Universidad de las Ciencias Informáticas

Facultad 9

Título: Desarrollo del módulo Editor Gráfico para la plataforma de construcción de simuladores.

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

Autor: José Eugenio Piñeiro Palmero

Fabian Zambrano Ramos

Tutor: Yoandrys González González

Ciudad de la Habana

Junio - 2010

Declaración de Autoría

Declaro que soy el único autor de este trabajo y autorizo a la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmo la presente a los _____días del mes de _____ del año_____.

_____ _____

José Eugenio Piñeiro Palmero Fabian Zambrano Ramos

_____

Ing. Yoandrys González González

Resumen

La industria química cubana busca lograr una mayor eficiencia en el proceso de producción. Para lograr esto se apoyan en varias técnicas, una de ellas, la simulación. Mediante la simulación de procesos químicos se pueden obtener aproximaciones con un alto nivel de exactitud de todo el proceso de producción. Para lograr la simulación de procesos químicos las industrias de primer nivel se apoyan en software expertos en estas temáticas. Cuba no tiene acceso a estos programas informáticos de primer nivel por diversas razones.

La Universidad de las Ciencias Informáticas (UCI) ha desarrollado diversos programas informáticos que se usan en el país para lograr la independencia tecnológica. En la Facultad 9 de esta universidad existe un grupo de trabajo que se dedica a realizar software de simulación de procesos químicos, pero no se cuenta con editor gráfico para la plataforma de desarrollo de simuladores de procesos químicos. Este trabajo de diploma tiene como objetivo describir el proceso de realización del editor gráfico para la plataforma de desarrollo de simuladores de procesos químicos de la Facultad 9 de la Universidad de las Ciencias Informáticas.

Palabras claves:

Simulación.

Editor gráfico

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Introducción

Se vislumbra, a partir de la historia de las ciencias, la filosofía de las ciencias y la teoría de sistemas, que todo proceso de aprendizaje, o de generación de tecnologías por más variado que sea (el descubrimiento de los elementos químicos que conforman la tabla periódica, la eficiencia de las máquinas térmicas, la simulación de proceso, etc.) se desarrolla lentamente al principio, para dispararse en un momento dado, y luego, nuevamente, aminorar la evolución en el tiempo, tendiendo lentamente a un valor asintótico (límite de eficiencia). Esta curva es la llamada curva de aprendizaje, logística o sigmoidea. (1)

La curva sigmoidea representa sorprendentemente los tramos característicos de evolución de numerosas aplicaciones tecnológicas provenientes de disciplinas diversas. Lo particularmente destacable es que en el campo de la informática como herramienta para resolver problemas de ingeniería recién está entrando en la fase que puede llamarse de crecimiento exponencial. Esto significa que si bien lo recorrido hasta aquí parece asombroso, existe una gran probabilidad que lo que depara el futuro cercano lo sea aún más.

Generalmente la representación de sistemas requiere de una abstracción de la realidad. Obviamente dada la complejidad de los sistemas fisicoquímicos estas construcciones abstractas conocidas como modelos, son solo meras aproximaciones de la realidad.

De aquí se desprende que si bien el sistema real a estudiar es único, puede existir un número muy grande de modelos asociado al mismo. Para adoptar un modelo que pueda resolverse (que sea útil), resulta necesario acoger un conjunto de hipótesis. Las hipótesis son un conjunto de restricciones de exactitud que el problema a resolver impone.

La construcción de un sistema se realiza mediante la representación de un conjunto de ecuaciones, a menudo, ecuaciones diferenciales. Evidentemente no todo sistema de ecuaciones puede resolverse de fácilmente, al menos no desde el punto de vista analítico. Esto impuso a lo largo de la historia limitaciones al tipo de modelo que podían resolverse, o de otra forma, la necesidad de recurrir a hipótesis inadecuadas o restrictivas, para al menos poder tratar el problema.

Grandes pasos se han dado en esta dirección con la llegada de una era fuertemente marcada por los avances alcanzados en el campo de la informática, las técnicas y herramientas que esta proporciona. En la búsqueda de soluciones para representar sistemas lo más aproximados a la realidad y teniendo a mano la gran capacidad en el procesamiento de datos alcanzado en los

últimos años por las máquinas computadoras, se han abierto caminos verdaderamente revolucionarios. Varias son las técnicas que se han descubierto, una de las más importantes para el diseño y operación de sistemas o procesos complejos es la simulación de procesos. La simulación de procesos se convierte en una herramienta muy potente y muy necesaria cuando:

1. Se quiere prever el comportamiento de un sistema antes de construirlo.
2. No existe una completa formulación matemática del problema o los métodos analíticos para resolver el modelo matemático no se han desarrollado aún.
3. Ahorra tiempo y dinero pues ayuda a perfeccionar el prototipo a construir.
4. Los métodos analíticos están disponibles, pero los procedimientos matemáticos son tan complejos y difíciles, que la simulación proporciona un método más simple de solución.
5. Evita malgastar esfuerzo y recursos en la implantación de un sistema que muy probablemente no tendrá éxito.

La simulación de procesos juega un papel significativo en la toma de decisiones, es un factor fundamental en la economía de cualquier nación, ahorra tiempo, esfuerzo y provee el conocimiento necesario para determinar hasta qué punto es factible el desarrollo de un sistema.

La industria química es uno de los eslabones fundamentales encargado de garantizar el desarrollo económico sostenible de una nación, entendido éste como fuente generadora de empleo y riqueza, persistente en el tiempo, compatible con el bienestar y calidad de vida de la población, la conservación y el uso racional de los recursos. Se entiende que la exigencia de un desarrollo sostenible obliga a la industria química a considerar en la definición de sus estrategias, en busca de mejorar la optimización de los procesos, la implantación de nuevas técnicas y tecnologías.

En la industria química moderna la búsqueda de una mayor eficiencia hace necesario un mayor control de los procesos existentes y su constante mejora, dando prioridad a los aspectos relacionados con los componentes que más influyen en los costos. Una de las vías más exitosas para lograr todo lo anteriormente expuesto es el desarrollo de software para la simulación de procesos químicos.

Actualmente se dispone de generalizadas y poderosas herramientas de simulación: Aspen Plus, Hysys, Sugar, Chemcad lideres en la simulación de procesos químicos, POWERFACTS de la Dow Chemical, GPSS II, CSL y CHIPS de IBM; GASP y GPS de la Corporación del Acero de USA; CHEOPS de la Compañía Petrolera Shell, PSPICE para simular circuitos eléctricos, PSS/E para la simulación de flujos de potencia en redes eléctricas son algunas muestras.

El problema con estas herramientas es que son propiedad de grandes trasnacionales. Unas veces son demasiado caras para poder pagar sus licencias y otras debido al bloqueo político y económico impuesto a Cuba por los Estados Unidos, aún contando con el presupuesto para comprarlas se prohíbe vendérselas a Cuba. Si bien Cuba, como respuesta a aquellas naciones que imponen o apoyan injustamente el bloqueo político y económico usa sus productos software sin pagar licencia, esto acarrea varios problemas, como por ejemplo la comercialización de software desarrollado sobre este tipo de productos, además no es así con aquellas naciones que no lo hacen, a las cuales el país haciendo un gran esfuerzo le compra el software necesario para impulsar el desarrollo de sectores importantes como son salud, educación, defensa nacional, cultura, deporte, turismo, etc.

El caso es que la mayoría de las herramientas líderes en simulación de procesos químicos a nivel mundial no están disponibles por diferentes razones para la industria química o cualquier sector de la economía cubana. Su uso en el país está autorizado solo para fines académicos.

Como alternativa al monopolio tecnológico impuesto por las grandes trasnacionales, a principios de la década de los ochenta Richard Stallman habla por primera vez de los conceptos de de software libre y crea la Fundación de Software Libre. Hoy en día se dispone de muy buenas herramientas de simulación de código abierto que permiten resolver una gran variedad de problemas de forma eficiente y rápida.

En Cuba se han realizado varios software de simulación de procesos químicos en pos de contribuir al desarrollo del país, y se han llegado a aplicar en procesos completos o subprocesos en diferentes ramas de la industria. En el Instituto Cubano de Investigaciones de los Derivados de la Caña de Azúcar (ICIDCA) se dispone de un Simulador, orientado a ecuaciones del proceso azucarero y de refinación de alcohol de caña llamado SIMFAD 3.0, resultado de todo un amplio trabajo anterior de modelación matemática y simulación de equipos y subprocesos de la industria azucarera. En la Universidad Central de Las Villas (UCLV) también se desarrolló un simulador de procesos tecnológicos, pero no aparecen referenciadas aplicaciones en la industria cubana. Otro simulador desarrollado en el MINAZ (Villa clara) es el

denominado AGE de carácter determinístico y que contiene recomendaciones de cómo actuar en función de los resultados. Pero al igual que otro paquete de simulación de la UCLV está basado en métodos de cálculos simplificados o rápidos que no permiten obtener todos los resultados importantes y necesarios que pueden dar los simuladores que usan modelos más precisos y sin simplificaciones innecesarias si se dispone de computadoras, estos simuladores están fuertemente limitados de manera general por la poca capacidad de análisis y síntesis de los resultados de la simulación y la carencia de interfaces gráficas. Aunque algunos cuentan con aditamento de este tipo, estas son muy pobres desde el punto de vista funcional y no permiten que el trabajo con estas herramientas sea flexible y amigable.

Debido a esta situación se tiene como problema la necesidad de un editor gráfico para una plataforma para el desarrollo de simuladores de procesos químicos en el grupo de simulación del polo de la facultad 9 de la Universidad de las Ciencias Informáticas. Para poder suplir esta necesidad se tiene como objetivo general: desarrollar un (Framework para la construcción de) editores gráficos para una plataforma que permitan realizar simuladores de procesos químicos utilizando técnicas estándares de diseño de interfaces gráficas, teniendo como objeto de estudio el proceso de visualización y configuración de los componentes de un simulador de procesos químicos a través de interfaces gráficas, el cuál será aplicado en el polo Petrosoft de la Facultad 9 de la Universidad de las Ciencias Informáticas. Si la plataforma de desarrollo de simuladores de procesos contara con un editor gráfico, los simuladores que se desarrollen en el grupo de simulación del polo Petrosoft de la Facultad 9 tendrán mayores facilidades a la hora de diseñar los modelos a simular.

Para poder realizar el editor gráfico se tendrá que hacer las siguientes tareas:

1. Evaluar el contenido de la información obtenida sobre el desarrollo de editores gráficos, establecer un diagnóstico de las tendencias actuales y tomar posición al respecto.
2. Caracterizar los principales simuladores existentes respecto a su concepción y las características y funcionalidades de los editores gráficos de los mismos.
3. Estudiar sistemas de simulación de procesos químicos.
4. Seleccionar la metodología y herramientas a utilizar en el desarrollo del subsistema en cuestión.
5. Especificar de los requerimientos funcionales y no funcionales del editor gráfico.
6. Realizar el análisis y diseño del editor gráfico.
7. Implementar el editor gráfico.

Métodos Científicos Investigativos

Para realizar un buen proceso de investigación científica se necesitan aplicar varios métodos que están definidos para realizar diversas fases de la investigación y permiten llegar a conclusiones de forma objetiva siguiendo un sistema de principios y normas de razonamiento (2). Estos métodos están divididos en dos grandes grupos: los teóricos y los empíricos. La presente investigación se apoya en los métodos teóricos: históricos, modelación, hipotético deductivo y sistemático y en el método empírico la observación.

Para hacer una investigación sobre determinado temática se hace necesario saber su historia y antecedentes, para poder lograr esto es necesario hacer un estudio profundo de su historia, por lo tanto, para que se pueda lograr ese conocimiento se aplicará el método histórico.

Para el desarrollo del software es necesario en muchos casos, el uso de modelos para lograr un mejor entendimiento de lo que se quiere hacer. Estos modelos se desarrollan mediante el método de la modelación, además de estar vinculado este método a lo que se refiere a los distintos tipos de modelos de simulación que se investigarán, por lo que se considera de máxima importancia este método investigativo.

Siempre que se realiza una investigación científica es porque se quiere demostrar la existencia de algo. Las investigaciones se basan en hipótesis, que son comprobadas al final de la investigación para ver si se pudo llegar al objetivo de la investigación. El método hipotético es orientado hacia la hipótesis o idea a defender durante la investigación, es aquí donde se ve vinculado dicho método investigativo.

Para poder estudiar determinados fenómenos hace falta casi siempre la observación, en este caso se verá vinculada durante todo el proceso de la investigación y desarrollo del trabajo. Donde primero se verá vinculada será durante la observación de las problemáticas que existen para la edición gráfica de los simuladores que se desarrollan en el polo de Petrosoft de la Facultad 9 de la UCI.

CAPÍTULO 1: Fundamentación Teórica.

En este capítulo se hará una breve introducción a la temática de la simulación, las interfaces gráficas en los programas de ordenadores, así como un acercamiento a los orígenes de la misma, una breve descripción de la situación por la que pasa el proyecto de simulación del Polo Petrosoft de la Facultad 9 de la Universidad de las Ciencias Informáticas. Así como la influencia que tendrá en el desarrollo de los simuladores de procesos químicos cubanos para lograr una independencia tecnológica sobre esta rama y sustituir importaciones de software.

1.1 Conceptos asociados al dominio del problema

Existen un conjunto de conceptos asociados al problema planteado anteriormente, a los cuales se le dará explicación en los próximos acápites.

1.1.1 Simulación.

Shannon define simulación como: “el proceso de diseñar un modelo de un sistema real y llevar a término experiencias con él, con la finalidad de comprender el comportamiento del sistema o evaluar nuevas estrategias -dentro de los límites impuestos por un cierto criterio o un conjunto de ellos- para el funcionamiento del sistema”. Por lo que se entiende que el proceso de simulación incluye tanto la construcción del modelo como su uso analítico para estudiar un problema. (3)

Thomas H. Naylor la define así: "Simulación es una técnica numérica para conducir experimentos en una computadora digital. Estos experimentos comprenden ciertos tipos de relaciones matemáticas y lógicas, las cuales son necesarias para describir el comportamiento y la estructura de sistemas complejos del mundo real a través de largos períodos de tiempo". (2)

De una forma más formal la simulación es la técnica que se utiliza para poder diseñar un modelo de un sistema real y obtener los resultados del mismo en un sistema informático antes de que ocurran en el sistema real. Sirve para evaluar varias alternativas y nos ayuda a tomar una decisión sobre cuál debemos escoger.

En la actualidad la simulación es una de las herramientas más importantes y más interdisciplinarias. Las aplicaciones de la simulación parecen no tener límites. Actualmente se simulan los comportamientos hasta las partes más pequeñas de un mecanismo, el desarrollo de las epidemias, el sistema inmunológico humano, las plantas productivas, sucursales bancarias, movimiento de los planetas, la evolución del universo son algunos ejemplos de la aplicación de esta herramienta. La simulación de procesos industriales y procesos químicos

son, hoy en día, un factor fundamental en los aportes que hace la industria al desarrollo de un país, pues proporciona el conocimiento necesario para la toma de decisiones y la determinación de hasta qué punto es factible la realización de un proyecto.

Sistemas de simulación

- **Sistema discreto**

En el sistema discreto las variables de estado cambian solo en puntos discretos o contables en el tiempo. Un ejemplo típico de simulación discreta ocurre en las colas donde están interesados en la estimación de medidas como la longitud de la línea de espera. Tales medidas solo cambian cuando un cliente entra o sale del sistema; en todos los demás momentos, no ocurre nada en el sistema desde el punto de vista de la inferencia estadística.

- **Sistemas continuos**

Las variables de estado cambian en forma continua a través del tiempo. Un ejemplo típico de simulación continua es el estudio de la dinámica de la población mundial.

1.1.2 Modelos

Un modelo de un sistema es una descripción o especificación de dicho sistema y su entorno con algún propósito en particular. Con frecuencia, se representa combinando gráficos y textos. Estos últimos, pueden escribirse en lenguaje natural o en algún lenguaje de modelado. (3)

Otra definición sería: conjunto de elementos que describe una realidad física, abstracta o hipotética. (4)

Un modelo es una construcción intelectual y descriptiva de una entidad en la cual un observador tiene interés. (4)

Los modelos son la descripción o especificación de un sistema y su ambiente para cierto propósito y frecuentemente representado como una combinación de dibujo y texto. Se utilizan normalmente para representar el comportamiento de las relaciones entre los componente de un sistema.

Tipos de modelos

- **Dinámicos**

Utilizados para representar sistemas cuyo estado varía con el tiempo. (5)

- **Estáticos**

Utilizados para representar sistemas cuyo estado es invariable a través del tiempo. Ejemplo: Arquitectónicos Autocad. (4)

- **Matemáticos**

Representan la realidad de forma abstracta de diversas maneras. Ejemplo: Gráficas de Ecuaciones. (5)

- **Físicos**

Son aquellos en que la realidad es representada por algo tangible, construido en escala o que por lo menos se comporta de forma análoga a esa realidad (maquetas, prototipos, modelos análogos, etc). Ejemplo: Túnel de Viento. (5)

- **Analíticos**

La realidad se representa por fórmulas matemáticas. Estudiar el sistema consiste en operar con estas fórmulas matemáticas (resolución de ecuaciones). (5)

- **Numéricos**

Se tiene el comportamiento numérico de las variables que intervienen. No se obtiene ninguna solución analítica. Ejemplo: La simulación numérica de yacimientos petroleros. (5)

- **Continuos**

Representan sistemas cuyos cambios de estado son graduales. Las variables que intervienen son continuas. Ejemplo: Simular el flujo de agua por una cañería. (5)

- **Discretos**

Representan sistemas cuyos cambios de estados son de salto. Las variables varían de forma discontinua. (5)

- **Determinísticos**

Son modelos cuya solución para determinadas condiciones es única y siempre la misma, para una entrada dada y estado el sistema siempre responde igual. Ejemplo: Una empacadora de alimentos. (6)

- **Estocásticos**

Representan sistemas donde los hechos suceden al azar, lo cual no es repetitivo. No se pueden asegurar si tales acciones ocurren en un determinado instante. Se conoce la probabilidad de ocurrencia y su distribución probabilística. (5)

1.1.3 Programación

La programación es el proceso de creación de un programa de computadora, mediante la aplicación de procedimientos lógicos, a través de los siguientes pasos:

1. El desarrollo lógico del programa para resolver un problema en particular.
2. Escritura de la lógica del programa empleando un lenguaje de programación específico (codificación del programa).
3. Ensamblaje o compilación del programa hasta convertirlo en lenguaje de máquina.
4. Prueba y depuración del programa.
5. Desarrollo de la documentación.

Tipos de programación

Existen tres tipos fundamentales de programación:

- **Programación generativa**

Paradigma de ingeniería del software basado en el desarrollo de familias de sistemas. El producto final de la Programación Generativa es un modelo generativo capaz de sintetizar todos los programas de una familia a partir de especificaciones de alto nivel de abstracción. (7)

- **Programación declarativa**

Paradigma de programación basado en la lógica en el que se estudian de forma simple muchos aspectos avanzados de los lenguajes de programación modernos. Este estilo de programación encuentra numerosas aplicaciones industriales en campos como las bases de datos, ingeniería del software, procesadores de lenguajes, lenguaje natural, investigación operativa, seguridad de redes, etc. (8)

- **Programación orientada a objetos**

Según John Zukowski, programador y escritor de libros sobre Java, la programación orientada a objetos es el modo de desarrollar software describiendo problemas mediante el uso de

elementos u objetos desde el espacio del problema y no mediante un conjunto de pasos secuenciales que se ejecutarán en el ordenador. (9)

1.1.4 Interfaz gráfica

Según Carlos Marrero Expósito, graduado de Estudios Avanzados en la Universidad de La Laguna con el proyecto de investigación "Aproximación semiótica y cognitiva a la interfaz gráfica de usuario", interfaz gráfica de usuario es un artefacto interactivo, que por su diseño y a través de ciertos interfaces humanos, posibilita la interacción de una persona con el sistema informático, haciendo uso de las gramáticas visuales y verbales.

De una manera más técnica se define a interfaz de usuario, como conjunto de componentes empleados por los usuarios para comunicarse con las computadoras. El usuario dirige el funcionamiento de la máquina mediante instrucciones, denominadas genéricamente entradas. Las entradas se introducen mediante diversos dispositivos, por ejemplo un teclado, y se convierten en señales electrónicas que pueden ser procesadas por la computadora. Estas señales se transmiten a través de circuitos conocidos como bus, y son coordinadas y controladas por la unidad de proceso central y por un soporte lógico conocido como sistema operativo. Una vez que la UPC ha ejecutado las instrucciones indicadas por el usuario, puede comunicar los resultados mediante señales electrónicas, o salidas, que se transmiten por el bus a uno o más dispositivos de salida, por ejemplo una impresora o un monitor.

La idea fundamental en el concepto de interfaz es el de mediación, entre hombre y máquina. La interfaz es lo que "media", lo que facilita la comunicación, la interacción, entre dos sistemas de diferente naturaleza, típicamente el ser humano y una máquina como el computador. Esto implica, además, que se trata de un sistema de traducción, ya que los dos "hablan" lenguajes diferentes: verbo-icónico en el caso del hombre y binario en el caso del procesador electrónico.

Resumiendo entonces podemos concluir que, una interfaz de gráfica es la parte de una aplicación que el usuario ve y con la cual interactúa. Está relacionada con la subyacente estructura, la arquitectura, y el código que hace el trabajo del software, pero no se confunde con ellos. La interfaz incluye las pantallas, ventanas, controles, menús, metáforas, la ayuda en línea, la documentación y el entrenamiento. Cualquier cosa que el usuario ve y con lo cual interactúa es parte de la interfaz. Una interfaz inteligente es fácil de aprender y usar. Permite a los usuarios hacer su trabajo o desempeñar una tarea en la manera que hace más sentido para ellos, en vez de tener que ajustarse al software. Una interfaz inteligente se diseña específicamente para la gente que la usará.

Características de las interfaces gráficas.

1. Posee un monitor gráfico de alta resolución.
2. Posee un dispositivo apuntador (típicamente un ratón).
3. Promueve la consistencia de la interfaz entre programas.
4. Los usuarios pueden ver en la pantalla los gráficos y textos tal como se verán impresos.
5. Sigue el paradigma de la interacción objeto-acción.
6. Permite la transferencia de información entre programas.
7. Se puede manipular en la pantalla directamente los objetos y la información.
8. Provee elementos de interfaz estándar como menús y diálogos.
9. Existe una muestra visual de la información y los objetos (iconos y ventanas).
10. Proporciona respuesta visual a las acciones del usuario.
11. Existe información visual de las acciones y modos del usuario/sistema (menús, paletas).
12. Existen controles gráficos (widgets) para la selección e introducción de la información.
13. Permite a los usuarios personalizar la interfaz y las interacciones.
14. Proporciona flexibilidad en el uso de dispositivos de entrada (teclado/ratón).

Fases de diseño de una interfaz gráfica

En el proceso de diseño de una interfaz de usuario se pueden distinguir cuatro fases o pasos fundamentales:

- Reunir y analizar la información del usuario.
- Diseñar la interfaz de usuario.
- Construir la interfaz de usuario.
- Validar la interfaz de usuario.

Reunir y analizar la información del usuario:

Para esto se concretan a través de técnicas de recogida de requerimientos, qué tipo de usuarios van a utilizar el programa, qué tareas van a realizar los usuarios y cómo las van a realizar, qué exigen los usuarios del programa, en qué entorno se desenvuelven los usuarios (físico, social, cultural).

Diseñar la interfaz de usuario

Es importante dedicar tiempo y recursos a esta fase, antes de entrar en la codificación. En esta fase se definen los objetivos de usabilidad del programa, las tareas del usuario, los objetos y acciones de la interfaz, los iconos, vistas y representaciones visuales de los objetos, los menús de los objetos y ventanas. Todos los elementos visuales se pueden hacer primero a mano y luego refinar con las herramientas adecuadas.

Elementos de diseño de pantalla y su percepción visual:

- **Análisis de Color**

Es probablemente el elemento de la interfaz que con más frecuencia es mal utilizado. El color comunica información, no es sólo decorativo (ejemplo: reforzar mensajes de error). Deben utilizarse combinaciones adecuadas (por ejemplo, las paletas proporcionadas por los sistemas operativos). El color debe atraer la atención, pero no cansar después de un rato de trabajo. Es especialmente importante seguir las líneas de diseño existentes. Principio básico: diseñar primero en blanco y negro, y luego añadir el color.

- **Análisis Animación**

Se define como un cambio en el tiempo de la apariencia visual de un elemento gráfico. Ejemplos de su uso: progreso de acciones (copia de ficheros en Windows 95, instalación de programas), estado de procesos (iconos de impresora), acciones posibles (cambios en el cursor al desplazar el ratón). La animación puede ayudar a subrayar iconos importantes, mostrar el estado de un objeto particular o explicar su comportamiento.

1.2 Objeto de estudio

Descripción general

Mucho ha llovido desde que el ordenador personal Xerox Alto utilizara por primera vez una interfaz gráfica de usuario moderna en 1973, el descubrimiento de este magnífico componente por parte de Centro de Investigaciones Xerox Palo Alto (PARC por sus siglas en inglés) ha sido un elemento crucial en el acercamiento de las personas a las máquinas computadoras. Las interfaces gráficas han evolucionado con el trascurso de los años, poco a poco trasladándose a sistemas operativos, aplicaciones escritorios, aplicaciones web, dispositivos móviles de comunicación, etc., cautivando hasta los usuarios más exigentes.

Las interfaces gráficas de usuarios (IGU) juegan un importante papel en el desarrollo y puesta en marcha de un proyecto, es un factor fundamental en la aceptación o rechazo del mismo, de esta manera es de suma importancia estandarizar y formalizar su diseño. En el diseño visual de interfaces están involucradas varias disciplinas de las que se puede citar: diseño de comunicación visual, diseño industrial y arquitectura.

Debido al lugar que ocupan las IGU dentro de un proyecto en los últimos años los diferentes paradigmas de desarrollo siguen una tendencia que involucra a la interfaz de usuario como una parte muy importante del proceso desarrollo, dentro de los que se puede citar el muy conocido Modelo Vista Controlador.

El campo de la simulación no ha estado exento a los avances que han alcanzado en post de humanizar la relación hombre-computadora y han sido significativos los avances en busca de:

- Elegancia y simplicidad.
- Proporcionalidad, contraste y escalamiento.
- Organización estructural.
- Modularidad.
- Representación de imágenes.
- Guías de estilo.
- Estandarización.
- Interacción.
- Manejo de errores.

1.3 Descripción actual del dominio del problema

La simulación en Cuba se encuentra en una etapa inicial. La aplicación de la disciplina es prematura en el país debido a que la mayoría de las herramientas de renombre a nivel internacional solo pueden ser utilizadas con fines académicos. Actualmente se disponen de pocas herramientas que puedan ser aplicadas a la simulación de procesos químicos y las desarrolladas en el país hasta el momento están fuertemente limitadas con respecto a las funcionalidades que son capaces de proporcionar.

Con el objetivo de mejorar el estado en que se encuentran los software de simulación en el país y teniendo en cuanta el aporte que puede significar el desarrollo de software de este tipo a la economía, se pretende desarrollar en el grupo de trabajo de simulación perteneciente al polo Petrosoft un framework para la construcción de simuladores.

1.4 Situación Problemática

El país se encuentra hoy en una batalla a muerte contra el consumo innecesario de energía y otros recursos. Las industrias luchan por aumentar la calidad de los productos y producir más con menos, aumentando la eficiencia. La simulación es una herramienta que tributa a que estas metas sean alcanzadas con mayor efectividad y rapidez. Insertar la simulación a la industria y a todos los sectores sociales es hoy día un reto para el país.

En la actualidad existen generalizadas y poderosas herramientas de simulación que permiten simular una variedad bastante amplia de problemas de forma eficiente. El inconveniente es que estas herramientas son software privativo, que pertenecen a grandes trasnacionales, inaccesibles por diferentes razones para la industria o cualquier sector de la economía cubana. Están disponibles para nuestro país solo con fines académicos.

El país está enfrascado en lograr soberanía tecnológica para ello se dedican grandes esfuerzos en desarrollar nuestros propios productos, el objetivo de este trabajo es contribuir con ese esfuerzo y desarrollar un modulo editor gráfico que lime las asperezas que presentan sus homólogos anteriormente construidos en Cuba, pues una de las principales deficiencias que presentan estos simuladores es que adolecen de interfaces gráficas flexibles y amigables para sus usuarios finales, además de estar fuertemente limitadas por las funcionalidades que proporcionan.

Qué se ha hecho hasta el momento?

Aproximadamente desde la década de los 60 empieza a desarrollarse y aplicarse la simulación en el campo de la informática, simuladores como el POWERFACTS de la Dow Chemical, GPSS II, CSL y CHIPS de IBM; GASP y GPS de la Corporación del Acero de USA; CHEOPS de la Compañía Petrolera Shell, PSPICE para simular circuitos eléctricos, PSS/E para la simulación de flujos de potencia en redes eléctricas, vieron la luz. La mayoría de estos simuladores son del tipo determinísticos y asumen condiciones de estado estacionario. Otros como el GASP II y el GPSS están orientados para la simulación probabilística de procesos como las colas que ocurren en el procesamiento de templas y el SPEEDUP para la simulación dinámica.

Todas estas generalizadas y poderosas herramientas de simulación son propiedad privada de grandes trasnacionales, inaccesibles por diferentes razones para la industria o cualquier sector de la economía cubana.

Como alternativa al monopolio tecnológico impuesto por las grandes trasnacionales, a principios de la década de los 80 Richard Stallman habla por primera vez de los conceptos de de software libre y crea la Fundación de Software Libre. Debido a la popularidad que ha alcanzado el software libre en los últimos años, en la actualidad se cuenta con potentes herramientas de desarrollo de software que ofrecen su código.

Hoy en día se dispone de muy buenas herramientas de simulación de código abierto que permiten resolver una gran variedad de problemas de forma eficiente y rápida. SimiOO, una herramienta pública que permite trazas de referencia a memoria; RIJYA, simulador para la maquina elemental JASP escrito en java script; MOCSPROC herramienta que se especializa en simulación de procesos contractivos; SIMTEGUC programa modular para la simulación de procesos en el tratamiento de emisiones atmosféricas son muestras de algunos simuladores que se han venido desarrollando.

En el empeño de contribuir al desarrollo del país, disminuyendo el costo en importaciones y dar un avance sustancial al mejoramiento de la tecnología, proporcionar software necesario para el adelanto de la medicina, la industria, la calidad de vida, al proceso docente educativo, teniendo en cuenta que la simulación es una potente herramienta que tiene una fuerte aplicación en estos campos, en Cuba se han realizado varios trabajos que circundan en este ámbito, desarrollándose diversos simuladores, dentro de los que se puede citar como ejemplo, el "TERMOAZUCAR", proyecto que se inició como una modificación del GEMCS en el período 1973-74 para poder aplicarlo en el proceso azucarero, decisión que resultó ser acertada desde el punto de vista económico pues un simulador como el SUGARS , desarrollado por la IBM para la industria azucarera australiana, tiene un costo inicial de 18,000 USD y para explotarlo hay que pagar cuotas adicionales. ACOPLA, SIMFAD y SIDEL para la industria alcoholera. De estos programas desarrollados en Cuba para la industria azucarera y alcoholera "TERMOAZUCAR" es el único de naturaleza modular-secuencial y que utiliza un sistema experto para el análisis de los resultados. STA, actualmente en desarrollo en la Universidad de Ciencias Informáticas, se ha desarrollado hasta el momento en plataforma .Net. El STA es un simulador estacionario de sistemas tecnológicos y termo energéticos. Este simulador presenta varias funcionalidades como:

1. Análisis de la factibilidad técnica a través de los indicadores fundamentales del proceso.
2. Análisis de la factibilidad económica de la solución.

3. Análisis de dependencia entre variables seleccionadas por el usuario.
4. Análisis de sensibilidad de una variable con respecto a otras.

En Cuba se han desarrollado varios simuladores y se han llegado a aplicar en procesos completos o subprocesos en diferentes ramas de la industria. En el Instituto Cubano de Investigaciones de los Derivados de la Caña de Azúcar (ICIDCA) se dispone de un simulador, orientado a ecuaciones del proceso azucarero y de refinación de alcohol de caña llamado SIMFAD 3.0, resultado de todo un amplio trabajo anterior de modelación matemática y simulación de equipos y subprocesos de la industria azucarera. En la Universidad Central de Las Villas (UCLV) también se desarrolló un simulador de procesos tecnológicos, pero no aparecen referenciadas aplicaciones en la industria cubana. Otro simulador desarrollado en el MINAZ (Villa clara) es el denominado AGE de carácter determinístico y que contiene recomendaciones de cómo actuar en función de los resultados. Pero al igual que otro paquete de simulación de la UCLV está basado en métodos de cálculos simplificados o rápidos que no permiten obtener todos los resultados importantes y necesarios que pueden dar los Simuladores que usan modelos más precisos y sin simplificaciones innecesarias si se dispone de computadoras (6)

1.5 Conclusiones parciales

Con el editor gráfico del simulador de procesos industriales se dará un paso de avance en los simuladores cubanos porque la mayoría de ellos no cuentan con funcionalidades gráficas que ayuden a comprender los procesos de simulación. Las anteriores interfaces de los simuladores presentan limitaciones que afectan la correcta comprensión de los procesos de simulación. Por estas razones se considera de vital importancia el desarrollo del editor gráfico de la plataforma de desarrollo de simuladores de procesos químicos.

CAPITULO 2: Tendencias y tecnologías actuales a desarrollar

Introducción

El desarrollo de software no es una tarea fácil, si bien es importante que los desarrolladores de software estén altamente cualificados, es evidente la necesidad de una herramienta que guíe a los desarrolladores en el proceso de desarrollo de software, pues la disciplina es joven aún. Las metodologías de desarrollo de software son esta herramienta, proporcionan normas, capturan y presentan las mejores prácticas que el estado de la tecnología actual permite, aportan una guía para las actividades de un equipo de desarrollo, dirigen las tareas de cada desarrollador por separado y del equipo en conjunto, ofrecen criterios para el control, medición de los productos y actividades del proyecto.

2.1 Metodologías de desarrollo del software

Usar o no una metodología de desarrollo de software

El concepto de metodología dentro de la ingeniería de software, es sin duda uno de los más oscuros y que más confusión produce tanto en estudiantes como en profesionales involucrados en el proceso de desarrollo de software.

Todo desarrollo de software es riesgoso y difícil de controlar, aunque las metodologías suponen una gran carga y amenazan con hacer más lento el proceso de desarrollo e incluso retrasar la terminación del producto, si no se lleva una metodología de por medio lo que se obtiene es clientes insatisfechos, resultados inesperados, detección tardía de errores, la introducción de nuevas herramientas afectará perjudicialmente al proceso, resultados distintos con nuevas clases de producto, rigidez. La ausencia de metodología en el desarrollo de un proyecto de software garantiza con seguridad también la ausencia de calidad.

¿Qué tipo de metodología utilizar?

Actualmente existen dos enfoques fundamentales hacia la metodología a utilizar, las tradicionales y las ágiles.

2.1.1 Metodologías Tradicionales

Las metodologías tradicionales fueron creadas por las décadas 70 y 80 para dotar a la ingeniería de software de un proceso disciplinado con el fin de hacer el desarrollo de software

más eficiente y predecible. Estas metodologías están centradas fundamentalmente en el control de procesos estableciendo rigurosamente las actividades involucradas, los artefactos que se deben producir y las herramientas y notaciones que se usarán. Estas propuestas han demostrado ser útiles en un gran número de proyectos, generalmente proyectos de gran tamaño (recursos y tiempo).

Entre las principales metodologías tradicionales se pueden encontrar los ya tan conocidos RUP y MSF entre otros, que centran su atención en llevar una documentación exhaustiva de todo el proyecto y centran su atención en cumplir con un plan de proyecto, definido todo esto, en la fase inicial del desarrollo del proyecto.

Proceso Unificado de Desarrollo de Software (RUP)

Provee un acercamiento disciplinado para asignar tareas y responsabilidades dentro de una organización de desarrollo. Su objetivo es asegurar la producción de software de alta calidad que satisfaga los requerimientos de los usuarios finales (respetando cronograma y presupuesto). Fue desarrollado por Rational Software, y está integrado con todas las herramientas Rational. Puede ser adaptado y extendido para satisfacer las necesidades de la organización que lo adopte. Es guiado por casos de uso y centrado en la arquitectura, y utiliza UML como lenguaje de notación. (10)

RUP es:

- Un proceso de desarrollo de software — el conjunto de actividades necesarias para transformar los requisitos de un cliente/usuario en un sistema de software.
- Más que un proceso individual — un armazón genérico para procesos que puede ser especializado para una clase grande de sistemas de software, para diferentes dominios de aplicación, tipos de organizaciones, niveles de competencia, y tamaños de proyectos.
- Es un proceso basado en componentes — el sistema de software que se construye está hecho de componentes de software interconectadas vía interfaces bien definidas.
- Usa UML para preparar todos los planos del sistema de software — UML y RUP fueron desarrollados conjuntamente. (11)

Fases

1. Inicio
2. Elaboración
3. Construcción
4. Transición

Ventajas

1. Evaluación en cada fase que permite cambios de objetivos.
2. Funciona bien en proyectos de innovación.
3. Es sencillo, ya que sigue los pasos intuitivos necesarios a la hora de desarrollar el software.
4. Seguimiento detallado en cada una de las fases.

Desventajas

1. La evaluación de riesgos es compleja.
2. Excesiva flexibilidad para algunos proyectos.
3. Pone a los clientes en una situación que puede ser incómoda para ellos.
4. El cliente debe ser capaz de describir y entender a un gran nivel de detalle para poder acordar un alcance del proyecto.

2.1.2 Metodologías Ágiles

El proceso de desarrollo hasta hace poco llevaba asociado un marcado énfasis en el control del proceso mediante una rigurosa definición de roles, actividades y artefactos incluyendo modelado y documentación detallada. Este enfoque tradicional ha demostrado ser efectivo y necesario para proyectos de gran tamaño (respecto a tiempo y recursos). Sin embargo este enfoque no resulta ser tan efectivo en muchos de los proyectos actuales donde intervienen factores como, entornos cambiantes, se exige reducir drásticamente los tiempos de desarrollos manteniendo una alta calidad.

Como alternativa a las restricciones de flexibilidad y tiempo que traen consigo la aplicación de las metodologías tradicionales, surge el enfoque ágil. Este enfoque está especialmente

orientado a proyectos pequeños, aporta una elevada simplificación que a pesar de ello no renuncia a las prácticas esenciales para asegurar la calidad del producto. Las metodologías ágiles son sin duda uno de los temas recientes de la ingeniería de software que están acaparando gran interés.

Entre los principales métodos ágiles se encuentran XP (eXtreme Programming), Scrum, Cristal Methods, AUP, entre otras.

En febrero de 2001, tras una reunión celebrada en Utah-EEUU, nace el término ágil aplicado al desarrollo de software. En esta reunión participan un grupo de 17 expertos de la industria del software, incluyendo algunos de los creadores o impulsores de metodologías de software. Su objetivo fue esbozar los valores y principios que deberían permitir a los equipos desarrollar software rápidamente y respondiendo a los cambios que puedan surgir a lo largo del proyecto. Se pretendía ofrecer una alternativa a los procesos de desarrollo de software tradicionales, caracterizados por ser rígidos y dirigidos por la documentación que se genera en cada una de las actividades desarrolladas. (12)

Como resultado de esta nueva teoría se crea un Manifiesto Ágil cuyas principales ideas son:

- Los individuos y las interacciones entre ellos son más importantes que las herramientas y los procesos empleados.
- Es más importante crear un producto software que funcione que escribir documentación exhaustiva.
- La colaboración con el cliente debe prevalecer sobre la negociación de contrato.
- La capacidad de respuesta ante un cambio es más importante que el seguimiento estricto de un plan

Estas metodologías ponen de relevancia que la capacidad de respuesta a un cambio es más importante que el seguimiento estricto de un plan. Lo proponen porque para muchos clientes esta flexibilidad será una ventaja competitiva y porque estar preparados para el cambio puede significar reducir su coste.

Es conveniente elegir metodologías ágiles cuando:

1. Los requerimientos son poco claros o altamente volátiles.
2. El cliente entiende el proceso y está involucrado en el proyecto.

3. Se cuenta con profesionales capacitados y competentes.
4. Se tienen canales ricos de comunicación.
5. El grupo de trabajo no es demasiado grande. Generalmente menor de diez personas.
6. Se desea fomentar la mejora continua del proceso (13)

AUP

El AUP es un acercamiento al desarrollo de software basado en el proceso unificado Rational de IBM, el ciclo de vida de UP es serial en lo grande e iterativo en lo pequeño.

Flujos de trabajo de Agile UP:

1. **Modelado:** La meta de este flujo es entender el negocio de la organización, el dominio del problema que el proyecto aborda e identificar una solución viable para abordar el dominio del problema.
2. **Implementación:** La meta de este flujo es transformar su modelo(s) en un código ejecutable y realizar una prueba de nivel básico en una unidad particular de prueba.
3. **Pruebas:** La meta de este flujo es ejecutar una evaluación de los objetivos para asegurar la calidad. Esto incluye encontrar defectos, validar que el sistema funcione como fue diseñado y verificar que los requerimientos están completos.
4. **Despliegue:** La meta de esta disciplina es planificar la entrega del sistema y ejecutar el plan para que el sistema esté disponible para los usuarios finales.
5. **Administración de la Configuración:** La meta de de este flujo es administrar el acceso a los entregables o productos del proyecto. Esto incluye no sólo el rastreo de versiones del producto en el tiempo, sino que también incluye controlar y administrar los cambios que ocurran.
6. **Administración del Proyecto:** La meta de de este flujo es dirigir las actividades que se llevan a cabo en el proyecto. Esto incluye administración del riesgo, la dirección de personas (asignar tareas, seguimiento de los procesos, etc.), y coordinar con los sistemas y personas fuera del alcance del proyecto para que este termine a tiempo y dentro del presupuesto.
7. **Ambiente:** La meta de este flujo es apoyar el resto de los esfuerzos por garantizar que, el proceso adecuado, la orientación (normas y directrices) y herramientas (hardware, software, etc.) estén disponibles para el equipo según sea necesario (14)

Fases

Inicio: Identificar el alcance inicial de proyecto, una arquitectura inicial del sistema y obtener un presupuesto inicial del proyecto y una aceptación de los involucrados.

Elaboración: Probar arquitectura del sistema.

Construcción: Construir un software funcional sobre una base regular e incremental, las cuales cumplan con las prioridades más importantes para los involucrados o usuarios del proyecto.

Transición: Validar y desplegar el sistema en su ambiente de la producción. (14)

Metodologías Ágiles	**Metodologías Tradicionales**
Basadas en heurísticas provenientes de prácticas de producción de código.	Basadas en normas provenientes de estándares seguidos por el entorno de desarrollo.
Especialmente preparados para cambios durante el proyecto.	Cierta resistencia a los cambios.
Impuestas internamente (por el equipo).	Impuestas externamente.
Proceso menos controlado, con pocos principios.	Proceso mucho más controlado, con numerosas políticas/normas.
No existe contrato tradicional o al menos es bastante flexible.	Existe un contrato prefijado.
El cliente es parte del equipo de desarrollo.	El cliente interactúa con el equipo de desarrollo mediante reuniones.
Grupos pequeños (<10 integrantes) y trabajando en el mismo sitio.	Grupos grandes y posiblemente distribuidos.
Pocos artefactos.	Más artefactos.
Pocos roles.	Más roles.
Menos énfasis en la arquitectura del	La arquitectura del software es esencial y

software.	se expresa mediante modelos.

Tabla 1 Comparación entre Metodologías.

2.1 Lenguajes de programación.

Una de las definiciones más comunes de lenguaje de programación es “una notación para comunicarle a una computadora lo que queremos que hagas” pero esta notación carece de formalidad.

Un lenguaje de programación es un sistema notacional para describir computaciones en una forma legible tanto para la máquina como para el ser humano. (15)

En la búsqueda de encontrar el mejor entendimiento entre las máquinas computadoras y el hombre, los programadores y desarrolladores de software han puesto todo su empeño en hacer evolucionar los lenguajes de programación. Los populares y famosos lenguajes con los que se trabaja hoy día distan mucho de lo que fueron sus homólogos hace varias décadas, se han alcanzado grandes avances en cuanto a nivel de atracción y facilidad para escribir e interpretar el código.

A partir de lenguajes de bajo nivel como por ejemplo ensamblador se ha ido progresando, los lenguajes se fueron haciendo cada vez más independientes de un tipo específico de máquinas y alcanzando cada vez niveles más altos de abstracción, hasta llegar a los actuales, sencillos y poderosos lenguajes de alto nivel, C, C++, JAVA entre otros.

2.2 Selección de las tecnologías de desarrollo

Debido a que la tecnología extranjera importada es cara, ya que en varias ocasiones se cuenta con del dinero y por el bloqueo económico y político impuesto por los EEUU a nuestro país por más de cinco décadas, no se puede obtener el producto necesitado. En busca de soberanía y autonomía tecnológica, desde el año 2005 Cuba tiene intención de migrar al software libre. En el empeño de contribuir con el país y hacer de la revolución cubana una revolución cada vez más fuerte, autónoma y soberana, este trabajo se desarrollará totalmente sobre tecnología libre.

2.2.1 JAVA

Como plataforma

JAVA es una plataforma que está compuesta por una gama amplia de tecnologías, cada una de las cuales ofrecen una parte del complejo de desarrollo JAVA. La plataforma JAVA es una Plataforma de Software, que se ejecuta sobre otras plataformas de software -Sistemas Operativos-. La plataforma java tiene dos componentes:

Máquina Virtual de JAVA: es la base de la plataforma JAVA y puede ser incorporada en todos los Sistemas Operativos. Contiene el intérprete de JAVA.

Interfaz de Programación de Aplicaciones (Java API): es una colección de componentes de software que proveen una amplia gama de funcionalidades. Está agrupado en paquetes o librerías de componentes relacionadas. (16)

Actualmente la plataforma JAVA es una de las más difundidas. La característica de poder correr un programa escrito en JAVA o cualquier programa que compile a bytecode sobre prácticamente cualquier Sistema Operativo, su capacidad para ampliar el API, además la amplia gama de herramientas que proporciona, entre otras características hacen de JAVA una plataforma muy atractiva.

Como Lenguaje de Programación.

Varias son las historias acerca del origen de JAVA, desde que fue un proyecto que revotó por varios departamentos de la empresa Sun Mycrosistems sin que nadie se interesara por él, hasta la que dice que fue en lenguaje diseñado para pequeños electrodomésticos.

El caso es que JAVA es un lenguaje de programación orientado a objeto diseñado por Gosling de Sun Mycrosistems entre 1994 y 1995. Está diseñado para poder ejecutarse en distintos tipos de procesadores, su sintaxis es bastante parecida a la de c++, aunque tomas ideas de Modula-3, Smalltalk, y Objetive-C, también hace como suyas características que en otros lenguajes se encuentran disponibles como extensiones. JAVA es uno de los lenguajes actuales más utilizados.

Características para la selección de java.

- **Simple:** sin unions, typedefs, preprocesors, aritmética de punteros, sobrecarga de operadores, herencia múltiple, funciones.
- **Orientado a objetos:** puro.

- **Tipado estáticamente:** primero compilado a “byte-codes” después interpretado por intérprete Java.
- **Independiente de la Arquitectura y Portable**.
- **Garbage collected:** libera al programador de desasignar memoria.
- **Robusto:** el intérprete controla todos los accesos al sistema, no hay crash del sistema.
- **Seguro:** verifica todos los accesos a Map. No hay acceso a áreas del sistema no autorizadas.
- **Multi-threaded:** los programadores pueden contener múltiples threads de ejecución lo que permite la concurrencia de tareas.
- **Extensible:** soporta métodos nativos (métodos implementados en otros lenguajes).

Una característica importante de JAVA es que puede ser utilizado en tres formas.

- Aplicaciones stand-alone.
- Applets: Aplicaciones que corren en un navegador.
- Servlets: Aplicaciones que corren en un servidor.

Niveles de seguridad.

- Fuertes restricciones al acceso a memoria, como son la eliminación de punteros aritméticos y de operadores ilegales de transmisión.
- Rutina de verificación de los códigos de byte que asegura que no se viole ninguna construcción del lenguaje.
- Verificación del nombre de clase y de restricciones de acceso durante la carga.
- Sistema de seguridad de la interfaz que refuerza las medidas de seguridad en muchos niveles. (17)

2.2.2 Entorno de desarrollo integrado.

¿Qué es un entorno de desarrollo integrado?

Entorno de desarrollo integrado o IDE (acrónimo en inglés de integrated development enviroment) es un conjunto de herramientas destinadas especialmente para un programador.

Un IDE es un conjunto de herramientas que han sido empaquetadas como un programa de aplicación, para brindar facilidades y crear un entorno agradable programación. Los IDE pueden ser aplicaciones por sí solas o pueden ser parte de aplicaciones existentes. Incluyen un editor de código, un compilador, un depurador, administrador de proyectos e integración con repositorios.

Eclipse como IDE.

Eclipse es un proyecto desarrollado inicialmente por IBM, cuyo código fue puesto a disposición de los usuarios. A partir de febrero de 2004 el proyecto eclipse es apadrinado por un consorcio de empresas entre las que se encuentran (IBM, Borland, QNX Software Systems, Rational Software, Red Hat, SuSE, TogetherSoft, Webgain, Hitachi, Instantiations, MontaVista, Scapa Technologies, Trans-Enterprise Integration, Serena, ETRI, HP, MKS, SlickEdit, Oracle, AltoWeb, Catalyst Systems, Flashline, Parasoft, SAP, teamstudio y TimeSys).

En la web oficial (www.eclipse.org) se define como “An IDE for everything and for nothing in particular” (un IDE para todo y para nada en particular). Varias autores prefieren llamar a eclipse como plataforma, otros como IDE. Los autores de este trabajo consideran a eclipse como una herramienta, un armazón que proporciona un conjunto de servicios sobre el cual se puede construir un entorno de desarrollo agradable y extensible mediante la adición de plugins.

Arquitectura de eclipse

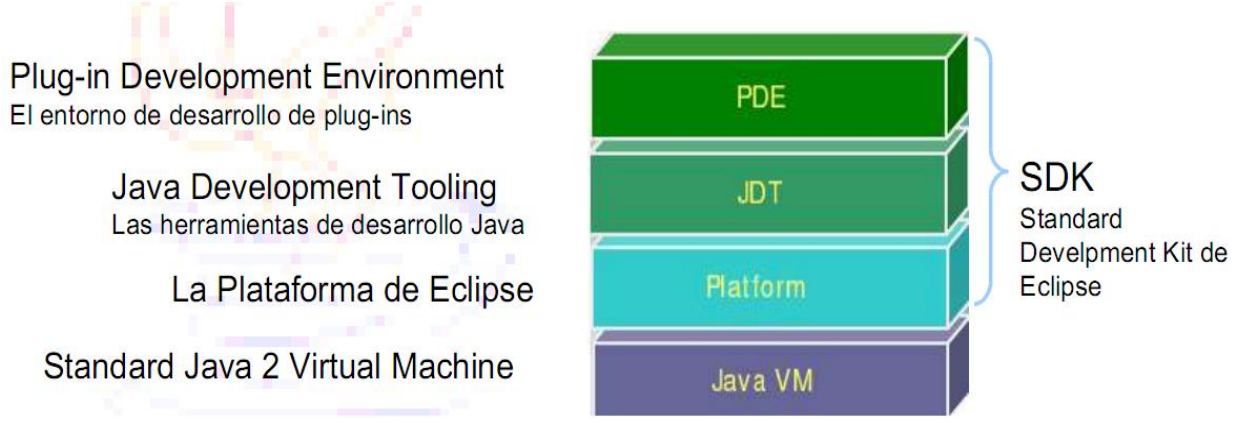


Figura 1 Arquitectura de Eclipse.

La Plataforma: es el núcleo básico o el kernel de Eclipse; emplea una estructura abierta de plug-ins (extensiones) que permite expandir las capacidades de la plataforma base; es de arquitectura abierta, pues es un producto de código fuente abierto u open source. Está escrita en Java.

Provee a las capas superiores de servicios tales como editor de código fuente, infraestructura para depuración independiente del lenguaje de programación, soporte de versiones, búsqueda, compilación, asistentes para creación, etc.

El JDT*:* agrupa un conjunto de plug-ins que extienden la plataforma básica proporcionando características de edición compilación, depuración y ejecución de código Java. Explica a la plataforma cómo entender Java. Viene incluido en el SDK.

El PDE: proporciona herramientas y asistentes que automatizan y facilitan la creación, desarrollo, depuración y distribución de plug-ins.

Selección de la herramienta CASE.

Desde que UML se convirtió en todo un estándar para la modelación orientada a objetos de sistemas informáticos, la selección de una herramienta para el modelado es una tarea complicada. En la actualidad existen en este ámbito un sin número de herramientas tanto de código abierto como propietarias para dar apoyo a la ingeniería de software.

En (7) se propone una metodología desarrollada para la selección de una herramienta CASE. Una característica que influyó fuertemente en los autores de este trabajo para la selección de la herramienta independientemente de la serie de características bástate atractivas que se comentan a continuación, es que para la Universidad de las Ciencias Informáticas, centro donde se realiza este trabajo, está disponible la licencia de la herramienta.

2.2.3 Visual Paradigm como herramienta CASE.

Visual Paradigm para el Lenguaje Unificado de Modelado (VP-UML) es una herramienta completamente equipada para el UML, diseñada para la construcción de sistemas de software de gran escala de forma fiable a través de la utilización del enfoque orientado a objetos. VP - UML soporta los últimos estándares de Java y UML anotaciones. Además, Las transiciones de diseño a la aplicación se encuentran perfectamente integradas en la herramienta de modelado visual, reduciendo así significativamente los esfuerzos en todas las etapas del desarrollo del ciclo de vida del software. (8)

VP-UML es una herramienta de modelado que apoya gran parte del ciclo de desarrollo del software. Se integra con los siguientes entornos de desarrollo:

- Eclipse/IBM WebSphere
- JBuilder
- NetBeans IDE
- Oracle JDevelope
- Visual Studio
- IntelliJ IDEA

Características de Visual Paradigm.

1. Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI.
2. Diagramas de Procesos de Negocio - Proceso, Decisión, Actor de negocio, Documento.
3. Modelado colaborativo con CVS y Subversion (nueva característica).
4. Ingeniería de ida y vuelta.
5. Ingeniería inversa - Código a modelo, código a diagrama.
6. Ingeniería inversa Java, C++, Esquemas XML, XML,.NET exe/dll, CORBA IDL.
7. Generación de código: Modelo a código, diagrama a código.
8. Editor de Detalles de Casos de Uso: Entorno todo en uno para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso.
9. Diagramas EJB - Visualización de sistemas EJB.
10. Generación de código y despliegue de EJB´s - Generación de beans para el desarrollo y despliegue de aplicaciones.
11. Diagramas de flujo de datos.

12. Soporte ORM: Generación de objetos Java desde la base de datos.
13. Generación de bases de datos: Transformación de diagramas de Entidad-Relación en tablas de base de datos.
14. Generador de informes para generación de documentación.
15. Distribución automática de diagramas: Reorganización de las figuras y conectores de los diagramas UML.
16. Importación y exportación de ficheros XMI.
17. Integración con Visio - Dibujo de diagramas UML con plantillas (stencils) de MS Visio.
18. Analizador de impacto con matriz de diagrama. (19).

Por todo lo antes planteado se decide utilizar como herramienta de modelado para el desarrollo de este proyecto, Visual Paradigm.

2.3 Conclusiones parciales

Luego de un estudio minucioso acerca de tecnologías y las herramientas actuales se decidió utilizar como lenguaje de programación a Java para la implementación por las posibilidades multiplataforma que este brinda. Se escogió Eclipse IDE como ambiente de desarrollo por sus grandes potenciales y para guiar el proceso de desarrollo se seleccionó AUP como metodología de desarrollo.

CAPÍTULO 3: Descripción de la solución propuesta.

Introducción

En este capítulo se describe la solución propuesta para el desarrollo del Editor Gráfico para la plataforma de desarrollo de simuladores, se definen los requerimientos funcionales y no funcionales, a partir de los cuales se obtienen y describen los casos de usos que guiarán la solución que se desarrolla utilizando la metodología seleccionada. También se mostrarán el modelo de dominio, los diagramas de secuencias y modelo de clases del diseño.

3.1 Especificaciones del contenido

La aplicación contará con una paleta de módulos, los cuales podrán ser insertados en el área de edición. Además se contará con una barra de herramientas donde se podrán realizar las distintas modificaciones a las imágenes que se encuentren en el área de edición. También tendrá un menú donde se le permitirá al usuario guardar diagramas, ver reportes, etc.

La paleta de módulos estará compuesta por los distintos módulos que pueda tener una fábrica de procesos químicos, así como turbinas, generadores, etc. También tendrá entre sus elementos el elemento conexión, que servirá para la representación de la unión entre dos módulos en la fábrica en cuestión.

3.2 Descripción del sistema propuesto

La propuesta de solución es el desarrollo de un editor gráfico que permita realizar modelos de sistemas de simulación, ver información sobre los distintos módulos y corrientes, y realizar modificaciones sobre la misma. Además debe permitir guardar la información gráfica para que pueda ser cargada por el usuario en el momento que estime conveniente.

3.3 Descripción del modelo conceptual.

Al no poderse identificar un proceso de negocio se decide realizar un modelo del dominio para una mayor comprensión de los conceptos del sistema. La descripción de este modelo de dominio se hace mediante un diagrama de clases UML, donde intervienen las principales clases que están presentes en el sistema.

3.3.1 Conceptos asociados al dominio.

Se le denomina **Módulos** a los diferentes equipos que forman parte de una fábrica, como pueden ser Turbinas, Generadores Eléctricos, Motores, etc.

Se le denomina **Corrientes** al flujo de las diferentes sustancias que viajan a través de los diferentes módulos, es con el producto que trabaja la fábrica.

Se le denomina **Puerto** al objeto que contiene la información de las corrientes en el módulo que pertenece.

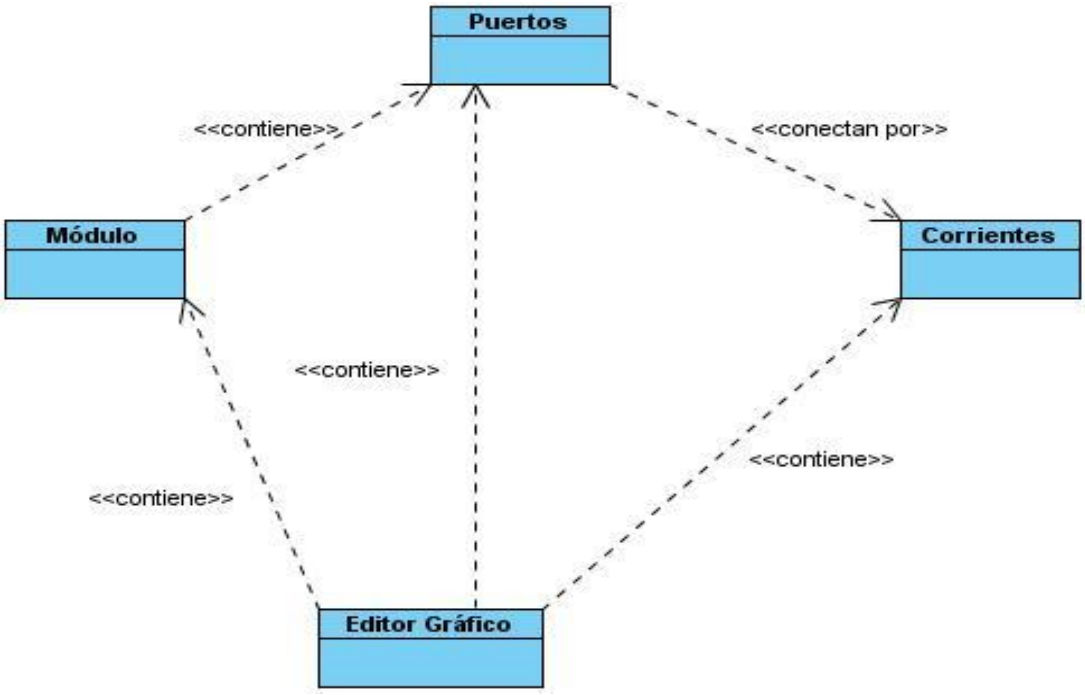


Figura 2 Diagrama de clases del dominio.

3.4 Requerimientos Funcionales (RF)

En AUP, al igual que en el proceso unificado de desarrollo de software, todo es dirigido por los casos de usos, y a partir de estos se generan una serie de artefactos y modelos. Los requisitos funcionales en el desarrollo del software son la representación de las funcionalidades que debe cumplir el sistema. De forma general a partir de cada requerimiento funcional se genera un caso de uso. A continuación se mencionan los requerimientos funcionales del sistema y se mencionará a que caso de uso se relaciona y su nivel de prioridad, así como una descripción de los mismos.

RF1 Mostrar las propiedades de las sustancias en los módulos.

Debe permitir que se muestre toda la información asociada al módulo en cuestión.

Descripción: Los clientes tienen la necesidad de que una vez introducida la información a los módulos pueda ser consultada en cualquier momento. Para poder cumplir con este requerimiento se decide que el caso de uso del sistema Mostrar información de los módulos solucionará dicha necesidad del cliente de poder ver la información de los módulos en cualquier momento. El nivel de prioridad de este requerimiento es crítico.

RF2 Mostrar información de la corriente.

Se muestra la información de las corrientes después de realizada la simulación.

Descripción: Los clientes tienen la necesidad de que una vez se la simulación del proceso se pueda acceder a la información de los valores de cada una de las corrientes asociadas al proceso, permitiendo ver los valores esperados que se deben producir cuando se lleve a cabo el proceso en la realidad. Para poder cumplir con este requerimiento se decide que el caso de uso del sistema Mostrar información de las corrientes solucione dicha necesidad. El nivel de prioridad de este requerimiento es crítico.

RF3 Modificar información de la corriente.

Debe permitir que se inserte la información de la corriente.

Descripción: Para poder realizar los procesos de simulación, el cliente tiene que introducir la información con la cual se va a simular, esta información es introducida en cada una de las corrientes. Para poder cumplir con esta funcionalidad se decide realizar el caso de uso del sistema Modificar información de las corrientes. El nivel de prioridad de este requerimiento es crítico.

RF4 Modificar Información de módulo que el usuario seleccione.

Debe permitir que se inserte la información inicial del módulo.

Descripción: Se necesita introducir la información de los distintos módulos, esta información es insertada antes de que se realice la simulación. El caso de uso del sistema que está relacionado con este requerimiento es el caso de uso del sistema Modificar Información del módulo. El nivel de prioridad de este requerimiento es crítico.

RF5 Adicionar Módulo al área de edición del editor.

Debe permitir que se adicione algún módulo hacia el área de edición.

Descripción: Es necesario insertar en el área de edición los distintos módulos. El caso de uso del sistema asociado a este requerimiento es Adicionar Módulo. El nivel de prioridad de este requerimiento es crítico.

RF6 Conectar Módulos de forma gráfica en el área de edición del editor.

Debe permitir que se haga una conexión entre un módulo y otro.

Descripción: Se necesita conectar dos o más módulos entre ellos, cuando se conectan dos módulos se crea una corriente. El caso de uso del sistema asociado es Crear Corriente. El nivel de prioridad de este requerimiento es crítico.

RF7 Gestionar Módulos.

Debe permitir las distintas operaciones de edición como cortar, copiar, pegar, mover, eliminar y rotar.

Descripción: El cliente necesita que se puedan realizar sobre cada módulo distintas operaciones como copiar, cortar, pegar, mover, eliminar y rotar. Este requerimiento será asociado a varios casos de usos, ellos son Copiar Módulo, Cortar Módulo, Pegar Módulo, Mover Módulo, Eliminar Módulo y Rotar Módulo. El nivel de prioridad de este requerimiento es crítico.

RF8 Ver diagrama en porcentaje determinado por el usuario.

Debe permitirle al usuario poner el diagrama en un valor en porcentaje que le sea conveniente.

Descripción: El cliente necesita que al área de edición se le pueda aplicar visualizar en diferentes porcientos, para si hay deficiencias visuales. El caso de uso del sistema asociado es Hacer Zoom. El nivel de prioridad de este requerimiento es secundario.

RF9 Selección de una porción del diagrama o el diagrama completo.

Debe permitir que se pueda seleccionar parte del diagrama o el diagrama completo así como las operaciones de copiar y cortar para ser pegado a un nuevo proyecto que cree el usuario.

Descripción: En ocasiones se necesita seleccionar el diagrama completo o parte de este para realizar distintas operaciones como copiar y cortar. El caso de uso del sistema asociado es Gestionar Diagrama. El nivel de prioridad de este requerimiento es crítico.

RF10 Mover Identificador del Módulo.

Debe permitir el usuario pueda mover el identificador del módulo.

Descripción: El cliente necesita que se pueda mover el identificador del módulo para algún lugar donde le sea más visible. El caso de uso del sistema asociado es Mover Identificador del Módulo. El nivel de prioridad de este requerimiento es opcional.

RF11 Mover Identificador de la corriente.

Debe permitir que el usuario pueda mover el identificador de la corriente.

Descripción: El cliente necesita que se pueda mover el identificador de la corriente para algún lugar donde le sea más visible. El caso de uso del sistema asociado es Mover Identificador de la Corriente. El nivel de prioridad de este requerimiento es opcional.

RF12 Gestionar Información.

Debe permitirle al usuario guardar y cargar la información.

Descripción: El cliente necesita que el sistema permita guardar toda la información que esté en el área de edición, para poder almacenar información sobre las simulaciones. En otras ocasiones necesita poder cargar información de anteriores gestiones en la aplicación para que sea consultada. Los casos de uso del sistema asociado son Guardar Información y Cargar Información. El nivel de prioridad de este requerimiento es crítico.

3.5 Requerimientos no Funcionales (RNF)

En el desarrollo del software, también existe otra categoría de requerimientos, los requerimientos no funcionales. Estos no representan qué debe hacer el sistema pero tienen igual importancia porque a través de ellos se pueden ver las diferentes características del sistema como apariencia, software, hardware, herramientas utilizadas, que están asociados. A continuación se mencionan los requerimientos no funcionales.

RNF Apariencia o interfaz gráfica:

- Utilizar botones que expresen su función.
- Idioma de la aplicación en español.
- Presentará un menú con varios elementos.
- Área de edición para editar los diagramas.

- Paleta de componentes que permitirá adicionar módulos al área de edición y la conexión entre los mismos.

RNF Software:

Para poder trabajar en la aplicación es necesario contar con la máquina virtual de java 1.6 instalada en el equipo.

RNF Hardware:

Se requiere de una computadora que tenga mouse, teclado, monitor con resolución de al menos 800x600 pixeles, además de en dependencia del sistema operativo y plataforma cumplir con las siguientes características como mínimo.

Plataforma	Versión	Memoria (RAM)	Espacio en Disco(MB)
Solaris SPARC 32 bit	Solaris 8 y 9	64 MB	29
Solaris SPARC 32 bit	Solaris 10	64MB	65
Solaris SPARC 64 bit	Solaris 8,9 y 10	64MB	53
Solaris x86 32 bit	Solaris 8,9 y 10	64MB	53
Solaris x86 64 bit	Solaris 10	64MB	53
Windows IA 32 bit	Windows XP Professional, Windows XP Home y Windows 2000 Professional	64MB	98
Windows IA 32 bit	Windows Server 2003 Web Edition, Windows Server 2003 Standard Edition y Windows Vista	128MB	98

Windows 64 bit	Windows XP, Windows Server 2003 y Windows Vista	128MB	110
Linux IA 32 bit	Suse Enterprise Linux Server 8, 9 y 10	64MB	58
Linux IA 32 bit	Enterprise Linux Desktop	64MB	58
Linux 64 bit	Suse Enterprise Linux Server 8, 9 y 10	64MB	56

Tabla 2 Requerimientos de hardware.

De esta tabla se puede sacar como conclusión de forma general que los requerimientos mínimos para una computadora personal en dependencia del sistema operativo instalado son: para Solaris 64 MB de memoria RAM y 65 MB de espacio libre en el disco duro, para Windows 128 MB de memoria RAM y 110 MB de espacio libre en el disco duro y para Linux 64 MB de memoria RAM y 58 MB de espacio libre en el disco duro.

RNF de diseño e implementación:

El sistema será una aplicación escritorio desarrollada en lenguaje de programación java, en el IDE Eclipse 3.4 por todas las características que presenta, utilizando como lenguaje de modelado el UML 2.0 y como herramienta para realizar los modelos el Visual Paradigm.

3.6 Análisis del sistema

A partir de los requisitos funcionales que fueron detectados, surgieron los casos de usos del sistema que aparecen relacionados en la Tabla 3.1 con su respectiva prioridad y a qué requerimiento funcional están asociados.

En AUP, al igual que en RUP, a partir de los casos de usos se derivan una serie de artefactos que ayudan la realización del producto final, como son los diagramas de casos de uso, diagrama de dominio, diagrama de clases del diseño, entre otros. Hay que tomar en cuenta que estos casos de usos van a guiar el proceso de realización del software.

3.7 Diagrama de Caso de Uso del Sistema.

El diagrama de caso de usos posibilita que el equipo de desarrollo del software sea capaz de comprender de una mejor manera cuáles son las necesidades del cliente, cuando esté hecho el diagrama de caso de uso será posible generar una serie de artefactos que mejorarán el proceso de desarrollo del editor gráfico.

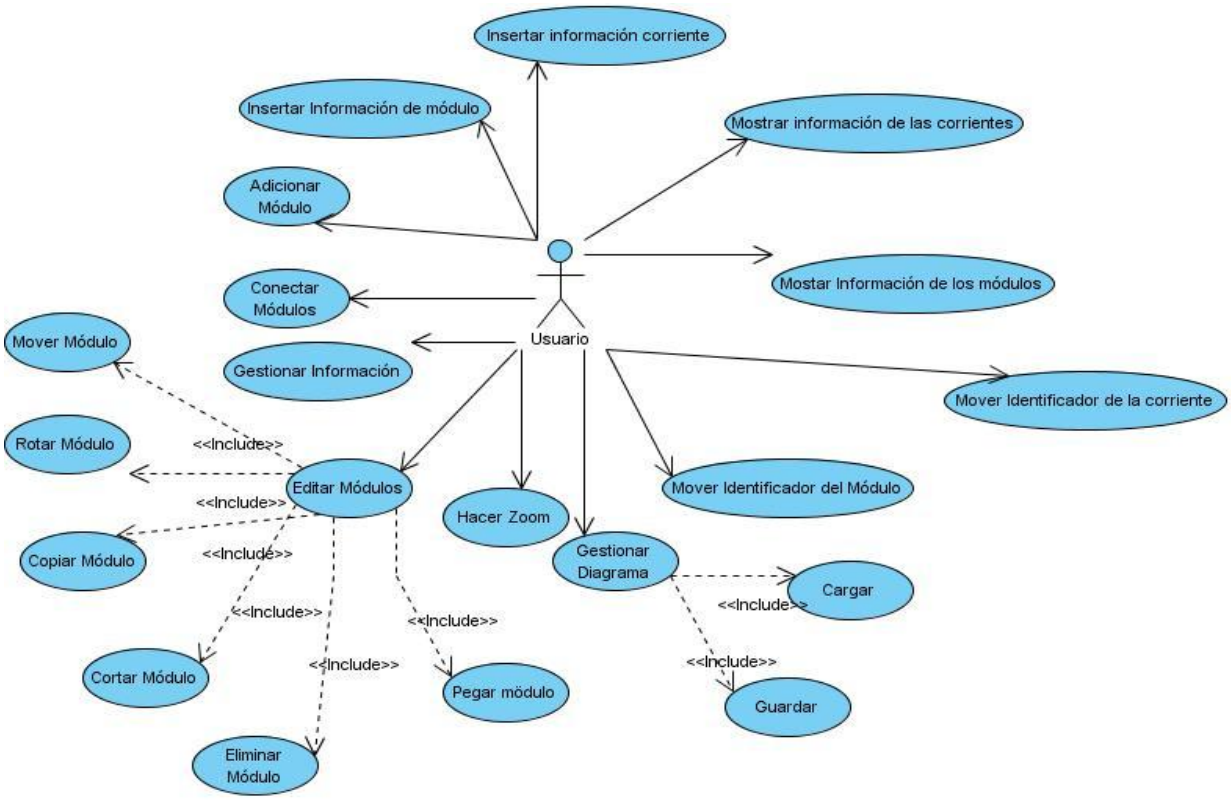


Figura 3 Diagrama de Casos de Usos del Sistema.

Actores del sistema

Actor	Justificación
Usuario	Es la persona que trabaja con el Editor Gráfico, quien va a diseñar el modelo a simular.

	Responsable de Guardar la información de la simulación.

Descripción de los casos de uso

Con la descripción de los casos de usos más significativos se puede comprender mejor como están relacionados cada uno de los casos de usos con los actores, en este caso el actor, que se encuentran envueltos en las necesidades de los clientes. A continuación se realiza la descripción de algunos de los principales casos de usos del sistema.

Caso de Uso:	Adicionar Módulo
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario oprime sobre algún módulo de la paleta.
Precondiciones:	El usuario debe haber iniciado el sistema.
Referencias	RF5
Prioridad	Crítico

Flujo Normal de Eventos	
Sección "Adicionar Módulo"	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace clic sobre algún módulo de los que se encuentran en la paleta de componentes. **2.** El usuario hace clic en el área de edición.	**3.** El sistema activa el componente y espera que el usuario de clic en el área de edición. • El sistema muestra el componente en el lugar donde el usuario hizo clic.

	• El sistema agrega el componente a una lista de componentes del área de edición.

Tabla 3 Descripción del Caso de Uso Adicionar Módulo.

Caso de Uso:	Mostrar información de los módulos.
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario hace clic dos veces sobre el módulo.
Precondiciones:	Debe existir algún módulo en el área de edición. El usuario debe haber seleccionado algún módulo.
Referencias	RF1
Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Mostrar Información"	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace clic dos veces sobre algún modulo en el área de edición.	2. El sistema busca en la lista de componentes del área de edición al módulo que se le ha dado doble clic. 3. El sistema muestra la información del módulo en un formulario.

Tabla 4 Descripción del caso de uso Mostrar Información de los Módulos.

Caso de Uso:	Rotar módulo
Actores:	Usuario
Resumen:	Caso de uso para rotar un módulo.
Precondiciones:	El usuario debe haber seleccionado algún módulo del área de edición.
Referencias	RF7
Prioridad	Crítico

Flujo Normal de Eventos	
Sección "Rotar Módulo"	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace clic derecho sobre algún módulo de los que se encuentran en área de edición. 3. El usuario selecciona la opción rotar.	**2.** El sistema muestra un grupo de opciones. **4.** El sistema rota el módulo 90 grados.

Tabla 5 Descripción del caso de uso Rotar Módulo.

Caso de Uso:	Mover Módulo
Actores:	Usuario
Resumen:	Caso de uso mover un módulo.
Precondiciones:	Debe existir algún módulo en el área de edición.
Referencias	RF7
Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Mover Módulo"	

Acción del Actor	Respuesta del sistema
1. El usuario selecciona a un módulo del área de edición	2. El sistema activa el módulo seleccionado. **3.** Espera por que el usuario mueva el módulo con el mouse presionado sobre el módulo.
4. El usuario mueve el mouse presionado sobre el módulo seleccionado hacia el área de edición que desea moverlo o bien con las flechas del teclado.	**5.** El sistema muestra el módulo donde el usuario dejo de arrastrar el mouse.

Tabla 6 Descripción del caso de uso mover módulo.

Caso de Uso:	Copiar Módulo
Actores:	Usuario
Resumen:	El usuario necesita copiar algún módulo.
Precondiciones:	Debe existir algún módulo en el área de edición. Debe estar seleccionado algún módulo.
Referencias	RF7
Prioridad	Crítico

Flujo Normal de Eventos

Sección "Copiar Módulo"

Acción del Actor	Respuesta del sistema
1. El usuario selecciona la opción Copiar. 3. El usuario selecciona la opción Copiar.	2. El sistema hace una copia del módulo seleccionado. 4. El sistema hace una copia del módulo seleccionado.

Tabla 7 Descripción del caso de uso Copiar Módulo.

Caso de Uso:	Cortar Módulo
Actores:	Usuario
Resumen:	El usuario necesita cortar algún módulo para pegarlo en otro lugar.
Precondiciones:	Debe existir algún módulo en el área de edición. Debe estar seleccionado algún módulo
Referencias	RF7
Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Cortar Módulo"	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace clic derecho sobre el módulo. 3. El usuario selecciona la opción cortar.	2. El sistema muestra un grupo de opciones. 4. El sistema hace una copia del módulo que se cortado. 5. El sistema elimina del área de edición y de la lista de componentes el módulo cortado.

Tabla 8 Descripción del caso de uso Cortar Módulo.

Caso de Uso:	Pegar Módulo
Actores:	Usuario
Resumen:	El usuario necesita pegar algún módulo.
Precondiciones:	Debe existir algún módulo en el área de edición. Debe haber sido copiado o cortado algún módulo.
Referencias	RF7

Prioridad	Crítico
Flujo Normal de Eventos	
Sección "Pegar Módulo"	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace clic derecho sobre algún lugar del área de edición. 3. El usuario selecciona la opción pegar.	2. El sistema muestra un grupo de opciones. 4. El sistema verifica que el usuario haya hecho la operación de copiar sobre algún módulo con anterioridad. 5. El sistema muestra en el área de edición el módulo que había sido copiado. 6. El sistema agrega a la lista de componentes el módulo pegado.

Tabla 9 Descripción del caso de uso Pegar Módulo.

Caso de Uso:	Eliminar módulo
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario desea eliminar un módulo.
Precondiciones:	Debe existir algún módulo en el área de edición. Debe estar seleccionado algún módulo.
Referencias	RF7
Prioridad	Crítico
Flujo Normal de Eventos	

Sección “Eliminar módulo”	
Acción del Actor	**Respuesta del Negocio**
1. El usuario hace clic derecho sobre el módulo. 3. El usuario selecciona la opción eliminar.	2. El sistema muestra un grupo de opciones. 4. El sistema elimina de la lista de componentes y del área de edición el módulo seleccionado.
Flujos Alternos	
Acción del Actor	**Respuesta del Negocio**
1. El usuario oprime la tecla “Delete”	2. El sistema elimina de la lista de componentes y del área de edición el módulo seleccionado.

Tabla 10 Descripción del caso de uso Eliminar Módulo.

Caso de Uso:	Modificar información de un módulo
Actores:	Usuario
Resumen:	Para cuando el usuario quiera insertar la información correspondiente a algún módulo.
Precondiciones:	Debe existir algún módulo en el área de edición. Debe estar abierto el formulario con la información del módulo.
Referencias	
Prioridad	Crítico
Flujo Normal de Eventos	
Sección “Modificar información de un módulo”	
Acción del Actor	**Respuesta del Negocio**
1. El usuario hace “doble clic” sobre el	2. El sistema muestra un formulario

módulo. 3. El usuario hace “clic” en el botón “Modificar Información”. 5. El usuario introduce la información en el formulario. 6. El usuario hace “clic” en el botón “Aceptar”.	con la información del módulo. 4. El sistema muestra un formulario para que el usuario modifique la información. 7. El sistema verifica que la información sea correcta. 8. El sistema modifica la información del módulo por la introducida por el usuario.
Flujos Alternos	
Acción del Actor	**Respuesta del Negocio**
5. El usuario introduce valores incorrectos. 7. El usuario cierra el mensaje de error o hace “clic” en el botón “aceptar”.	6. El sistema muestra un mensaje de error.

Tabla 11 Descripción del caso de uso Modificar Información de un Módulo.

Caso de Uso:	Crear corriente
Actores:	Usuario
Resumen:	Sucede cuando el usuario selecciona en el panel de componentes una corriente y después da un clic en dos módulos de forma consecutiva.
Precondiciones:	Deben existir al menos dos módulos en el área de edición.
Referencias	RF6
Prioridad	Crítico
Flujo Normal de Eventos	
Sección “Crear corriente”	

Acción del Actor	Respuesta del sistema
1. El usuario hace clic en el panel de componentes sobre una corriente. 4. El usuario hace clic sobre un módulo que se encuentra en el área de edición. 6. El usuario hace clic sobre otro módulo.	2. El sistema activa la corriente en el panel de componentes. 3. El sistema espera porque el usuario haga clic sobre algún módulo. 5. El sistema espera porque el usuario haga clic sobre otro módulo. 7. El sistema crea una corriente y la agrega a la lista de componentes. 8. El sistema muestra en el área de edición la corriente entre los dos módulos.

Tabla 12 Descripción del caso de uso Crear Corriente.

Caso de Uso:	Modificar información de la corriente
Actores:	Usuario
Resumen:	Sucede cuando el usuario hace “doble clic” en el área de edición sobre una corriente.
Precondiciones:	Deben existir al menos una corriente en el área de edición.
Referencias	
Prioridad	Crítico
Flujo Normal de Eventos	
Sección “Modificar información de corriente”	

Acción del Actor	Respuesta del Negocio
1. El usuario hace “doble clic” sobre	2. El sistema muestra un formulario

una corriente en el área de edición. 3. El usuario hace “clic” en el botón “Modificar Información”. 5. El usuario introduce la información en el formulario. 6. El usuario hace “clic” en el botón “Aceptar”.	con la información de la corriente. 4. El sistema muestra un formulario para que el usuario modifique la información. 7. El sistema verifica que la información sea correcta. 8. El sistema modifica la información de la corriente por la introducida por el usuario.
Flujos Alternos	
Acción del Actor	**Respuesta del Negocio**
6. El usuario introduce valores incorrectos. 8. El usuario cierra el mensaje de error o hace “clic” en el botón “aceptar”.	7. El sistema muestra un mensaje de error.

Tabla 13 Descripción del caso de uso Insertar información de la corriente.

Caso de Uso:	Mostrar información de corriente
Actores:	Usuario
Resumen:	El caso de uso se inicia cuando el usuario hace “doble clic” sobre una corriente en el área de edición.
Precondiciones:	Debe existir alguna corriente en el área de edición.
Referencias	RF2
Prioridad	Crítico
Flujo Normal de Eventos	

Sección “Crear corriente”	
Acción del Actor	**Respuesta del sistema**
1. El usuario hace “doble clic” sobre una corriente en el área de edición.	2. El sistema busca corriente que se le ha hecho la acción en área de edición. 3. El sistema muestra toda la información de la corriente en un formulario.

Tabla 14 Descripción del caso de uso Mostrar información de la corriente.

3.8 Diseño del sistema.

En la fase de diseño se logra una solución lógica de la problemática que se basa en el paradigma de orientada a objetos. Para lograr esto se modelan los diagramas de interacción, que pueden ser de dos tipos, los diagramas de colaboración y los diagramas de secuencia. Estos diagramas muestran a través de mensaje cómo interactúan los objetos para realizar cada uno de los casos de usos. Para poder realizar estos diagramas es necesario conocer los principios de asignación de responsabilidades y los patrones de diseño. Los diagramas que se utilizarán son los diagramas de secuencia. A partir de los diagramas de interacción es posible construir el diagrama de clases del diseño que será implementado.

3.8.1 Patrones para asignar responsabilidades.

Los Patrones GRASP (General Responsability Assignment Software Patterns) describen los principios fundamentales de diseño de objetos para la asignación de responsabilidades. En cuanto a la responsabilidad Booch y Rumbaugh la definen como “contrato u obligación de un tipo o clase” (20). Las responsabilidades se relacionan con las obligaciones de un objeto respecto a su comportamiento, estas responsabilidades pertenecen a las siguientes categorías:

1. Conocer
2. Hacer

Las responsabilidades de un objeto relacionadas con “conocer” son las siguientes:

1. Estar enterado de los datos privados encapsulados.

2. Estar enterado de la existencia de objetos conexos.
3. Estar enterado de cosas que se puede derivar o calcular.

Las responsabilidades de un objeto relacionadas con “hacer” son las siguientes:

1. Hacer algo en uno mismo.
2. Iniciar una acción en otros objetos.
3. Controlar y coordinar actividades en otros objetos. (21)

Se puede decir que las responsabilidades están relacionadas de forma general con algún método de la clase en cuestión. Por ejemplo una responsabilidad de tipo ”conocer” sería el conocimiento que podría tener cualquier clase sobre los datos de los atributos del objeto, una responsabilidad de tipo “hacer” sería las diferentes operaciones de modificación que podría hacer una clase sobre un atributo del objeto, como por ejemplo cambiar el valor de un atributo. (21)

A continuación se hace una descripción de los patrones GRASP que se utilizan en el diseño del sistema.

3.8.1.1 Patrón Experto.

Este patrón de diseño es uno de los más utilizados en la programación orientada a objetos. Como cada patrón tiene un problema a resolver y brinda una solución. Uno de los beneficios que brinda este patrón es que el comportamiento se distribuye entre las clases que cuentan con la información requerida, alentando con ello definiciones de clases sencillas y que son más fáciles de comprender y de mantener, brindando soporte a una alta cohesión.
Problema: ¿Cuál es el principio fundamental en virtud del cual se asignan las responsabilidades en el diseño orientado a objetos?
Solución: Asignar una responsabilidad al experto en información: la clase que cuenta con la información necesaria para cumplir la responsabilidad.

Ejemplo: En el sistema para el requerimiento de mostrar información del módulo es necesario para esto que el módulo muestre su información, por lo tanto, el módulo es el responsable de conocer la información que solo él tiene (Ver Figura 3).

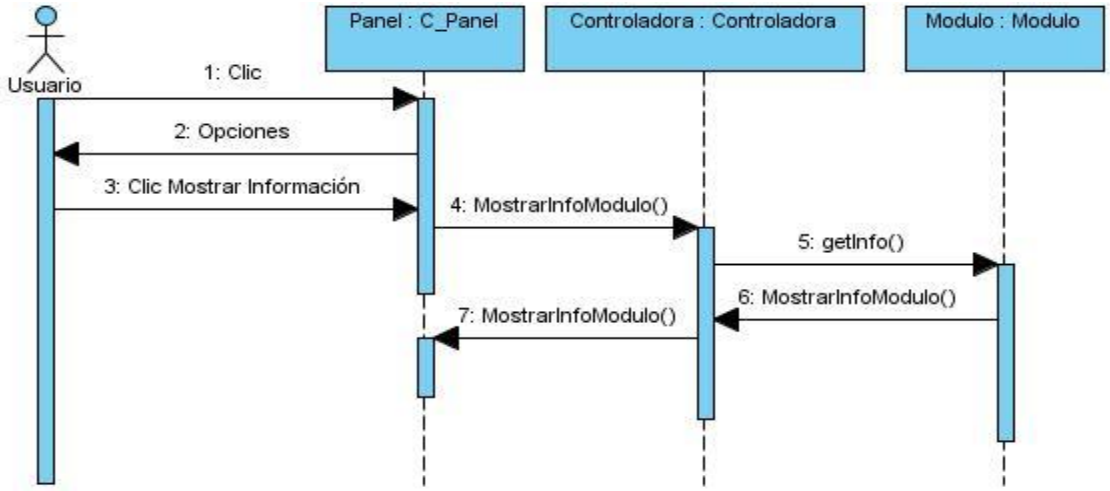


Figura 4 Diagrama de Secuencia del Caso de Uso del Sistema Mostrar Información del Módulo.

3.8.1.2 Patrón Creador.

Este patrón guía asignación de responsabilidades relacionadas con la creación de objetos, tarea muy frecuente en los sistemas orientados a objetos. El propósito fundamental de este patrón es encontrar un creador que debemos conectar con el objeto producido en cualquier evento.

En un diagrama de clases se registran las relaciones muy frecuentes entre las clases. El patrón Creador indica que la clase incluyente del contenedor o registro es idónea para asumir la responsabilidad de crear la cosa contenida o registrada. Desde luego, se trata tan solo de una directriz.

Problema: ¿Quién debería ser responsable de crear una nueva instancia de alguna clase?

Solución: Asignarle a la clase A la responsabilidad de crear una instancia de clase B en uno de los siguientes casos:

1. A agrega los objetos B.
2. A contiene los objetos B.
3. A registra las instancias de los objetos B.
4. A utiliza específicamente los objetos B.
5. A tiene los datos de inicialización que serán transmitidos a B cuando este objeto sea creado (así que A es un Experto respecto a la creación de B). (21)

Ejemplo: En el sistema la clase controladora es encargada de crear una instancia de la clase pila para poder realizar las operaciones de deshacer rehacer (Ver Figura 4).

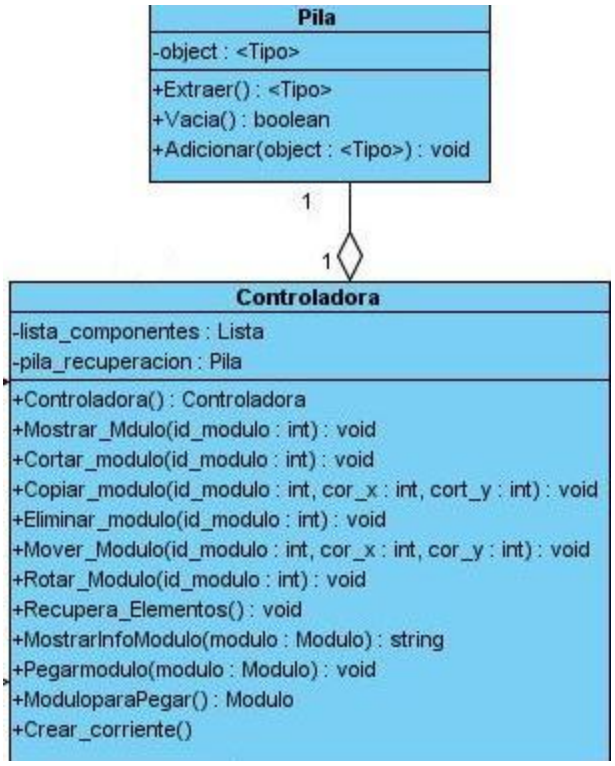


Figura 5 Relación entre la clase Controladora y la clase Pila.

3.8.1.3 Patrón Controlador

Este patrón, como su nombre lo indica es el encargado en un diseño de clase de controlar todo el flujo de información y datos en el sistema. Realiza las operaciones que tiene que hacer el sistema.

Problema: ¿Quién debería encargarse de atender un evento del sistema?

Solución: Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una clase que represente una de las siguientes opciones:

1. El "sistema" global (controlador de fachada).
2. La empresa u organización global (controlador de fachada).
3. Algo en el mundo real que es activo (por ejemplo, el papel de una persona) y que pueda participar en la tarea (controlador de tareas). (21)
4. Un manejador artificial de todos los eventos del sistema de un caso de uso, generalmente denominados "Controlador".

A continuación se muestra la clase controladora con las funcionalidades que debe cubrir (métodos).

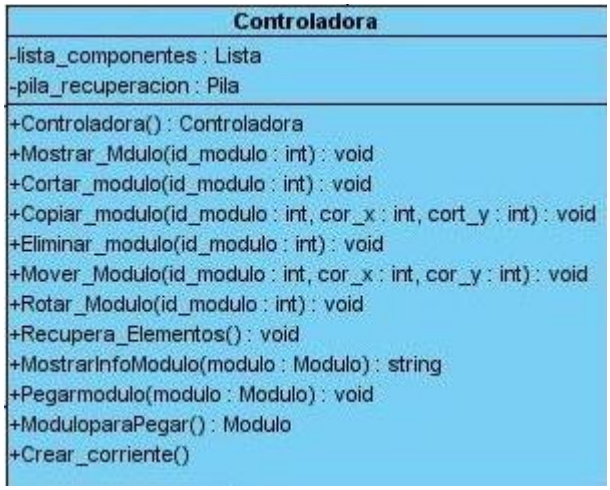


Figura 6 Clase Controladora.

3.8.1.4 Patrón Polimorfismo.

Siempre que se tenga que llevar a cabo una responsabilidad que dependa del tipo, se tiene que hacer uso del polimorfismo, cuando las alternativas o comportamientos relacionados varían según el tipo, hay que asignar la responsabilidad para el comportamiento, utilizando operaciones polimórficas, a los tipos para los que varia el comportamiento. Se le asigna el mismo nombre a servicios en diferentes objetos.

Ejemplo: En el sistema existen varios tipos de componentes gráficos. De los que se van a representar módulos, puertos y corrientes. Pero todos ellos presentan características comunes que se agrupan en una sola clase llamada Componente. A continuación una representación de las relaciones entre estas clases.

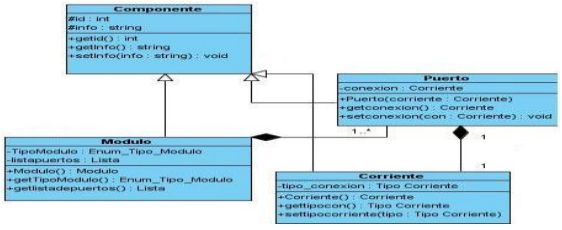


Figura 7 Relación entre la Clase Componente y sus tres clases “hijas”.

3.9 Diagramas de Secuencias.

Un diagrama de secuencia muestra la interacción de un conjunto de objetos en una aplicación a través del tiempo y se modela para cada método de la clase. Mientras que el diagrama de casos de uso permite el modelado de una vista del negocio del escenario, el diagrama de secuencia contiene detalles de implementación del escenario, incluyendo los objetos y clases que se usan para implementar el escenario, y mensajes intercambiados entre los objetos. A continuación se muestra el diagrama de secuencia del caso de uso Adicionar Módulo. Donde el usuario comienza haciendo clic en la paleta de componentes, seguidamente da clic en el panel de edición, en ese momento es creado el módulo y mostrado mientras es almacenado en una lista de la clase controladora.

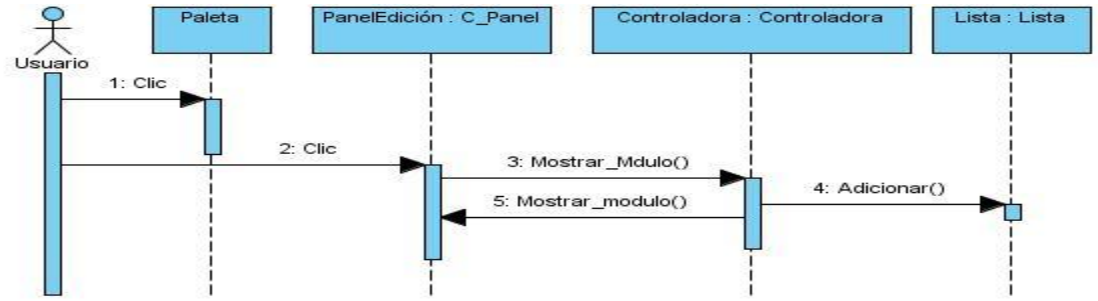


Figura 8 Diagrama de Secuencia Adicionar Módulo.

3.10 Diagrama de Clases del diseño

Un diagrama de clases es un tipo de diagrama estático que describe la estructura de un sistema mostrando sus clases, atributos y las relaciones entre ellos. Los diagramas de clases son utilizados durante el proceso diseño de los sistemas, donde se crea el diseño conceptual de la información que se manejará en el sistema, y los componentes que se encargarán del funcionamiento y la relación entre uno y otro.

Cuando ya se han realizado los diagramas de secuencias se pueden extraer de ellos las clases con cada una de las funcionalidades que debe cumplir el sistema, se debe ser capaz de realizar el diagrama de clases del diseño del sistema.

A continuación se muestra dicho diagrama:

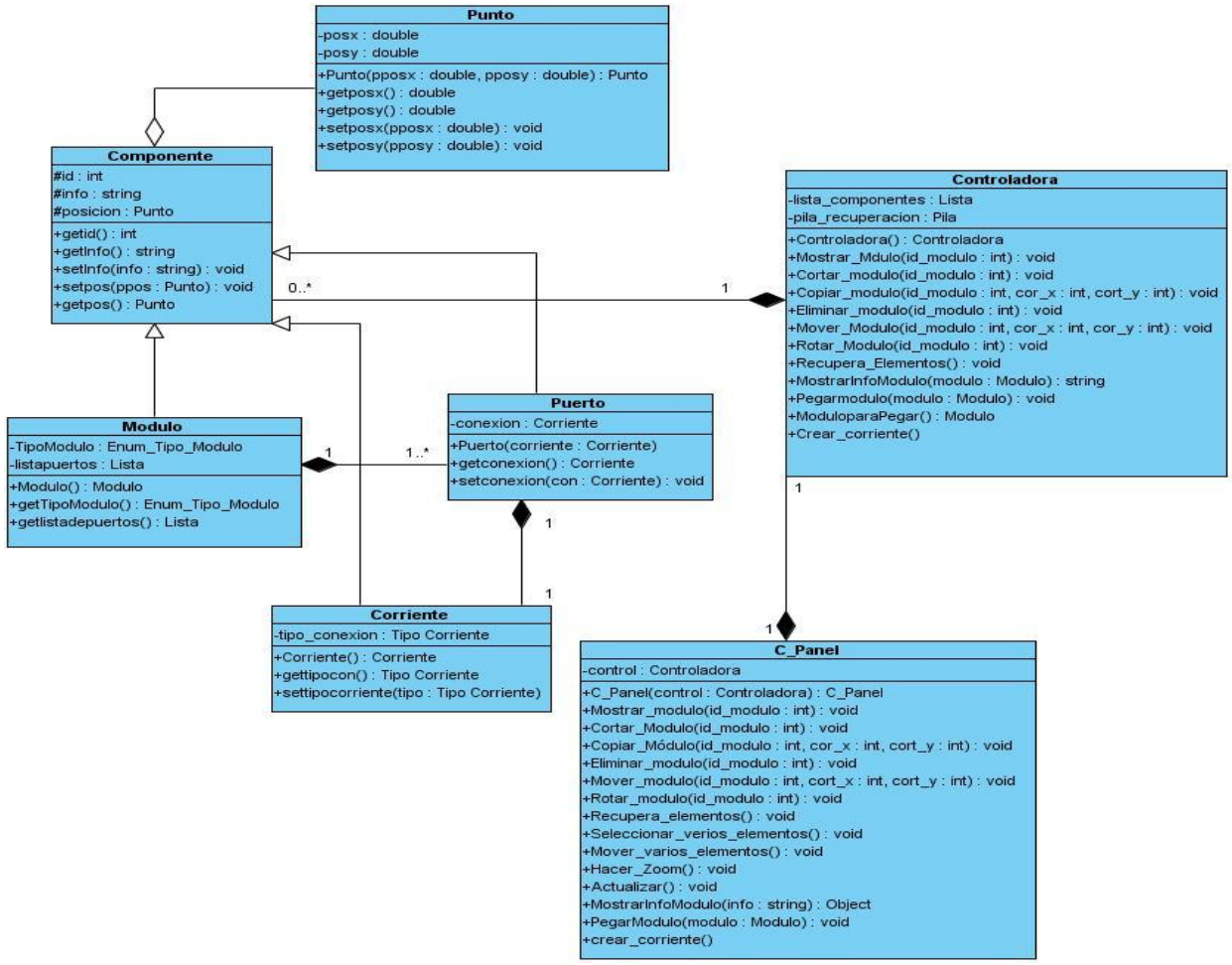


Figura 9 Diagrama de Clases del diseño

Patrón Arquitectónico

Para poder realizar el diseño de la aplicación se utiliza un patrón de diseño de la categoría arquitectónico, el Modelo Vista Controlador, el cuál es descrito a continuación.

Como todo patrón, este presenta un problema y una solución.
Problema: Conviene desacoplar los objetos dominio, las vistas y los manejadores, a fin de brindar soporte a una mayor reutilización de los objetos dominio y reducir al mínimo el impacto que los cambios de la interfaz y los manejadores tienen en ellos. ¿Qué hacer?

Solución:

Definir las clases dominio para que tengan el menor acoplamiento y visibilidad directa respecto a las clases de las vistas y para que los datos de la aplicación y de la funcionalidad se conserven en las clases de dominio, no en las de ventana. Definir las clases manejadores para que procesen los eventos al sistema y re direccionen a las clases dominio y ventana tanto el procesamiento como la visualización de resultados respectivamente.

Se puede decir sobre este patrón arquitectónico que se han desarrollado tres variantes desde que fue presentado a la comunidad científica. La primera variante que se ha presentado en el desarrollo del software a nivel mundial es en la cual no existe ninguna comunicación entre el Modelo y la Vista y esta última recibe los datos a mostrar a través del Controlador. La segunda variante en cambio es algo flexible, disminuyendo las responsabilidades del controlador desarrollando una comunicación entre el Modelo y la Vista, debido a que esta última mostrará los datos buscándolos directamente en el Modelo. Por último, en la tercera variante se diversifica las funcionalidades del Modelo teniendo en cuanta las características de las aplicaciones multimedia, donde tienen un gran peso las medias utilizadas en estas. En el desarrollo del presente software se utiliza la segunda variante.

3.11 Conclusiones parciales

Con la ejecución de los distintos artefactos como el diagrama de caso de usos del sistema, **la** descripción de los mismos, así como el modelo de clases del diseño, además de haber detectado los distintos patrones de diseño a utilizar y el patrón arquitectónico definido se podrá proceder a realizar la implementación y prueba del editor gráfico para la plataforma de desarrollo de simuladores de procesos químicos.

CAPÍTULO 4: Implementación y prueba

Introducción

Este capítulo estará basado en los artefactos que se generan en las fases de implementación y pruebas. En la fase de implementación del ciclo de desarrollo basado en AUP se realizan los modelos de implementación, los cuáles modelan el hardware utilizado para la implementación del sistema, así como las dependencias lógicas entre componentes software. Los diagramas que modelan el hardware son conocidos como diagramas de despliegues, mientras que los diagramas que modelan las dependencias entre los componentes son los diagramas de componentes. A continuación se hace una descripción de los modelos de implementación.

4.1 Diagrama de Despliegue.

La aplicación que se ha desarrollado está basada en que se utilizará en una computadora personal que tendrá instalada la máquina virtual de java y contará de conexión por puerto usb a una impresora para que se impriman los resultados de las simulaciones. A continuación se muestra el diagrama de despligue.

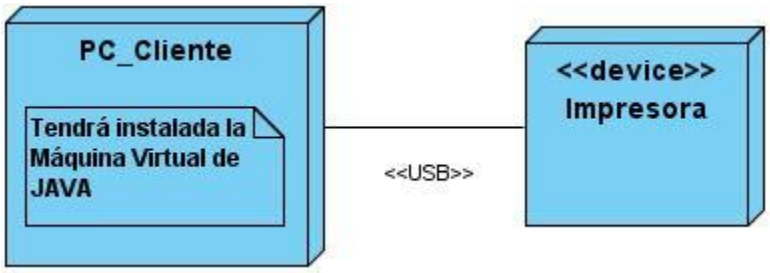


Figura 10 Diagrama de Despliegue

4.2 Diagrama de Componentes

Como se ha mencionado con anterioridad el diagrama de componentes representa las dependencias que existen entre los diferentes componentes. En el siguiente diagrama se puede vislumbrar que el sistema está estructurado de forma tal que utiliza el patrón modelo vista controlador.

En el modelo se cuenta con tres componentes, los cuales utilizan una librería que pertenece a Eclipse. En el control se cuenta con un componente, el cuál esta relacionado con uno de los componentes del modelo y utiliza dos librerías pertenecientes a java. Por último en la capa visual hay un componente que está relacionado de forma directa al componente que forma parte de la capa control y utiliza cuatro librerías para su desarrollo, dos de la eclipse, una de java y la otra de javax. A continuación el diagrama de componentes.

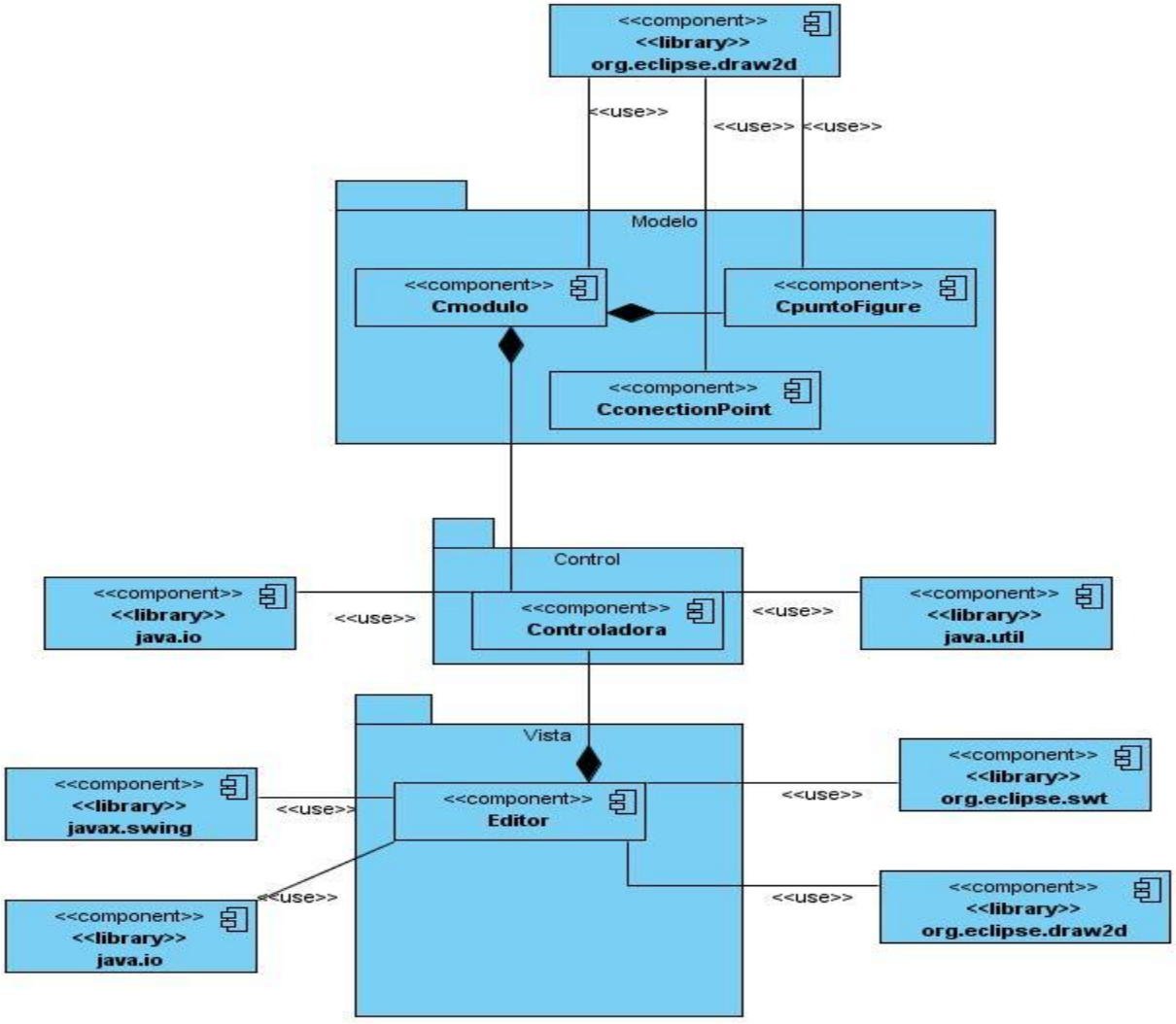


Figura 11 Diagrama de Componentes

4.3 Pruebas

El desarrollo de sistemas de software implica una serie de actividades de producción en las que las posibilidades de que aparezca el fallo humano son enormes. Los errores pueden empezar a darse desde el primer momento del proceso, en el que los objetivos pueden estar especificados de forma errónea o imperfecta, así como en posteriores pasos de diseño y desarrollo, debido a la imposibilidad humana para trabajar y comunicarse de forma perfecta, el desarrollo de software a de ir acompañado de una actividad que garantice la calidad.

4.3.1 Objetivos de las pruebas.

El objetivo de la etapa de pruebas es garantizar la calidad del producto desarrollado. Esto implica:

1. La prueba es el proceso de ejecución de un programa con la intención de descubrir un error. (9)
2. Un buen caso de prueba es aquel que tiene una alta probabilidad de mostrar un error no descubierto hasta entonces. (9)
3. Una prueba tiene éxito si descubre un erro no detectado hasta el momento. (9)
4. Verificar la interacción de componentes.
5. Verificar la integración adecuada de los componentes.
6. Verificar que todos los requisitos se han implementado correctamente.
7. Identificar y asegurar que los defectos encontrados se han corregido antes de entregar el software al cliente.
8. Diseñar pruebas que sistemáticamente saquen a la luz diferentes clases de errores, haciéndolo con la menor cantidad de tiempo y esfuerzo.

4.3.2 Pruebas unitarias

Las pruebas unitarias son una forma de probar el correcto funcionamiento de un módulo de código. Esto sirve para asegurar que cada uno de los módulos funcione correctamente por separado. Luego, con las Pruebas de Integración, se podrá asegurar el correcto funcionamiento del sistema o subsistema en cuestión.

El objetivo de las pruebas unitarias es aislar cada parte del programa y mostrar que las partes individuales son correctas. Proporcionan un contrato escrito que el trozo de código debe satisfacer. Estas pruebas aisladas proporcionan cinco ventajas básicas:

Fomentan el cambio: facilitan que el programador cambie el código para mejorar su estructura, puesto que permiten hacer pruebas sobre los cambios y así asegurarse de que los nuevos cambios no han introducido errores.

Simplifica la integración: permiten llegar a la fase de integración con un grado alto de seguridad de que el código está funcionando correctamente.

Documenta el código: Las propias pruebas son documentación del código puesto que ahí se puede ver cómo utilizarlo.

Separación de la interfaz y la implementación: Dado que la única interacción entre los casos de prueba y las unidades bajo prueba son las interfaces de estas últimas, se puede cambiar cualquiera de los dos sin afectar al otro.

Los errores están más acotados y son más fáciles de localizar: dado que se tienen pruebas unitarias que pueden desenmascararlos.

4.3.3 Pruebas de caja blanca

Las pruebas de caja blanca se realizan sobre las funciones internas de un módulo en concreto, están dirigidas a las funciones internas. Entre las técnicas usadas se encuentran; la cobertura de caminos, pruebas sobre las expresiones lógico-aritméticas, pruebas de camino de datos, comprobación de bucles.

En la siguiente tabla se representa las pruebas de Caja Blanca.

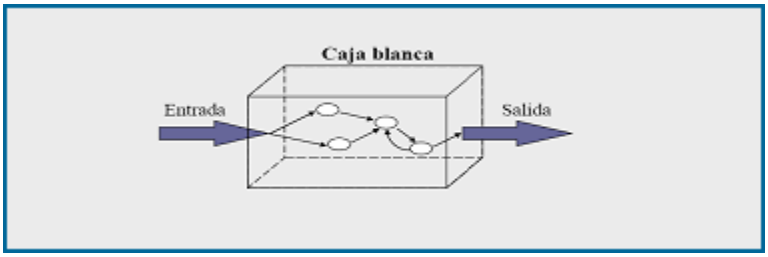


Figura 12 Representación de pruebas de Caja Blanca

Tipos de prueba de caja blanca:

Prueba de Condición: Es un método de diseño de casos de prueba que ejercita las condiciones lógicas contenidas en el módulo de un programa.

Prueba de Flujo de Datos: Se seleccionan caminos de prueba de un programa de acuerdo con la ubicación de las definiciones y los usos de las variables del programa.

Prueba de Bucles: Es una técnica de prueba de caja blanca que se centra exclusivamente en la validez de las construcciones de bucles.

Prueba del Camino Básico: Esta técnica permite obtener una medida de la complejidad lógica de un diseño y usar esta medida como guía para la definición de un conjunto básico. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes por los cuales puede circular el flujo de control. Para obtener dicho conjunto de caminos

independientes se construye el Grafo de Flujo asociado y se calcula su complejidad ciclomática.

La técnica del camino básico permite obtener una medida de la complejidad lógica del código de cada método, programa o módulo dado. La idea es derivar casos de prueba a partir de un conjunto dado de caminos independientes que existen en la codificación por los cuales puede circular el flujo de control. Es además una de las más eficientes en cuanto a cobertura de código, pues logra que se ejecuten todos los bucles en sus límites operacionales. (10)

Los pasos que se siguen para aplicar esta técnica son:

1. A partir del diseño o del código fuente, se dibuja el grafo de flujo asociado.
2. Se calcula la complejidad ciclomática del grafo.
3. Se determina un conjunto básico de caminos independientes.
4. Se preparan los casos de prueba que obliguen a la ejecución de cada camino del conjunto básico.

Los casos de prueba obtenidos del conjunto básico garantizan que durante la prueba se ejecuta por lo menos una vez cada sentencia del programa.

Para aplicar la técnica del camino básico se debe introducir la notación para la representación del flujo de control, este puede representarse por un grafo de flujo en el cual:

1. Cada nodo del grafo corresponde a una o más sentencias de código fuente.
2. Todo segmento de código de cualquier programa se puede traducir a un grafo de flujo.
3. Se calcula la complejidad ciclomática del grafo.

Para construir el grafo se debe tener en cuenta la notación para las instrucciones.

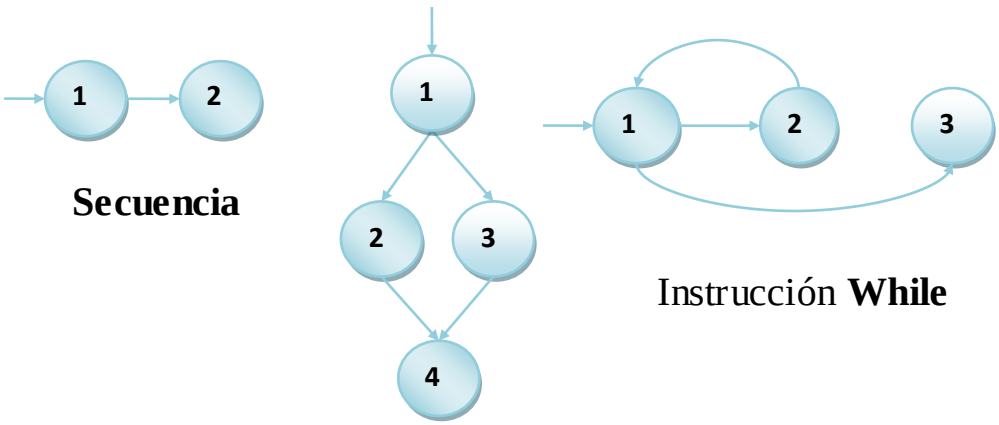


Figura 13 Notación de grafos de flujo para las instrucciones: Secuenciales, If, While

Un grafo de flujo está formado por 3 componentes fundamentales que ayudan a su elaboración, su comprensión y brindan información para confirmar que el trabajo se está haciendo adecuadamente.

Componentes del grafo de flujo:

Nodo: son los círculos representados en el grafo de flujo, el cual representa una o más secuencias del procedimiento, donde un nodo corresponde a una secuencia de procesos o a una sentencia de decisión. Los nodos que no están asociados se utilizan al inicio y final del grafo.

Aristas: son constituidas por las flechas del grafo, son iguales a las representadas en un diagrama de flujo y constituyen el flujo de control del procedimiento. Las aristas terminan en un nodo, aun cuando el nodo no representa la sentencia de un procedimiento

Figura 14 Notación de grafos de flujo para la instrucción Case

Regiones: son las áreas delimitadas por las aristas y nodos donde se incluye el área exterior del grafo, como una región más. Las regiones se enumeran siendo la cantidad de regiones equivalente a la cantidad de caminos independientes del conjunto básico de un procedimiento.

Para realizar la prueba de caja blanca específicamente la prueba del camino básico es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a analizar.

Cálculo de la complejidad ciclomática

La complejidad ciclomática es una métrica de software extremadamente útil pues proporciona una medición cuantitativa de la complejidad lógica de un programa. El valor calculado como complejidad ciclomática define el número de caminos independientes del conjunto básico de un programa y da un límite superior para el número de pruebas que se deben realizar para asegurar que se ejecute cada sentencia al menos una vez.

Para efectuar el cálculo de la complejidad ciclomática del código es necesario tener varios parámetros como son la cantidad total de aristas del grafo, cantidad total de nodos para la siguiente fórmula:

V (G) = (A– N) + 2

Siendo “A” la cantidad total de aristas y “N” la cantidad total de nodos.

Se puede usar también:

V (G) = P + 1

Siendo “P” la cantidad total de nodos predicados (son los nodos de los cuales parten dos o más aristas).

V (G) = R

Siendo “R” la cantidad total de regiones, para cada formula “V (G)” representa el valor del cálculo.

Después de haber extraído los caminos básicos del flujo, se procede a ejecutar los casos de pruebas para este procedimiento, se debe realizar al menos un caso de prueba por cada camino básico.

Para realizarlos es necesario cumplir con las siguientes exigencias:

1. **Descripción:** Se hace la entrada de datos necesaria, validando que ningún parámetro obligatorio pase nulo al procedimiento o no se entre algún dato erróneo.
2. **Condición de ejecución:** Se especifica cada parámetro para que cumpla una condición deseada para ver el funcionamiento del procedimiento.
3. **Entrada:** Se muestran los parámetros que entran al procedimiento.
4. **Resultados Esperados:** Se expone resultado que se espera que devuelva el procedimiento.

Para realizar la prueba de caja blanca específicamente la prueba del camino básico es necesario calcular antes la complejidad ciclomática del algoritmo o fragmento de código a

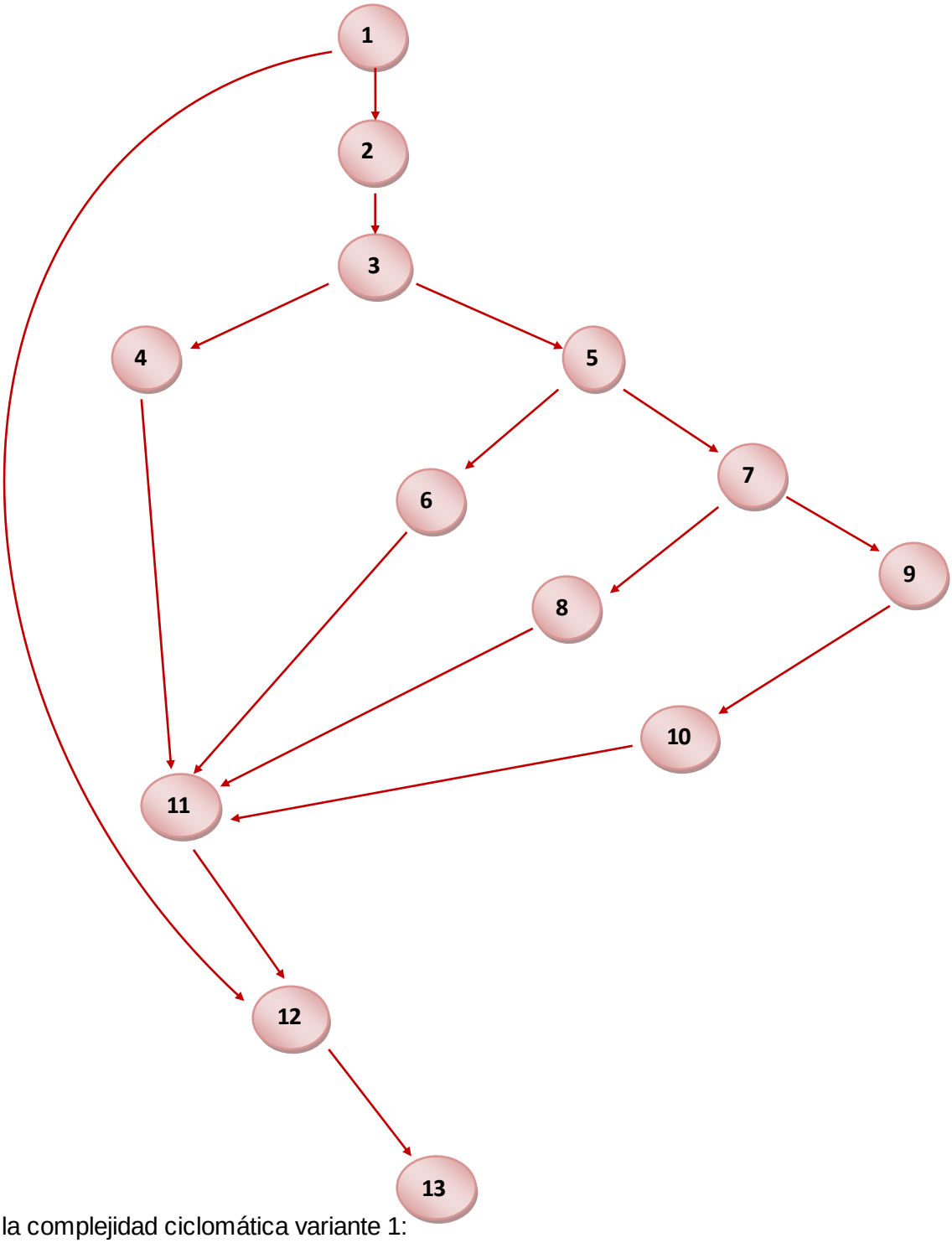
analizar. A continuación se enumera las sentencias de código del procedimiento realizado sobre el método mouseDoubleClick(MouseEevent e) el cual es el encargado de mandar a construir los objetos tipos módulos y visualizarlos.

.

```
public void mouseDoubleClick(MouseEvent e) {

if(addmodulo)//1
    {
    Figure fig=null;//2
switch(idmodulo)//2
{
case 1://3
    fig=editor.figure(display,1);//4
contentsLayout.setConstraint(fig, new Rectangle(e.x,e.y,42,42));//4
contents.setLayoutManager(contentsLayout);//4
contents.add(fig);//4
idmodulo++;//4
break;//4

case 2://5

    fig=editor.figure(display,2);//6
    contentsLayout.setConstraint(fig, new Rectangle(e.x,e.y,44,64));//6
    contents.setLayoutManager(contentsLayout);//6
    contents.add(fig);//6
    idmodulo++;//6
break;//6

case 3://7
    fig=editor.figure(display,3);//8
    contentsLayout.setConstraint(fig, new Rectangle(e.x,e.y,50,50));//8
    contents.setLayoutManager(contentsLayout);//8
    contents.add(fig);//8
    idmodulo++;//8
    break;//8

case 4://9
    fig=editor.figure(display,4);//10
    contentsLayout.setConstraint(fig, new Rectangle(e.x,e.y,64,80));//10
    contents.setLayoutManager(contentsLayout);//10
    contents.add(fig);//10

    break;//10
}//11
    lws.setContents(contents);//11
    ListaModulos.addLast(fig);//11
}//12

}//13
```

Figura 15 Código de la función mouseDoubleClick(MouseEvent e).

En la siguiente figura se muestra el grafo de flujo asociado al código anterior.



Cálculo de la complejidad ciclomática variante 1:

V (G) = (A– N) + 2

V (G) = (16 – 13) + 2

V (G) = 5

Cálculo de la complejidad ciclomática variante 2:

V (G) = P + 1

V (G) = 4 + 1

V (G) = 5

Cálculo de la complejidad ciclomática variante 2:

V (G) = R

V (G) = 5

El cálculo efectuado mediante las tres fórmulas ha dado el mismo valor, por lo que se puede plantear que la complejidad ciclomática del código es de 5, lo que significa que existen cinco posibles caminos por donde el flujo puede circular, este valor representa el límite mínimo del número total de casos de pruebas para el procedimiento tratado.

Seguidamente es necesario representar los caminos básicos por los que puede recorrer el flujo. En estas representaciones se subrayan los elementos de cada camino que los hacen independientes a los demás.

Númer	Caminos básicos
1	1–12 – 13
2	1– 2 – 3 – 4 – 11 – 12 – 13
3	1 – 2 – 3 – 5 – 6 – 11 – 12 – 13
4	1 – 2 – 3 – 5 – 7 – 8 – 11 – 12 – 13
5	1 – 2 – 3 – 5 – 7 – 9 – 10 – 11 – 12 – 13

Tabla 15Caminos básicos

Caso de prueba para el camino básico 5:

Camino 5: [1 – 2 – 3 – 5 – 7 – 9 – 10 – 11 – 12 – 13]

Descripción:

Los datos de entrada cumplirán con los siguientes requisitos:

El parámetro tomará las coordenadas en las que se encuentra situado el mouse en el área de edición del editor, los valores de addmodulo e idmodulo serán constantes para la prueba de este camino.

Condición de ejecución:

1. El mouse se encuentra en el área de edición del editor.
2. Se ha seleccionado previamente el módulo que se desea adicionar al área de edición.
3. La variable de tipo bool addmodulo ha tomado un valor verdadero.
4. Se ejecuta un doubleclick en el área de edición.

Entrada:

addmodulo=true.

Idmodulo=4.

Resultados esperados:

Se espera que se adicione al área de edición del editor el módulo cuyo id es cuatro.

4.4 Conclusiones parciales

En este capítulo se mostraron los artefactos generados en el flujo de trabajo de Implementación del sistema, específicamente el diagrama de componentes que brinda una vista de la relaciones de dependencia que se establecen entre los elementos de implementación. Se mostró y explicó detalladamente una sección de código fuente de la aplicación, se escogió para ello una sección que desempeña una funcionalidad crítica dentro del sistema para lograr un entendimiento de cómo el sistema opera para adicionar módulos, requisito para el cual fue desarrollado. Luego se propuso y mostró la estrategia de pruebas realizada una vez concluida la implementación para garantizar que el funcionamiento del sistema es el esperado. En dichas pruebas se obtuvieron resultados satisfactorios.

Conclusiones

Con el desarrollo de la propuesta de solución se logro realizar un editor gráfico para la plataforma de desarrollo de simuladores de procesos químicos, esto fue gracias a la utilización de una metodología de desarrollo de software como AUP que permitió realizar el software de una forma ágil. Se ha documentado el proceso de desarrollo de software, lo que permitirá una mayor comprensión del editor gráfico.

Con el editor gráfico del simulador de procesos químicos se dará un paso de avance en los simuladores cubanos porque la mayoría de ellos no cuentan con funcionalidades gráficas que ayuden a comprender los procesos de simulación.

Referencias bibliográficas

Trabajos citados

1. **Scnna, Nicolás José.** *MODELADO, SIMULACIÓN Y OPTIMIZACIÓN DE PROCESOS QUÍMICOS.*

2. **Hernández León, Rolando Alfredo y Coello González, Sayda.** *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA .* Ciudad de la Habana : s.n., 2002.

3. **Shannon, Robert y Johannes, James D.** Systems simulation: the art and science. s.l. : IEEE Transactions on Systems, Man and Cybernetics., 1976, Vol. 6(10), págs. 723-724.

4. **Peña, Msc.Pierre.** *Teoria de simuladores.*

5. **Pérez de Alejo Vitoria, DrC.Hector Eugenio y Hernández León, DrC. Rolando Alfredo.** *Informatización de la ingeniería de proceso en la industria química cubana.* Ciudad de la Habana : s.n., 2005.

6. **Corrales Valdés, Yurisnel y Suárez López, Juan Carlos.** *Desarrollo de la arquitectura del Sistema para la Simulación de Procesos en la Industria Química Cubana.* Ciudad de La Habana : s.n., 2007.

7. **Geónova, Gonzalo, Fuentes, José M y Llorens, Juan.** *Evaluacion de herramientas CASE para UML.*

8. *Manual del Visual Paradigm.*

9. **Pressman, Roger S.** *Ingeniería de Software.* 2002.

10. **Márques Alpízar, Yaimí y Valdez Echavarrí, yenni.** *Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico.* 2008.

11. **Zukowski, John.** *Programación Java 2 J2SE 1.4.* 2003.

12. **Naylor, Thomas.** *"Técnicas de simulación en computadoras ".* 1996.

13. **Mellor, S. J., y otros.** *MDA Distilled. Principles of Model-Driven Architecture.* s.l. : Addison-Wesley, 2004.

14. **Meléndrez., Edelsys Hernández.** [En línea] Escuela Nacional de Salud Publica, 2006. http://eva.uci.cu/file.php/63/Bibliografia_del_tema_2/Como_escribir_una_tesis.pdf.

15. **Luoma, J., Kelly, S., Tolvanen, J.-P.** *"Defining Domain-Specific Modeling Languages: Collected Experience".* s.l. : OOPSLA Workshop on DSLs, 2004.

16. **León, DrC. Hector Eugenio Pérez de AlejoVitoria y DrC. Rolando Alfredo Hernández.** *Informatización de la ingeniería de proceso en la industria química cubana.* s.l. : Universidad de las Ciencias Informáticas, 2005.

17. **Figueroa, Pablo.** *Desarrollo de Software por Aspectos.*

18. **Díaz, Dra. Rosa María Lam.** [En línea] Instituto de Hepatología e Inmunología. http://eva.uci.cu/file.php/63/Bibliografia_del_tema_2/Como_escribir_un_proyecto_de_investigacion.pdf.

19. **Booch, G., Rumbaugh, J. y Jacobson, I.** *El Lenguaje Unificado de Modelado.* s.l. : Addison-Wesley, 1999.

20. **Universidad Autónoma de Guadalajara.** [En línea] http://www.uag.mx/66/proceso1.htm.

21. **Universidad de las Ciencias Informaticas.** *Propuesta de Guía para la presentación del Trabajo de Diploma.* 2006.

22. **1.** *METODOLOGÍA DE DESARROLLO DE SOFTWARE BASADA EN EL PARADIGMA GENERATIVO. REALIZACIÓN MEDIANTE LA TRANSFORMACIÓN DE EJEMPLARES.*

23. **DISEÑO GRÁFICO Y COMUNICACIÓN VISUAL UNIVERSIDAD DE LA LAGUNA TENERIFE .** *INTERFAZ GRÁFICA DE USUARIO Aproximación semiótica y cognitiva.* 2006.

24. **Universidad de las Ciencias Informáticas.** Conferencia 2 metodología de investigación. [En línea] 2008-2009. http://eva.uci.cu/file.php/63/Materiales_de_Consulta_y_Conferencias_-_Tema_No._1/Conferencia_N_2_MI.pdf.

25. [En línea] http://www.lsi.us.es/docencia/asignaturas/dihm/tema1/tema1.html.

26. [En línea] MDA Guide Version 1.0.1, 12 de June de 2003. http://www.omg.org/mda/.

27. [En línea] http://www.dosideas.com/actualidad/487-los-lenguajes-especificos-de-dominio.htm.

28. [En línea] http://users.dsic.upv.es/asignaturas/facultad/pdd/prd.html..

29. [En línea] http://webdia.cem.itesm.mx/ac/rtrejo/Interfaz/index.html.

30. [En línea] http://www.bayesinf.com/spanish/product/forphone/help/4inteelem/contens.htm.

31. [En línea] http://pla-netx.com/linebackn/guis/guitimeline.html.

32. [En línea] http://groucho.siggraph.org.mx/boletin/Ene99/index.htm.

33. **Pérez de Alejo Vitoria, Hector Eugenio y Hernández León, Rolando Alfredo.** *Informatización de la ingeniería de proceso en la industria química cubana.*

34. **Solis Alvares, Camilo Javier y Figueroa Dias, Roberth Gustavo.** *Metodologias tradicionales vs mrtodologias agiles.*

35. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *Proceso Unificado De Desarrollo De Software.* 2004.

36. **Canós, Jose, Letelier, Patricio y Penadés, Carmen.** *Metodologias Agiles en el Desarrollo del Software.*

37. **Passerini, Ing. Nicolás y Brey, Ing. Gustavo A.** *Metodologías Iterativas de Desarrollo.* Buenos Aires : s.n., 2005.

38. **Louden, Kenneth C.** *Lenguajes de Programacion.*

39. **Manzanedo del Campo, Miguel Ángel y Cruzado Nuño, Ignacio.** *Guía de iniciación al lenguaje Java.* 1999.

40. **Queiruga, lic. Claudia y Fava, lic. Laura.** *Herramientas de desarrollo para JAVA enSoftware Libre.*

41. *Paradigma_Visual_para_UML.* [En línea] [Citado el: 20 de 2 de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(MÍ)_14720_p/.

42. *AUP.* [En línea] [Citado el: 17 de 2 de 2010.] http://cgi.una.ac.cr/AUP/.

43. *visual-paradigm.* [En línea] [Citado el: 18 de 1 de 2010.] http://www.visual-paradigm.com.

44. **Booch, G., Jacobson, I. y Rumbaugh, J.** *The UML specification documents.* Santa Clara : s.n., 1997.

45. **Larman, Craig.** *UML y Patrones Introducción al análisis y diseño orientado a objeto.*

Bibliografía consultada

Bibliografía

1. **Scnna, Nicolás José.** *MODELADO, SIMULACIÓN Y OPTIMIZACIÓN DE PROCESOS QUÍMICOS.*

2. **Hernández León, Rolando Alfredo y Coello González, Sayda.** *EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTIFICA .* Ciudad de la Habana : s.n., 2002.

3. **Shannon, Robert y Johannes, James D.** Systems simulation: the art and science. s.l. : IEEE Transactions on Systems, Man and Cybernetics., 1976, Vol. 6(10), págs. 723-724.

4. **Peña, Msc.Pierre.** *Teoria de simuladores.*

5. **Pérez de Alejo Vitoria, DrC.Hector Eugenio y Hernández León, DrC. Rolando Alfredo.** *Informatización de la ingeniería de proceso en la industria química cubana.* Ciudad de la Habana : s.n., 2005.

6. **Corrales Valdés, Yurisnel y Suárez López, Juan Carlos.** *Desarrollo de la arquitectura del Sistema para la Simulación de Procesos en la Industria Química Cubana.* Ciudad de La Habana : s.n., 2007.

7. **Geónova, Gonzalo, Fuentes, José M y Llorens, Juan.** *Evaluacion de herramientas CASE para UML.*

8. *Manual del Visual Paradigm.*

9. **Pressman, Roger S.** *Ingeniería de Software.* 2002.

10. **Márques Alpízar, Yaimí y Valdez Echavarrí, yenni.** *Procedimiento general de pruebas de caja blanca aplicando la técnica del camino básico.* 2008.

11. **Zukowski, John.** *Programación Java 2 J2SE 1.4.* 2003.

12. **Naylor, Thomas.** *"Técnicas de simulación en computadoras ".* 1996.

13. **Mellor, S. J., y otros.** *MDA Distilled. Principles of Model-Driven Architecture.* s.l. : Addison-Wesley, 2004.

14. **Meléndrez., Edelsys Hernández.** [En línea] Escuela Nacional de Salud Publica, 2006. http://eva.uci.cu/file.php/63/Bibliografia_del_tema_2/Como_escribir_una_tesis.pdf.

15. **Luoma, J., Kelly, S., Tolvanen, J.-P.** *"Defining Domain-Specific Modeling Languages: Collected Experience".* s.l. : OOPSLA Workshop on DSLs, 2004.

16. **León, DrC. Hector Eugenio Pérez de AlejoVitoria y DrC. Rolando Alfredo Hernández.** *Informatización de la ingeniería de proceso en la industria química cubana.* s.l. : Universidad de las Ciencias Informáticas, 2005.

17. **Figueroa, Pablo.** *Desarrollo de Software por Aspectos.*

18. **Díaz, Dra. Rosa María Lam.** [En línea] Instituto de Hepatología e Inmunología. http://eva.uci.cu/file.php/63/Bibliografia_del_tema_2/Como_escribir_un_proyecto_de_investigacion.pdf.

19. **Booch, G., Rumbaugh, J. y Jacobson, I.** *El Lenguaje Unificado de Modelado.* s.l. : Addison-Wesley, 1999.

20. **Universidad Autónoma de Guadalajara.** [En línea] http://www.uag.mx/66/proceso1.htm.

21. **Universidad de las Ciencias Informaticas.** *Propuesta de Guía para la presentación del Trabajo de Diploma.* 2006.

22. **1.** *METODOLOGÍA DE DESARROLLO DE SOFTWARE BASADA EN EL PARADIGMA GENERATIVO. REALIZACIÓN MEDIANTE LA TRANSFORMACIÓN DE EJEMPLARES.*

23. **DISEÑO GRÁFICO Y COMUNICACIÓN VISUAL UNIVERSIDAD DE LA LAGUNA TENERIFE .** *INTERFAZ GRÁFICA DE USUARIO Aproximación semiótica y cognitiva.* 2006.

24. **Universidad de las Ciencias Informáticas.** Conferencia 2 metodología de investigación. [En línea] 2008-2009. http://eva.uci.cu/file.php/63/Materiales_de_Consulta_y_Conferencias_-_Tema_No._1/Conferencia_N_2_MI.pdf.

25. [En línea] http://www.lsi.us.es/docencia/asignaturas/dihm/tema1/tema1.html.

26. [En línea] MDA Guide Version 1.0.1, 12 de June de 2003. http://www.omg.org/mda/.

27. [En línea] http://www.dosideas.com/actualidad/487-los-lenguajes-especificos-de-dominio.htm.

28. [En línea] http://users.dsic.upv.es/asignaturas/facultad/pdd/prd.html..

29. [En línea] http://webdia.cem.itesm.mx/ac/rtrejo/Interfaz/index.html.

30. [En línea] http://www.bayesinf.com/spanish/product/forphone/help/4inteelem/contens.htm.

31. [En línea] http://pla-netx.com/linebackn/guis/guitimeline.html.

32. [En línea] http://groucho.siggraph.org.mx/boletin/Ene99/index.htm.

33. **Pérez de Alejo Vitoria, Hector Eugenio y Hernández León, Rolando Alfredo.** *Informatización de la ingeniería de proceso en la industria química cubana.*

34. **Solis Alvares, Camilo Javier y Figueroa Dias, Roberth Gustavo.** *Metodologias tradicionales vs mrtodologias agiles.*

35. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *Proceso Unificado De Desarrollo De Software.* 2004.

36. **Canós, Jose, Letelier, Patricio y Penadés, Carmen.** *Metodologias Agiles en el Desarrollo del Software.*

37. **Passerini, Ing. Nicolás y Brey, Ing. Gustavo A.** *Metodologías Iterativas de Desarrollo.* Buenos Aires : s.n., 2005.

38. **Louden, Kenneth C.** *Lenguajes de Programacion.*

39. **Manzanedo del Campo, Miguel Ángel y Cruzado Nuño, Ignacio.** *Guía de iniciación al lenguaje Java.* 1999.

40. **Queiruga, lic. Claudia y Fava, lic. Laura.** *Herramientas de desarrollo para JAVA enSoftware Libre.*

41. *Paradigma_Visual_para_UML.* [En línea] [Citado el: 20 de 2 de 2010.] http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(MÍ)_14720_p/.

42. *AUP.* [En línea] [Citado el: 17 de 2 de 2010.] http://cgi.una.ac.cr/AUP/.

43. *visual-paradigm.* [En línea] [Citado el: 18 de 1 de 2010.] http://www.visual-paradigm.com.

44. **Booch, G., Jacobson, I. y Rumbaugh, J.** *The UML specification documents.* Santa Clara : s.n., 1997.

45. **Larman, Craig.** *UML y Patrones Introducción al análisis y diseño orientado a objeto.*

Glosario de términos

Sistemas físicos químicos: Es un sistema en el que se llevan a cabo procesos tanto físicos como químicos. El proceso físico es aquel en el que los componentes involucrados no alteran su composición química, mientras que en un proceso químico sí.

UCI: Universidad de las Ciencias Informáticas.

Polo Petrosoft: Grupo de trabajo de estudiantes y profesores de la Facultad 9 de la UCI que se dedica a desarrollar software para la industria petrolera.

Diseño en comunicación visual: disciplina profesional que estudia los sistemas de información, con el objeto de convertir los datos en formas visuales, teniendo en cuenta los procesos perceptivos. Consiste en la creación de imágenes funcionales con fines netamente comunicacionales.

Diseño industrial: rama del diseño que busca crear o modificar objetos o ideas para hacerlos útiles, prácticos o simplemente bellos con la intención de cubrir necesidades del ser humano.