El caso de uso termina. | |

**Tabla 7: Descripción del CU Administrar Propiedades de los Componentes y Herramientas**

#### 2.5.3.7 Adicionar Elemento al Escenario

| | |
|---|---|
| Nombre | CU Adicionar Elemento al Escenario |
| Objetivo | Incluir al escenario cualquiera de los siguientes elementos: Cursores, Etiquetas, Puntos y Vectores. |
| Actores | Usuario (Inicia): Adicionar elemento. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Debe haberse realizado la compilación sin errores. |
| Poscondiciones | Se adicionó un elemento al escenario. |

| | |
|---|---|
| | Se mostró de forma puntual o vectorial el comportamiento de una variable.<br>Se midió el valor de una variable. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el actor realiza una acción que le permita incluir un elemento, dichos elementos pueden ser: Cursores, Etiquetas, Puntos y Vectores. Además, el actor puede medir el valor de una variable definida en una etiqueta o cursor, así como, observar el comportamiento de la misma de forma puntual o vectorial. El caso de uso termina. | |

**Tabla 8: Descripción del CU Adicionar Elemento al Escenario**

#### 2.5.3.8 Graficar

| | |
|---|---|
| Nombre | CU Graficar |
| Objetivo | Graficar el comportamiento de una variable durante la simulación. |
| Actores | Usuario (Inicia): Graficar. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Debe haberse realizado la compilación de manera correcta. |
| Poscondiciones | Se graficó el comportamiento de una variable durante la simulación. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el Usuario selecciona la opción de graficar. Puede introducir el nombre de la gráfica y seleccionar el comportamiento de las variables en los ejes de coordenadas X y Y. El caso de uso termina. | |

**Tabla 9: Descripción del CU Graficar**

#### 2.5.3.9 Administrar Elementos del Simulador

| | |
|---|---|
| Nombre | CU Administrar Elementos del Simulador |

| | |
|---|---|
| Objetivo | Abrir una simulación.<br>Guardar una simulación.<br>Mostrar u ocultar los diferentes elementos que componen el simulador.<br>Mostrar la ayuda. |
| Actores | Usuario (Inicia): Administrar Elementos del Simulador. |
| Complejidad | Baja. |
| Nivel | Usuario. |
| Precondiciones | Debe haberse generado el escritorio de trabajo del usuario autenticado.<br>Debe haberse guardado una simulación con anterioridad en caso que se desee abrir una.<br>Debe haberse creado una simulación para poder guardarla. |
| Poscondiciones | Se abrió una simulación.<br>Se guardó una simulación.<br>Se mostró la ayuda. |
| Descripción del Caso de Uso | |
| El caso de uso se inicia cuando el Usuario decide abrir una simulación previamente guardada o guardar una creada, así como crear una nueva. También puede mostrar la ayuda general del software y mostrar u ocultar los diferentes elementos que componen el simulador. El caso de uso termina. | |

**Tabla 10: Descripción del CU Administrar Elementos del Simulador**

### 2.5.4 Diccionario de Datos

Variable de control:

- Variable de control: Formato selección.
- Retardo: Formato numérico.

Variables iniciales de la imagen:

- Posición x: Formato numérico.
- Posición y: Formato numérico.

- Alto: Formato numérico.
- Ancho: Formato numérico.

Variables iniciales del cursor:

- Posición x: Formato numérico.
- Posición y: Formato numérico.
- Mínimo: Formato numérico.
- Máximo: Formato numérico.

Variables iniciales de la etiqueta:

- Posición x: Formato numérico.
- Posición y: Formato numérico.

Animación de la imagen:

- Posición x: Formato numérico.
- Posición y: Formato numérico.
- Ancho: Formato numérico.
- Alto: Formato numérico.
- Desplazamiento x: Formato selección (Derecha o Izquierda).
- Desplazamiento y: Formato selección (Superior o Inferior).

Visibilidad de la imagen:

- Siempre visible: Formato selección.

Animación del cursor:

- Variable: Formato selección.

Orientación del cursor:

- Horizontal: Formato selección.
- Vertical: Formato selección.

Etiqueta de texto:

- Texto: Formato texto.
- Dinámica: Formato selección.

Etiqueta dinámica:

- Variable: Formato selección.
- Dinámica: Formato selección.

Variables iniciales del vector:

- Posición x: Formato numérico.
- Posición y: Formato numérico.
- Componente x: Formato numérico.
- Componente y: Formato numérico.

Variables iniciales del punto:

- Posición x: Formato numérico.
- Posición y: Formato numérico.
- Ancho: Formato numérico.

Variables iniciales del medidor:

- Posición x: Formato numérico.
- Posición y: Formato numérico.
- Cantidad de unidades: Formato numérico.
- Unidad de medida: Formato texto.

Animación del vector:

- Posición x: Formato selección.
- Posición y: Formato selección.
- Componente x: Formato selección.
- Componente y: Formato selección.

Componentes X y Y: Formato selección.

Animación del punto:

- Posición x: Formato selección.
- Posición y: Formato selección.

Marcar Trayectoria: Formato selección.

Animación del medidor:

- Variable: Formato selección.

## 2.6 Patrones de Casos de Uso

Los patrones de casos de uso son comportamientos que deben existir en el sistema, ayudan a describir qué es lo que el sistema debe hacer, es decir, describen el uso del sistema y cómo este interactúa con los usuarios. Estos patrones son utilizados generalmente como plantillas que describen como debería ser estructurados y organizados los casos de uso (Larman, 1999).

### 2.6.1 Patrones de Casos de Uso utilizados

#### 2.6.1.1 Patrón Concordancia (Commonality) (Software, 2009)

Extrae una subsecuencia de acciones que aparecen en diferentes lugares del flujo de casos de uso y es expresado por separado.

**Adición**: Consta de 3 casos de uso. La subsecuencia común de casos de uso, extiende los casos de uso compartiendo la subsecuencia de acciones. Los otros casos de uso modelan el flujo que será expandido con la subsecuencia. O sea, el caso de uso Gestionar Objeto, posee funcionalidades que pueden ser utilizadas por los casos de uso: Adicionar Elemento al Escenario y Administrar Propiedades de los Componentes y Herramientas. A continuación se muestra un ejemplo de su uso en el diagrama:

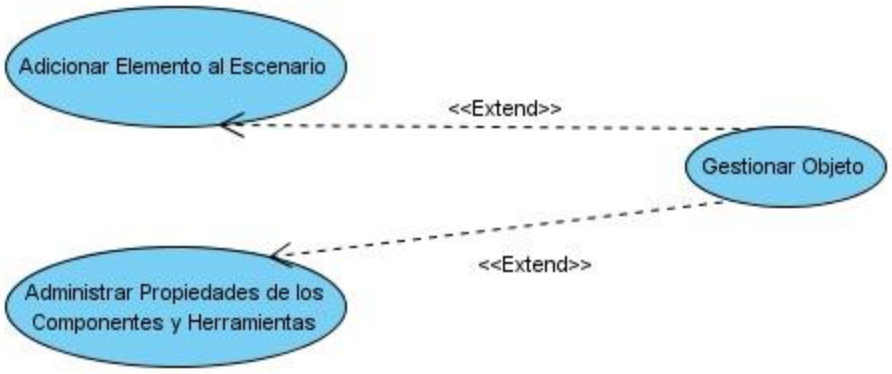



**Figura 5: Patrón Concordancia. Adición**

#### 2.6.1.2 Patrón CRUD (Creating, Reading, Updating, Deleting) (Software, 2009)

Este patrón se basa en la fusión de casos de uso simples para formar una unidad conceptual.

**Completo**: Este patrón consta de un caso de uso, llamado Información CRUD o Gestionar información, modela todas las operaciones que pueden ser realizadas sobre una parte de la información de un tipo específico, tales como creación, lectura, actualización y eliminación. Suele ser utilizado cuando todos los flujos contribuyen al mismo valor del negocio, y estos a su vez son cortos y simples. A continuación se muestra un ejemplo de su uso en el diagrama:

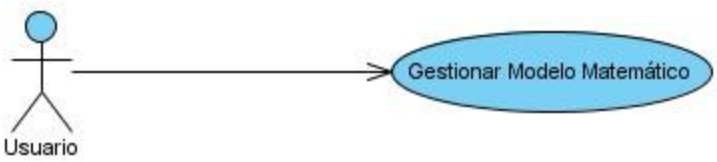



**Figura 6: Patrón CRUD. Completo**

## 2.7 Conclusiones

En este capítulo se ha definido los requerimientos que debe tener el sistema para un exitoso desarrollo, los cuales se representan mediante un Diagrama de Casos de Uso, así mismo se describieron la las acciones que debe realizar cada actor que interactúa con el sistema a través de una descripción resumida de los casos de uso. Se explicaron además, los patrones de casos de uso que se evidencian en el diagrama general de casos de uso y se abordaron los principales conceptos relacionados con el dominio del problema.

## Capítulo 3: "Análisis y Diseño del Sistema"

### 3. Introducción:

En el presente capítulo se expone el análisis y diseño realizado para la propuesta del sistema, modelándose los artefactos necesarios que contribuyen a la implementación del sistema. Estos artefactos son el Modelo de Análisis y el Modelo de Diseño, los cuales se encargan de describir en términos de clases cómo deben funcionar los casos de uso del sistema. Aquí también se logra representar la colaboración entre estas clases y se dan a conocer los patrones de diseño que se utilizan en la elaboración de las mismas.

### 3.1 Modelo de Análisis

El Modelo de Análisis es la primera representación técnica de un sistema. El Modelo de Análisis debe lograr tres objetivos primarios:

- Describir lo que requiere el cliente.
- Establecer una base para la creación de un diseño de software.
- Definir un conjunto de requisitos que se pueda validar una vez que se construye el software.

El propósito del análisis es definir todas las clases que son relevantes al problema que se va a resolver, las operaciones y atributos asociados, las relaciones y comportamientos asociadas con ellas (Pressman, 2002).

Este modelo es usado para representar la estructura global del sistema, describe la realización de casos de uso y sirve como una abstracción del Modelo de Diseño. El Modelo de Análisis puede contener: las clases y paquetes de análisis, las realizaciones de los casos de uso, las relaciones y los diagramas.

"Durante el análisis, analizamos los requisitos que se describen en la captura de requerimientos, refinándolos y estructurándolos. El objetivo de hacerlo es conseguir una comprensión más precisa de los requisitos y una descripción de los mismos que sea fácil de mantener y que nos ayude a estructurar el sistema entero, incluyendo su arquitectura" (Jacobson, 1999).

### 3.1.1 Diagramas de Clases del Análisis

El objetivo principal del flujo de trabajo del análisis son los diagramas de clases de análisis, los cuales muestran qué clases participan en las realizaciones de los distintos casos de usos.

Un Diagrama de clases del análisis es un artefacto en el que se representan los conceptos en un dominio del problema. Representa las cosas del mundo real, no de la implementación automatizada de estas cosas. Incorpora, además, las definiciones y relaciones entre las clases. Las clases del se clasifican en:

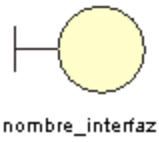
| Nombre | Características | Representación |
|---|---|---|
| Entidad | Modelan información que posee larga vida y que es a menudo persistente. |  |
| Interfaz | Modelan la interacción entre el sistema y sus actores. |  |
| Control | Coordinan la realización de uno o unos pocos casos de uso coordinando las actividades de los objetos que implementan la funcionalidad del caso de uso. |  |

**Tabla 11: Descripción de las Clases del Análisis**

A continuación se muestra los diagramas de clases de análisis para el Módulo Instrumentos del Simulador FisMat.

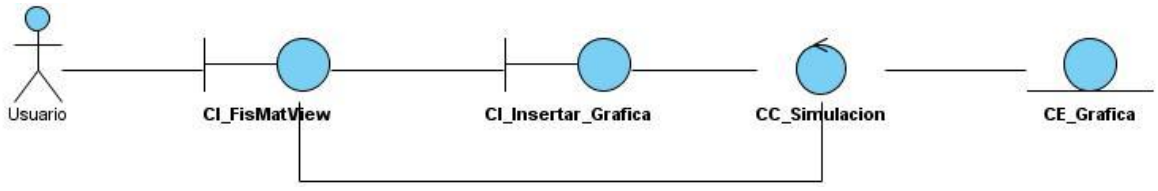


**Figura 8: Diagrama de Clases del Análisis del CU Graficar**

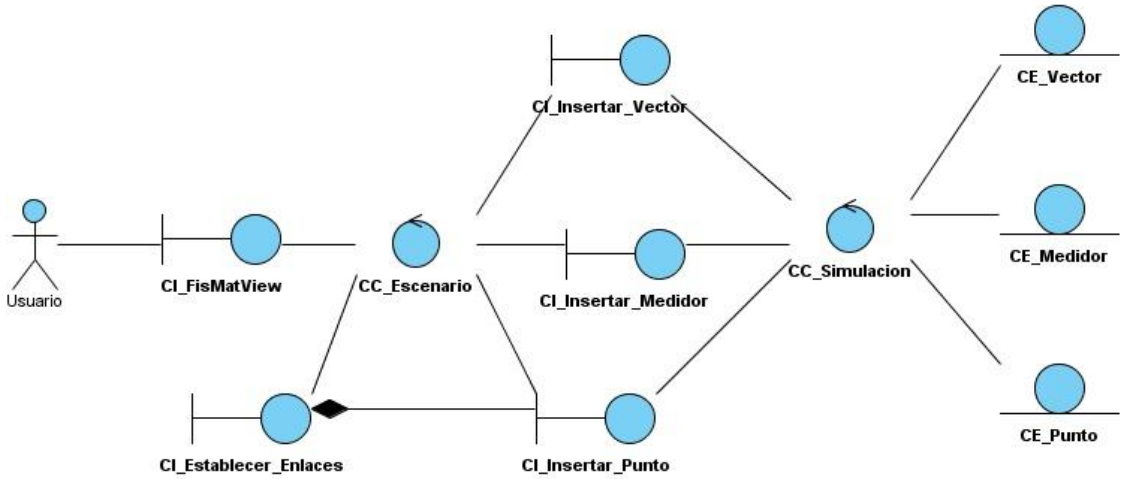



**Figura 9: Diagrama de Clases del Análisis del CU Adicionar Elemento al Escenario**

### 3.1.2 Diagramas de Interacción

Los diagramas de interacción se utilizan para modelar los aspectos dinámicos de un sistema, lo que conlleva modelar instancias concretas o prototípicas de clases interfaces, componentes y nodos, junto con los mensajes enviados entre ellos, todo en el contexto de un escenario que ilustra un comportamiento. En el contexto de las clases describen la forma en que grupos de objetos colaboran para proveer un comportamiento.

Se dividen en dos categorías los diagramas de colaboración y los diagramas de secuencia.

Un diagrama de secuencia es un diagrama de interacción que destaca la ordenación temporal de los mensajes. Muestran las interacciones expresadas en función de secuencias temporales.

Un diagrama de colaboración es un diagrama de interacción que destaca la organización estructural de los objetos que envían y reciben mensajes. Muestra las relaciones entre los objetos y los mensajes que intercambian.

A continuación se ilustran los diagramas de secuencia de los dos casos de uso pertenecientes al módulo Instrumentos, para consultar los diagramas de colaboración correspondientes, remitirse al Anexo 1.

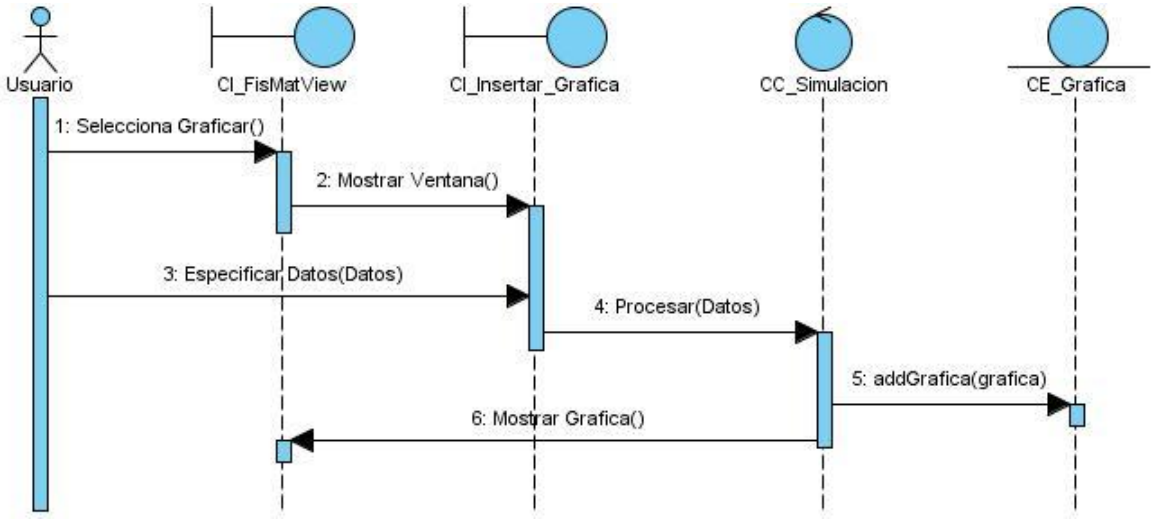



**Figura 10: Diagrama de Secuencia del CU Graficar**

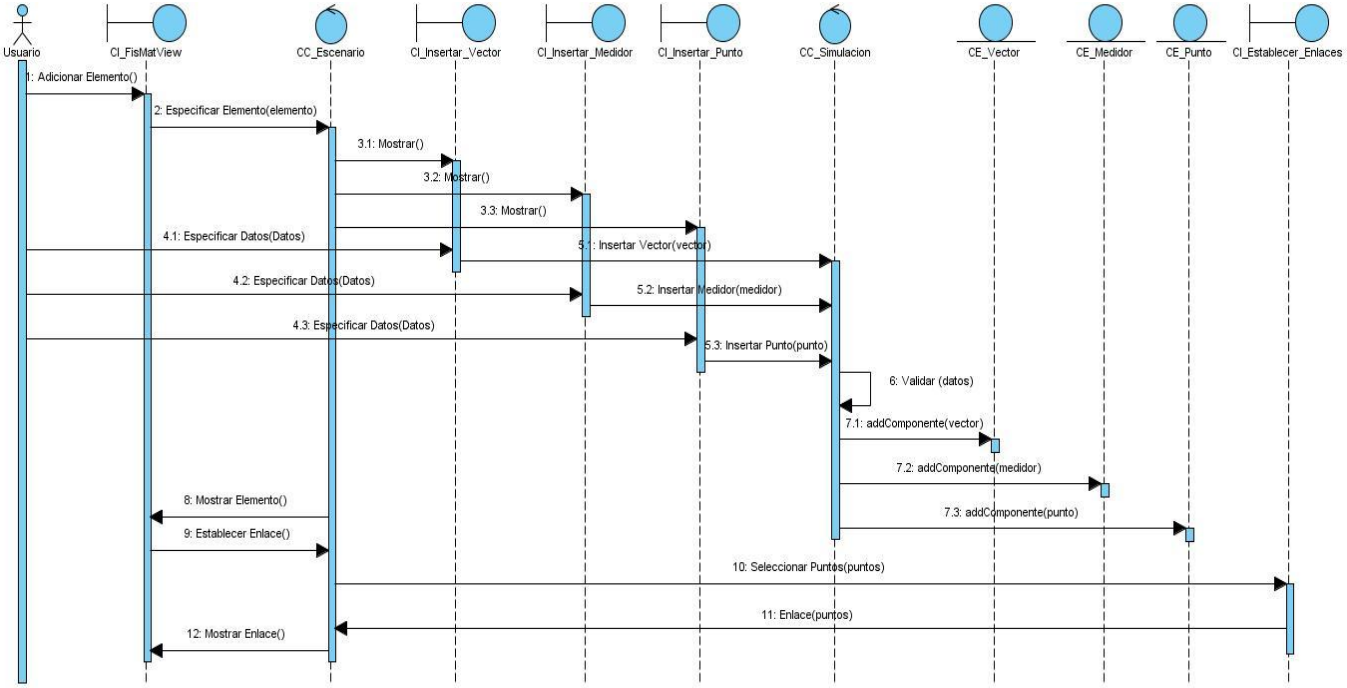



**Figura 11: Diagrama de Secuencia del CU Adicionar Elemento al Escenario**

## 3.2 Modelo de Diseño

El diseño consiste en el refinamiento de los Modelos de Análisis para crear especificaciones adicionales que enriquecen el modelo de análisis con detalles próximos a la implementación. Una solución lógica, de forma que se cumplan los requerimientos (asignación de responsabilidades, interacciones entre objetos y otros) (Carvajal Pérez, 2009).

Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación. Este modelo puede contener: los diagramas, las clases, paquetes, subsistemas, cápsulas, protocolos, interfaces, relaciones, colaboraciones, atributos, las realizaciones de los casos de uso, entre otros que se puedan considerar para el sistema en desarrollo (Carvajal Pérez, 2009).

"El Modelo de Diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además, el Modelo de Diseño sirve de abstracción de la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental de las actividades de implementación" (Jacobson, 1999).

### 3.2.1 Diagrama de Clases del Diseño

A través del flujo de diseño, uno de los artefactos más importantes a obtener son los diagramas de clases de diseño, donde se exponen las clases que intervienen en las realizaciones de los casos de uso del sistema. En este tipo de diagrama se representa un nivel de detalle más alto que los diagramas de clases del análisis, relacionándose con el lenguaje de programación del cual se hará uso en la implementación del sistema (Carvajal Pérez, 2009).

El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información (Visconti, y otros, 2003):

- Clases, asociaciones y atributos
- Interfaces, con sus operaciones y constantes

- Métodos
- Información sobre los tipos de los atributos, navegabilidad y dependencias

A continuación se muestran los diagramas de clases del diseño, correspondientes al Módulo Instrumentos del simulador. Para consultar las clases con todos sus atributos y métodos, consultar el Anexo 2.

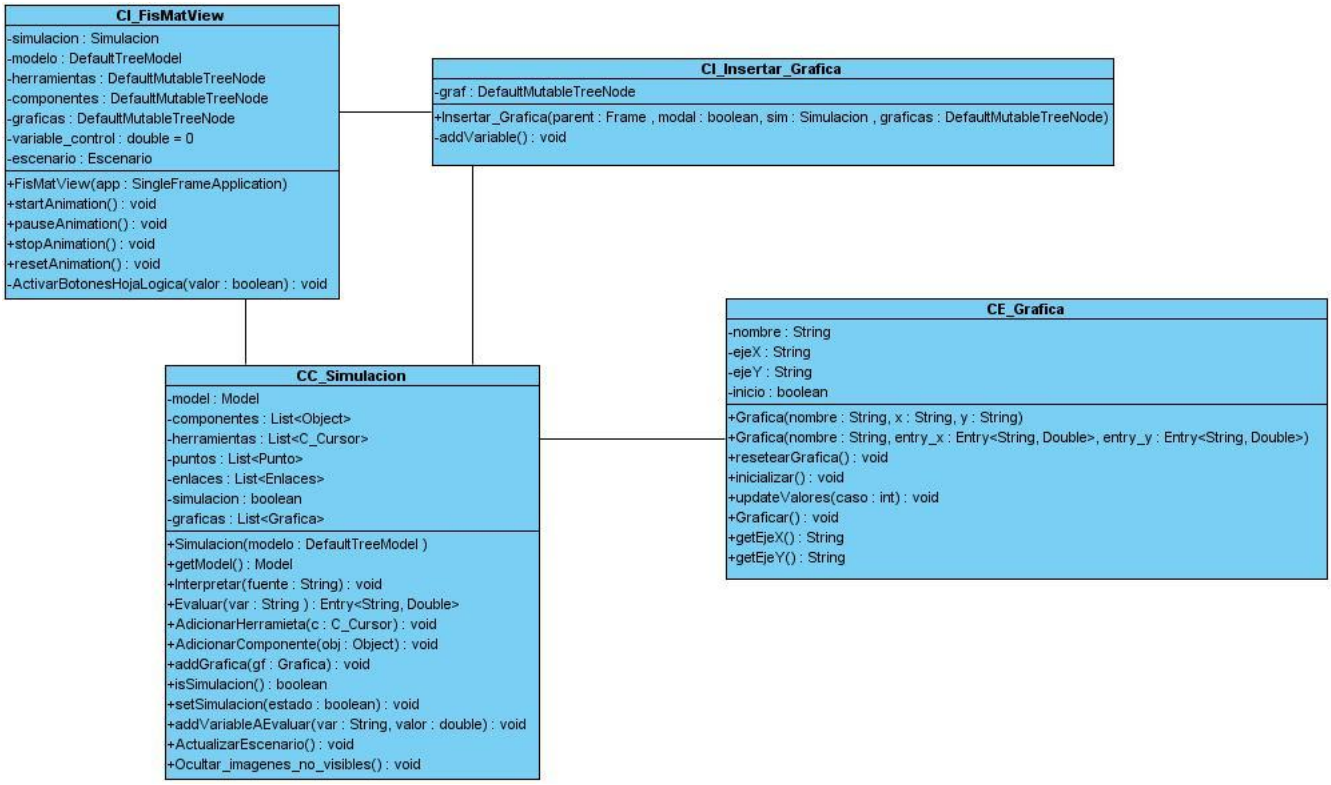



**Figura 12: Diagrama de Clases del Diseño del CU Graficar**

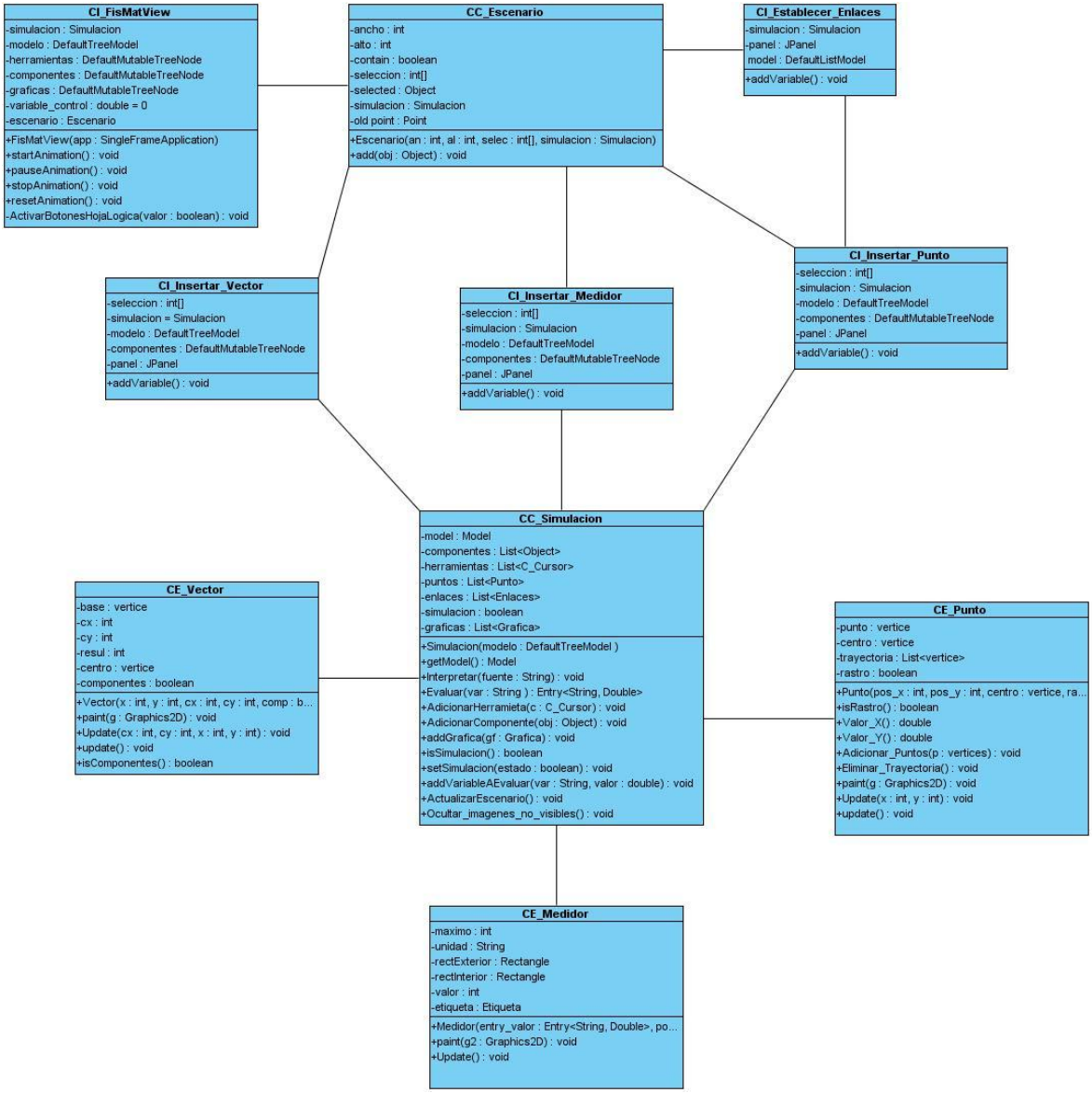



**Figura 13: Diagrama de Clases del Diseño del CU Adicionar Elemento al Escenario**

### 3.2.2 Diagramas de Interacción

En este epígrafe se muestran los diagramas de interacción del diseño correspondientes al Módulo Instrumentos del simulador, para consultar los diagramas de colaboración correspondientes, remitirse al Anexo 1.

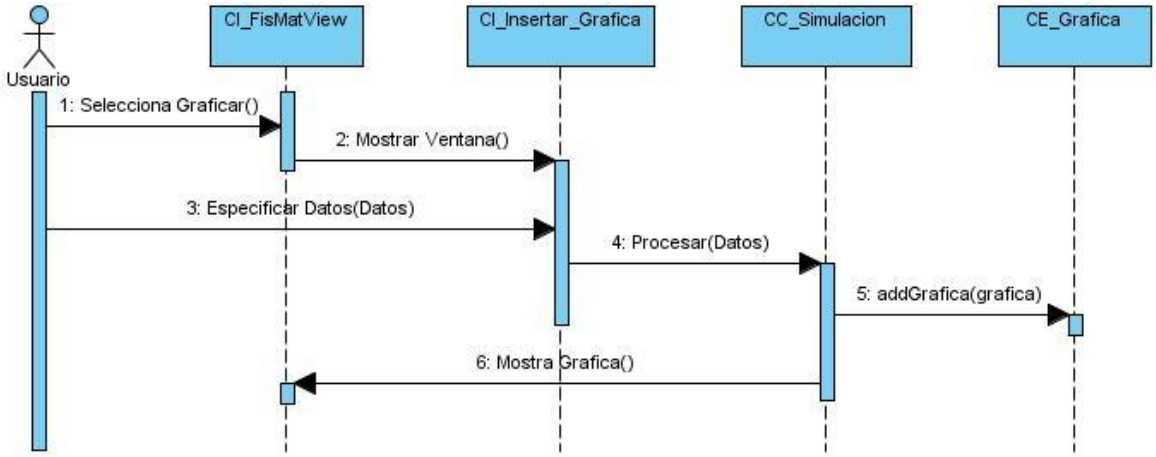



**Figura 14: Diagrama de Secuencia del Diseño del CU Graficar**

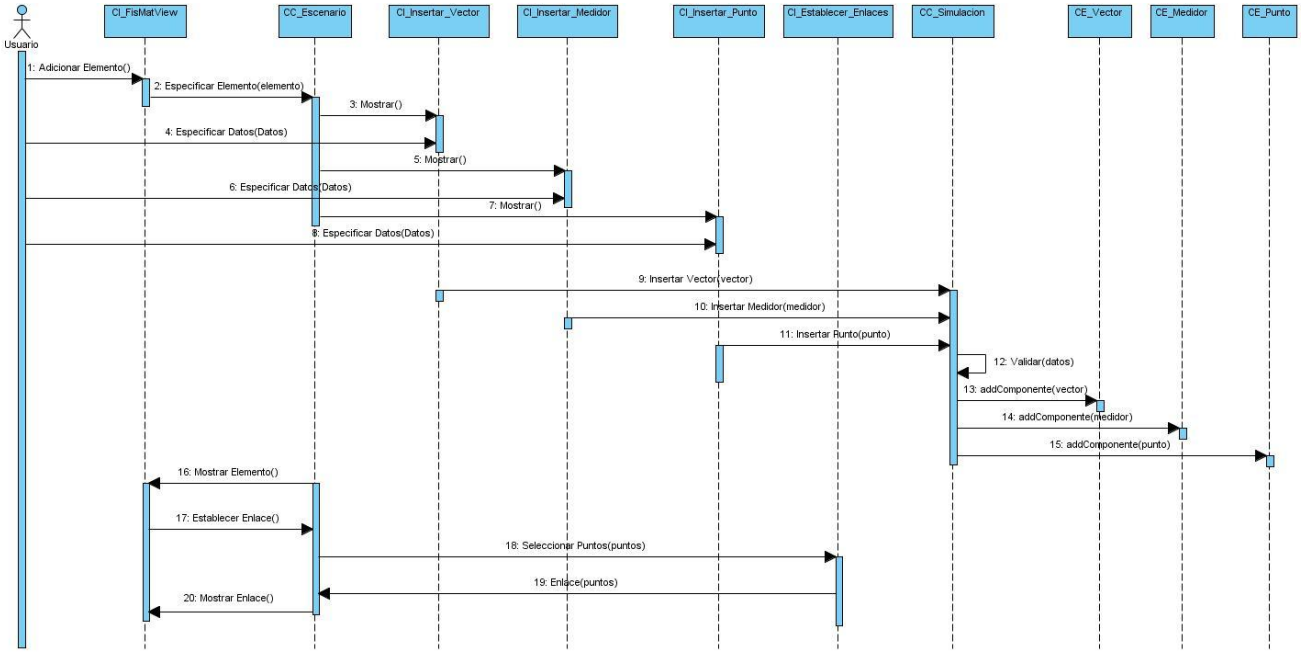



**Figura 15: Diagrama de Secuencia del Diseño del CU Adicionar Elemento al Escenario**

## 3.3 Patrones de Diseño

Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reusable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias). Los patrones

son soluciones de sentido común que deberían formar parte del conocimiento de un diseñador experto. Además facilitan la comunicación entre diseñadores, pues establecen un marco de referencia (terminología, justificación) (Carvajal Pérez, 2009).

Por otro lado, los patrones de diseño, facilitan el aprendizaje al programador inexperto, pudiendo establecer parejas problema - solución. En la programación orientada a objetos resulta complicado descomponer el sistema en objetos (encapsulación, granularidad, dependencias, flexibilidad, reusabilidad y otros), estos patrones permitirán identificar a los objetos apropiados de una manera mucho más sencilla. Además, los patrones de diseño, también ayudarán a especificar las interfaces, identificando los elementos claves en las interfaces y las relaciones existentes entre distintas interfaces. De igual modo facilitarán la especificación de las implementaciones. También, y de forma casi automática, ayudan a reutilizar código, facilitando la decisión entre "herencia o composición" (favorece la composición sobre la herencia y hace uso de la delegación). Relacionan estructuras en tiempo de compilación y en tiempo de ejecución y permiten hacer un diseño preparado para el cambio (Carvajal Pérez, 2009).

Los patrones se pueden clasificar según su propósito (Carvajal Pérez, 2009):

- **Patrones de creación**: Para creación de instancias.
- **Patrones estructurales**: Relaciones entre clases, combinación y formación de estructuras mayores.
- **Patrones de comportamiento**: Interacción y cooperación entre clases.

**Patrones de creación** (Carvajal Pérez, 2009)

Los patrones de creación abstraen la forma en la que se crean los objetos, permitiendo tratar las clases a crear de forma genérica dejando para más tarde la decisión de qué clases crear o cómo crearlas. Según donde se tome dicha decisión se pueden clasificar a los patrones de creación en *patrones de creación de clase* (la decisión se toma en los constructores de las clases y usan la herencia para determinar la creación de las instancias) y *patrones de creación de objeto (se modifica la clase desde el objeto)*.

Entre los principales que se relacionan con esta clasificación se pueden mencionar los siguientes:

- Patrón Factoría

- Patrón Factoría Abstracta
- Patrón Singleton (Instancia Única)
- Patrón Prototipo

**Patrones estructurales** (Carvajal Pérez, 2009)

Tratan de conseguir que cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y, estas están determinadas por las interfaces que soportan los objetos. Estudian cómo se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos.

De estos, se puede hacer referencia a los siguientes:

- Patrón Adaptador
- Patrón Puente
- Patrón Composición
- Patrón Decorador
- Patrón Fachada
- Patrón Proxy

**Patrones de comportamiento**

Los patrones de comportamiento estudian las relaciones de llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal. Dentro de ellos se encuentran los mencionados a continuación:

- Patrón Cadena de Responsabilidad
- Patrón Comando
- Patrón Intérprete
- Patrón Iterador
- Patrón Mediador
- Patrón Recuerdo (Memento)
- Patrón Observador

- Patrón Estado
- Patrón Estrategia
- Patrón Plantilla
- Patrón Visitante

**Patrones GRASP**

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. Constituyen un apoyo para la enseñanza que ayuda a entender el diseño de objeto esencial y aplica el razonamiento para el diseño de una forma sistemática, racional y explicable. GRASP es un acrónimo que significa General Responsibility Asignment Software Patterns (patrones generales de software para asignar responsabilidades).

Dentro de esta clasificación de patrones se encuentran, entre otros:

- Bajo Acoplamiento
- Alta Cohesión
- Experto
- Creador
- Controlador

### 3.3.1 Patrones a utilizar

Después de haber analizado de forma general los patrones de diseño vistos anteriormente se decidió utilizar para el desarrollo del trabajo los patrones GRASP. Estos son usados en el desarrollo de los diagramas de clases del diseño proporcionando que estos sean reusables y flexibles.

De manera general, a continuación se brinda una breve descripción de lo que realiza cada uno de estos patrones y para qué son utilizados fundamentalmente.

- **Experto**: Asignar una responsabilidad al experto en información: La responsabilidad de realizar una labor es de la clase que tiene o puede tener los datos involucrados (atributos). Una clase, contiene toda la información necesaria para realizar la labor que tiene encomendada. Este patrón se usa más que cualquier otro al asignar responsabilidades; es un principio básico que suele utilizarse en el diseño orientado a objetos; expresa simplemente la "intuición" de que los objetos hacen cosas relacionadas con la información que poseen (Larman, 1999). Se pone de manifiesto en las clases entidades, ya que estas son las expertas en la información que poseen.

- **Creador**: Asignar a la clase B la responsabilidad de crear una instancia de clase A en alguno de los siguientes casos (B agrega los objetos de A; B contiene a los objetos de A; B registra las instancias de los objetos de A; B utiliza específicamente los objetos de A; B tiene los datos de inicialización que serán transmitidos a A cuando este objeto sea creado). El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento. Al escogerlo como creador, se da soporte al bajo acoplamiento (Larman, 1999). Principalmente, este patrón es utilizado en las clases controladoras (o clases B: CC_Escenario y CC_Simulación), ya que estas son las encargadas de crear las instancias de las clases que controlan (clases A: CE_Vector, CE_Medidor, CE_Punto y CE_Grafica).

- **Controlador**: Asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas. Esto facilita la centralización de actividades (validaciones, seguridad y otros). La mayor parte de los sistemas reciben eventos de entrada externa, por lo cual, si se recurre a un diseño orientado a objetos, hay que elegir los controladores que manejen esos eventos de entrada (Larman, 1999). En esencia, se evidencian en las clases controladoras (CC_Escenario y CC_Simulación).

- **Alta Cohesión:** Asignar una responsabilidad de modo que la cohesión siga siendo alta. Una clase tiene responsabilidades moderadas en un área funcional y colabora con las otras para llevar a cabo las tareas. Una clase con mucha cohesión es útil porque es bastante fácil darle mantenimiento, entenderla y reutilizarla. Su alto grado de funcionalidad, combinada con una reducida cantidad de operaciones, también simplifica el mantenimiento y los mejoramientos. La

ventaja que significa una gran funcionalidad también soporta un aumento de la capacidad de reutilización (Larman, 1999).

- **Bajo Acoplamiento**: Asignar una responsabilidad para mantener bajo acoplamiento. Debe haber pocas dependencias entre las clases. Si todas las clases dependen de todas ¿cuánto software podemos extraer de un modo independiente y reutilizable en otro proyecto? .Para determinar el nivel de acoplamiento de clases, son muy buenos los diagramas de colaboración de UML. Uno de los principales síntomas de un mal diseño y alto acoplamiento es una herencia muy profunda. Siempre hay que considerar las ventajas de la delegación respecto de la herencia**.** Además soporta el diseño de clases más independientes, que reducen el impacto de los cambios, y también más reutilizables, que acrecienta la oportunidad de una mayor productividad (Larman, 1999).

También se utiliza el patrón de creación Singleton está diseñado para restringir la creación de objetos pertenecientes a una clase o el valor de un tipo a un único objeto. Su intención consiste en garantizar que una clase sólo tenga una instancia y proporcionar un punto de acceso global a ella. El patrón Singleton hace que la clase sea responsable de su única instancia, quitando así este problema a los clientes.

## 3.4 Conclusiones:

El análisis y diseño de un software brinda la primera visión de lo que pudiera ser la solución en el desarrollo del mismo. En este capítulo se ha tratado de mostrar una idea de las funcionalidades que tiene el Módulo Instrumentos del simulador FisMat a través de una serie de modelos que permiten tener un mayor entendimiento de este. Se definieron las clases del análisis y del diseño mostrando sus relaciones, así como la colaboración entre ellas mediante diferentes diagramas. Para lograr un mejor diseño se utilizaron algunos patrones que ayudan a la optimización del mismo.

## Conclusiones generales

Una vez finalizada la investigación y realizado todas las tareas propuestas, se puede afirmar que se cumplió el objetivo planteado pues:

- Se realizó un estudio de los principales simuladores físicos existentes, para encontrar las características más importantes de estos.
- Se realizó una comparación de las diferentes metodologías de desarrollo de software existentes, escogiéndose RUP para el desarrollo del presente trabajo.
- Después de un análisis minucioso y crítico de herramientas CASE quedó definida para el trabajo la herramienta Visual Paradigm, la cual usa el lenguaje de modelado UML para la construcción de diagramas y modelos y se vincula muy bien con la metodología seleccionada.
- Se realizó un modelo del dominio logrando relacionar los principales conceptos existentes en el simulador.
- Se obtuvieron todas las funcionalidades y características que debía cumplir el simulador mediante los requisitos funcionales y no funcionales del mismo, permitiendo realizar descripción de los casos de uso contenidos en la propuesta..
- Se especificaron los casos de uso del sistema y se utilizaron patrones para lograr un mejor entendimiento de los mismos.
- Se realizó el análisis y diseño del Módulo Instrumentos del simulador FisMat.
- Se utilizaron patrones de diseño para optimizar la flexibilidad del diseño realizado.
- Se generaron todos los artefactos necesarios para completar la información necesaria sobre todas las actividades realizadas y lograr una mejor comprensión para la implementación del simulador.

**Recomendaciones:**

- Seguir perfeccionando el modelado de sistema mediante la actualización de los cambios que sean necesarios durante el proceso de implementación y pruebas.

- Realizar talleres u otro tipo de encuentro para el traspaso de conocimiento a otros analistas del proyecto, permitiendo la continuidad del desarrollo del simulador en posteriores versiones del sistema.

## Referencias Bibliográficas


**Alessi, S. M. y Trollip, S. R. 1985.** *Computer-Based Instruction. Method and Development.* New Jersey : Ed. Prentice Hall Inc., 1985.

**Bohigas, Xavier, Jaén, Xavier y Novell, Montse.** INNOVACIONES, APPLETS EN LA ENSEÑANZA DE LA FÍSICA. [En línea] [Citado el: 9 de marzo de 2010.]

**Calero Solís, Manuel. 2003.** willy.net. [En línea] 2003. [Citado el: 11 de marzo de 2010.] http://www.willydev.net/descargas/prev/ExplicaXp.pdf.

**Carvajal Pérez, Erlys. 2009.** *Análisis y diseño del subsistema de Análisis de Resultados de un Simulador de Procesos Químicos.* 2009.

**—. 2009.** *Análisis y diseño del subsistema de Análisis de Resultados de un Simulador de Procesos Químicos.* Ciudad de La Habana : s.n., 2009. pág. pág. 28.

**Computación, Departamento de Ciencia Informáticas y.** assembla. [En línea] [Citado el: 12 de marzo de 2010.] http://www.google.com.cu/url?sa=t&source=web&ct=res&cd=2&ved=0CB0QFjAB&url=http%3A%2F%2Fwww.dsic.upv.es%2F~letelier%2Fpub%2Fp16.ppt&rct=j&q=rup+caracteristicas&ei=a8v7S8jfJMOclgfB5-jWDw&usg=AFQjCNGILsSZrR8u7YbgWl1RDRsDSC8gwg.

**Corporation, Rational Software. 2002.** *Product: Rational Software Corporation.* 2002.

**—. 1998.** *Rational Unified Process. Best Practices for Software Development Teams.* 1998.

**Cuéllar-Vázquez, Edith, Rodríguez-Gómez, Gustavo y Muñoz-Arteaga, Jaime.** Aplicación de Patrones de Software en el Dominio de los Simuladores de Procesos Dinámicos. [En línea] [Citado el: 9 de marzo de 2010.] http://ccc.inaoep.mx/~grodrig/Descargas/PatSSD.pdf.

**García Montero, María de los Ángeles. 2008.** *Ingeniería de Requerimientos aplicada al proceso de Inicio de Investigación y Registro de Notificaciones en el CICPC.* Ciudad de La Habana : s.n., 2008.

**Jacobson, Ivar. 1999.** *El Proceso Unificado de Desarrollo de Software. Volumen 1.* 1999.

**Jacobson, Ivar, Booch, Grady y Rumbaugh, James. 2004.** *El Proceso Unificado de Desarrollo de Software.* Ciudad de La Habana : Félix Varela, 2004. pág. pág. 407. Vol. II.

**Larman, C. 1999.** *UML Y Patrones. Introducción al análisis y diseño orientado a objeto.* México : s.n., 1999.

**Larman, Craig. 2004.** *UML y Patrones de Diseño. Introducción al análisis y diseño orientado a objetos.* La Habana : Félix Varela, 2004.

**Mendoza Sanchez, María A. 2004.** informatizate. [En línea] 7 de junio de 2004. [Citado el: 10 de marzo de 2010.] http://www.willydev.net/Descargas/cualmetodologia.pdf.

**Molpeceres, A. 2002.** Procesos de desarrollo: RUP, XP y FDD. [En línea] 2002. [Citado el: 10 de marzo de 2010.] http://www.javahispano.org/articles.article.action?id=76.

**Périssé, Marecelo Claudio. 2001.** *Proyecto Informático. Una Metodología Simplificada.* 2001.

**Pressman, Roger S. 2002.** *Ingeniería del software "Un enfoque práctico".* Madrid : McGraw-Hill, 2002.

**Rodríguez Trujillo, Isyed. 2008.** *Ingeniería de requerimientos aplicada a la concepción del Portal Web del CICPC.* Ciudada de La habana : s.n., 2008.

**Rumbaugh, James, Jacobson, Ivar y Booch, Grady. 2000.** *El Lenguaje Unificado de Modelado. Manual de Referencia.* s.l. : Addison Wesley, 2000.

**Sánchez, M. 2004.** *Metodologías de Desarrollo de Software.* 2004.

**Software, Departamento Central de Ingeniería de. 2009.** *Entorno Virtual de Aprendizaje.* 3 de noviembre de 2009.

**Software, Departamento central de Ingeniería de. 2009.** *Fase de Inicio. Disciplina de Requisitos. .* s.l. : Universidad de las Ciencias Informáticas, 2009.

**Software, Departamento Central de Ingeniería de. 2009.** *Introducción a la Ingeniería de Software.* s.l. : Universidad de las Ciencias Informáticas, 2009.

**Vaquero López, MsC. Jissie y González Santos, Dr. Ana Isabel.** Un simulador como apoyo visual para el aprendizaje de las técnicas de. [En línea] [Citado el: 10 de marzo de 2010.]http://biblioteca.reduc.edu.cu/biblioteca.virtual/cgi/CD-ROM/otros/UCIENCIA%202007%20.

**Visconti, M. y Astudillo, H. 2003.** *Fundamentos de Ingeniería de Software.* 2003.

**www.areaciencia.com.** Portal de ciencias. [En línea] [Citado el: 9 de marzo de 2010.] http://www.areaciencias.com/DESCARGA%20PROGRAMAS/DESCARGA%20FENOMENOS%20FISICOS.htm.

**www.microsiervos.com.** microsiervos. [En línea] [Citado el: 9 de marzo de 2010.] http://www.microsiervos.com/archivo/ordenadores/phun-simulador.html.

## Anexos

### Anexo 1: Diagramas de colaboración.

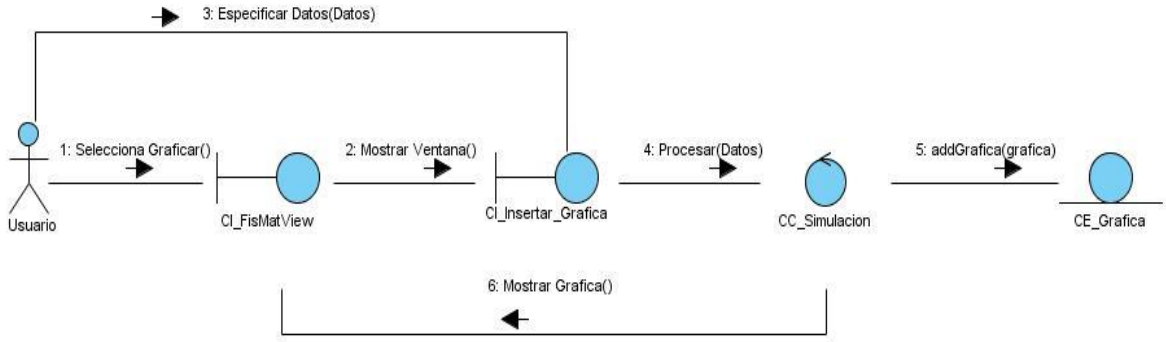



**Figura 16: Diagrama de Colaboración del Análisis del CU Graficar**

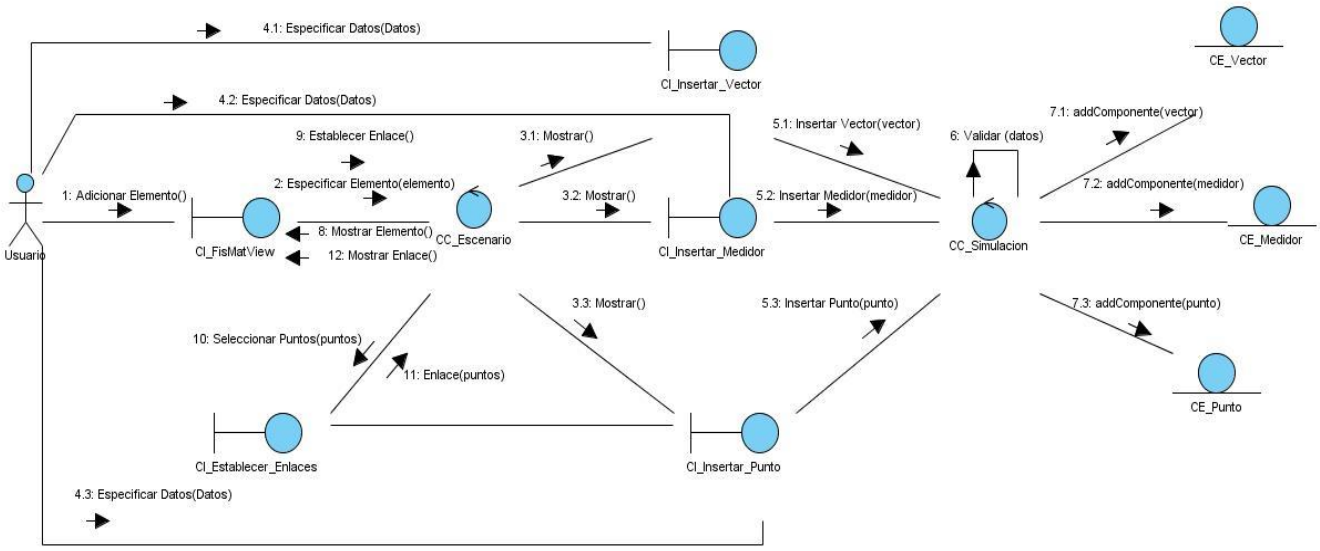



**Figura 17: Diagrama de Colaboración del Análisis del CU Adicionar Elemento al Escenario**

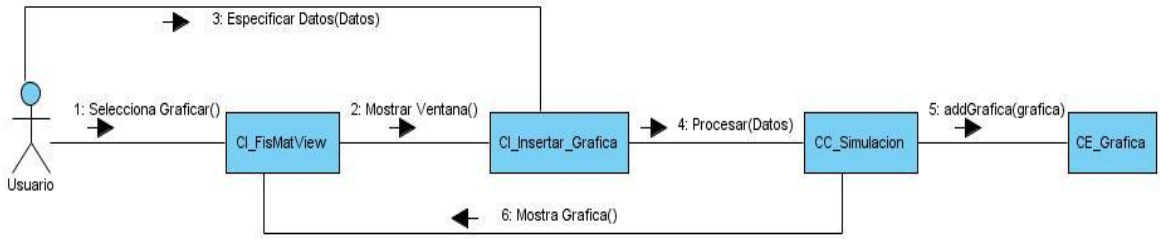



**Figura 18: Diagrama de Colaboración del Diseño del CU Graficar**

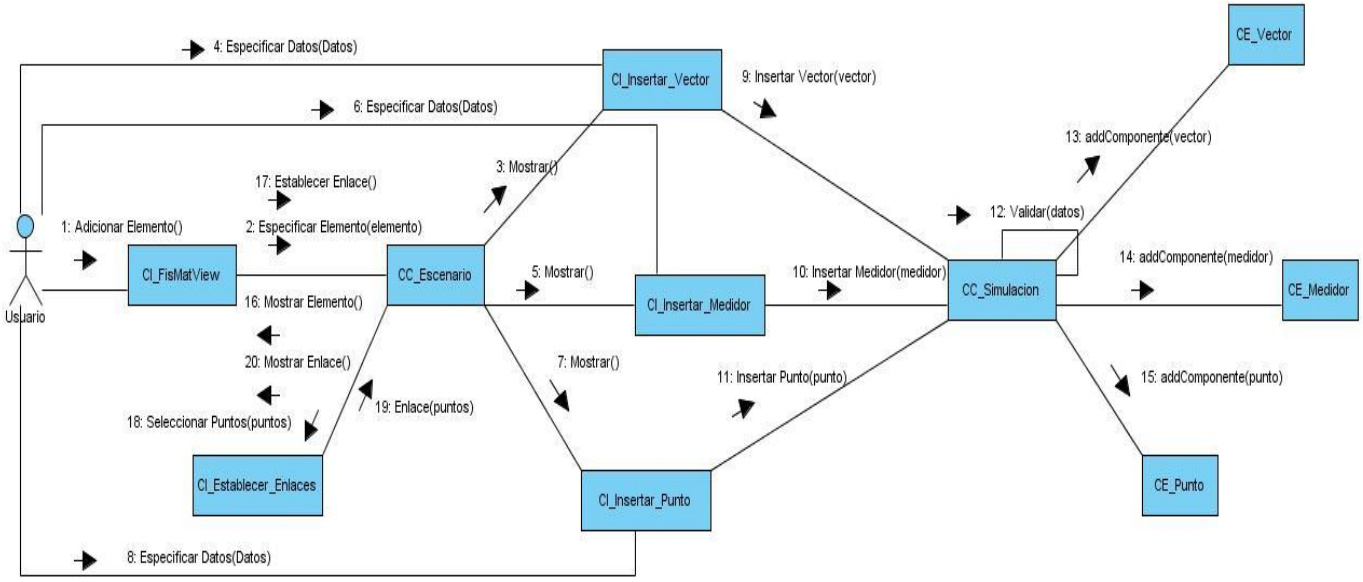



**Figura 19: Diagrama de Colaboración del Diseño del CU Adicionar Elemento al Escenario**

## Anexo 2: Clases del Diseño.

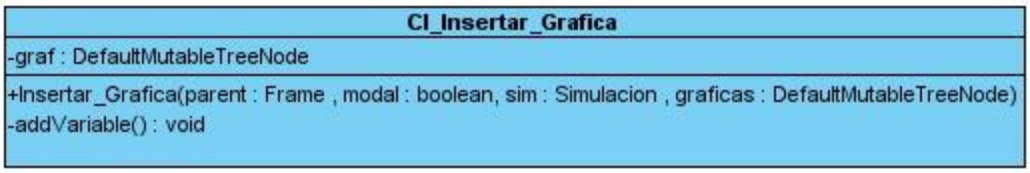



**Figura 20: Clase del Diseño: Insertar_Gráfica**


**Figura 21: Clase del Diseño: FisMatView**

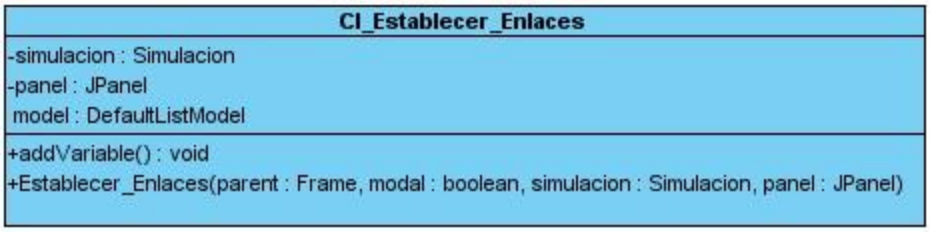



**Figura 22: Clase del Diseño: Establecer_Enlaces**


**Figura 23: Clase del Diseño: Insertar_Medidor**


**Figura 24: Clase del Diseño: Simulacion**

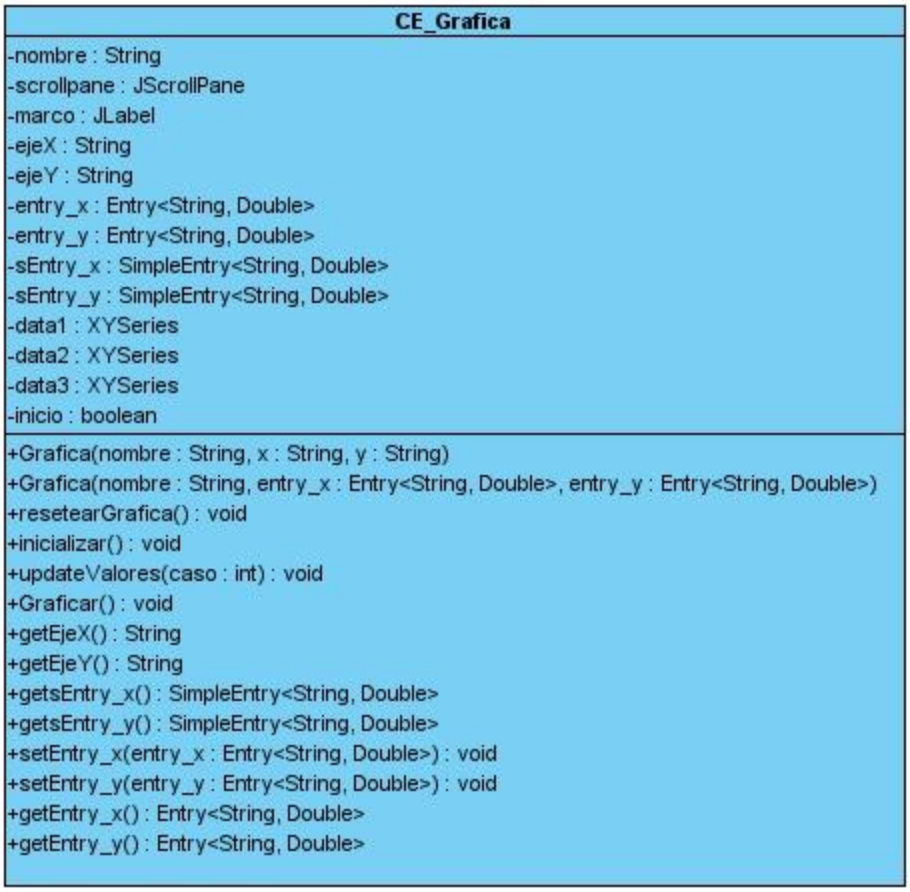



**Figura 25: Clase del Diseño: Grafica**

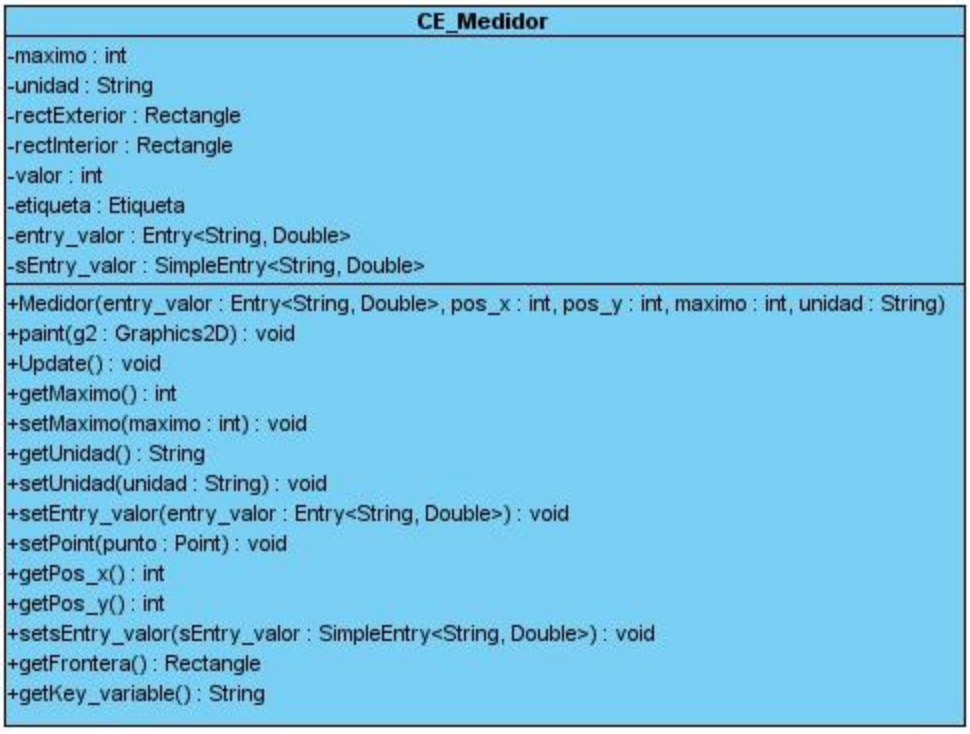



**Figura 26: Clase del Diseño: Medidor**

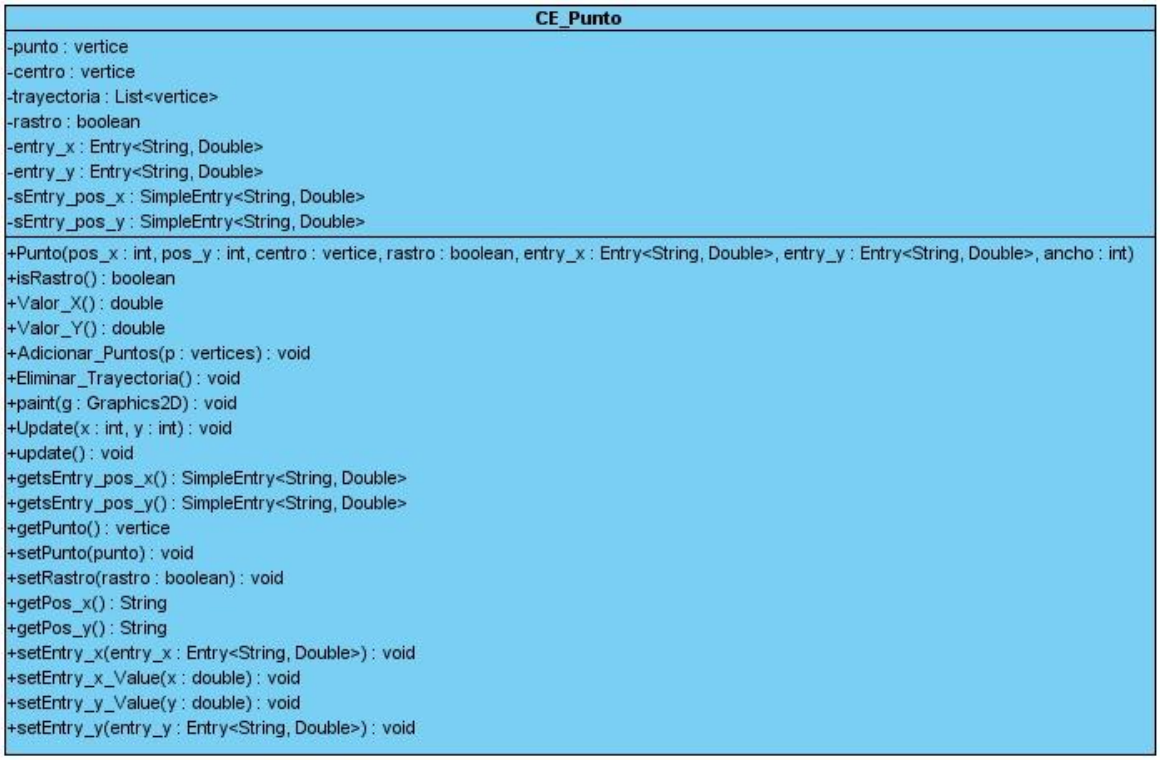



**Figura 27: Clase del Diseño: Punto**

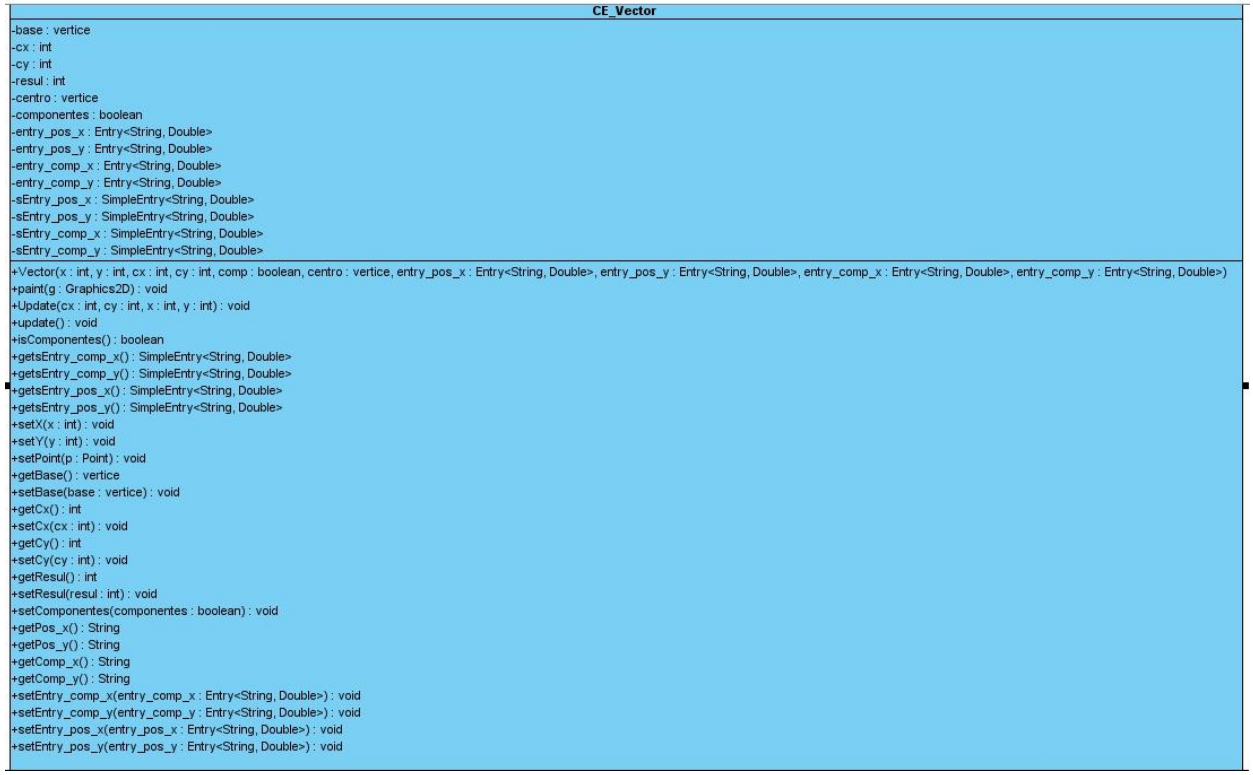



**Figura 28: Clase del Diseño: Vector**

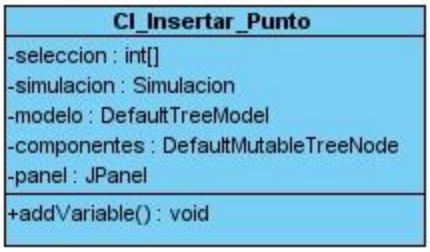



**Figura 29: Clase del Diseño: Insertar_Punto**

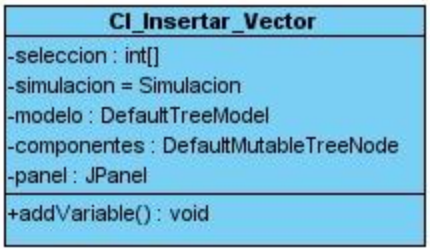



**Figura 30: Clase del Diseño: Insertar_Vector**

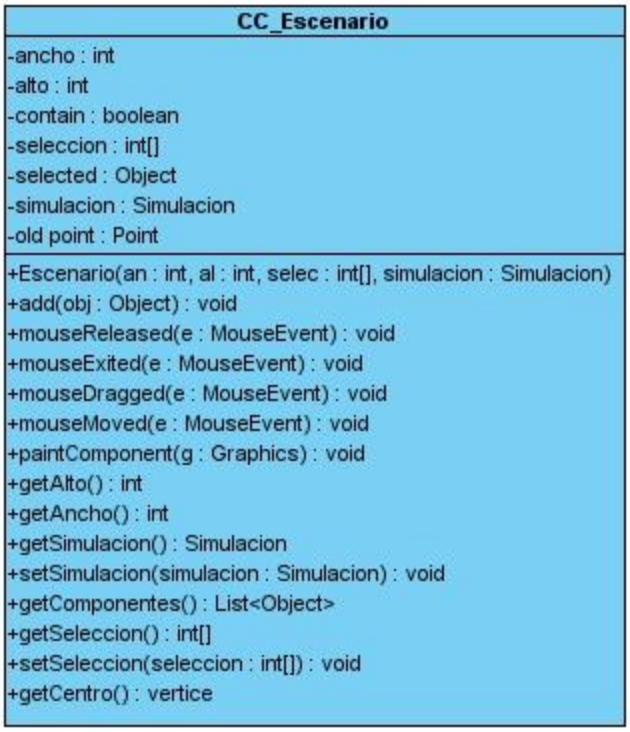



**Figura 31: Clase del Diseño: Escenario**