Universidad de las Ciencias Informáticas
Facultad 3

Título: "Desarrollo de la arquitectura del Sistema para la Simulación de Procesos en la Industria Química Cubana."

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Yurisnel Corrales Valdés

Juan Carlos Suárez López

Tutor: Ing. Arnaldo Gandol Álvarez

Consultante: Dr. Héctor E. Pérez de Alejo Victoria

Ciudad de La Habana, Mayo 19, 2007

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio así como a la institución mencionada.

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

Juan Carlos Suárez López	**Yurisnel Corrales Valdes**	**Ing. Arnaldo Gandol Álvarez**
_____	_____	_____
Firma del Autor	Firma del Autor	Firma del Tutor

DATOS DE CONTACTO

Tutor
Ing. Arnaldo Gandol Álvarez.
E-mail: aga@uci.cu, agacode@gmail.com
Universidad de las Ciencias Informáticas.

Consultante
Dr. Héctor E. Pérez de Alejo Victoria.
E-mail: alejo@quimica.cujae.edu.cu
Facultad de Ingeniería Química, Cujae.
Teléfono: 8322970.

La virtud, como el arte, se consagra constantemente a lo que es difícil de hacer, y cuanto más dura es la tarea, más brillante es el éxito.

Anónimo.

Agradecimientos

Quisiéramos agradecer en primer lugar a nuestro tutor Arnaldo que sin él, muchas de las cosas que hemos logrado con este trabajo no hubieran sido posibles. Le agradecemos por su apoyo, guía y esfuerzo personal.
A nuestro grupo de desarrollo Energía, a los que están y los que se han ido, gracias a su esfuerzo este proyecto ha tenido los logros y reconocimientos alcanzados.
Al Dr. Héctor Pérez de Alejo, por su constancia, conocimiento y experiencia aportada.

Dedicatoria

Hay alguien especial en este mundo a quien le debemos ante todo el alma, el ser, la vida. Alguien que llora con tus tristezas y ríe con tus alegrías, pero no te deja solo; alguien que te mira con los ojos y te toca el alma; alguien que lucha por ti aun cuando tu fuerzas han fallecido; y es precisamente a mi madre que quisiera dedicar esta tesis en primer lugar. Vieja de veras te mereces esto y mucho más por ser mi luz y ejemplo.

A mi padre que aunque no como quizás ambos hubiéramos querido pero que si ha estado ahí a mi lado y nunca me ha dejado solo. Sus ideas y ética profesional me acompañarán toda la vida.

Si tener una madre y un padre es un regalo de dios, tener dos madres y dos padres es una bendición y yo le doy gracias a Dios por haberme dado la oportunidad de poder contar con ello. A mi mamá Luisa y a mi papá Lorenzo quiero dedicar esta tesis, a ustedes por darme la oportunidad de tenerlos como padres y deberles parte de mi forma de ver y luchar por la vida.

Un hombre sin la fe de una mujer que luche a tu lado, te de fe cuando la pierdas, te de fuerzas cuando la necesites, te guíe cuando tu ojos no puedan ver, te de amor, pasión y compañía; es hombre que esta pasando por la vida sin la mitad de su ser. A mi novia Daymi por ser mi amante, mi amiga y mi hermana.

A mi prima Yarisel por estar a mi lado todo este tiempo y ayudarme.

A mi familia toda por mantenerse unida y ayudar al niño a crecer y sentir como siente.

Juan Carlos

A mi familia, por el apoyo constante. A mis amigos de los buenos y malos momentos. A mi padre que me enseñó a luchar siempre por lo que se quiere. A mi madre Celina, mi fuente de inspiración constante. A mi segunda madre Marlenis. A mis hermanos Ernesto, Edel, Erick y Yoandrys.

Yurisnel

Resumen

El desarrollo de la computación y la informática en el mundo ha revolucionado prácticamente todos los aspectos de la vida y de la actividad social, política, económica, productiva y de investigación del ser humano. Una de las formas en que se ven influenciados los procesos de investigación y producción es mediante la simulación de los procesos antes mencionados haciendo uso de animaciones, modelos matemáticos y otras técnicas soportadas por la computación. En Cuba han sido aplicadas simulaciones a los procesos de las producciones de azúcar crudo y refino, alcohol, ácido sulfúrico, sosa cáustica, refinación de petróleo, níquel y cobalto y otras; pero lo logrado en este sentido está fuertemente limitado debido a que : a) los software cubanos, carecen de algunas de las funcionalidades de sus similares extranjeros, sobre todo aquellas que posibilitan el análisis y la síntesis de los resultados de las simulaciones de procesos completos y b) los software extranjeros solo pueden usarse en Cuba para fines académicos. El trabajo que se presenta a continuación guía su atención hacia el desarrollo de un framework que permite la informatización de los procesos químicos a través del uso de simuladores y se centra fundamentalmente en las actividades realizadas por el grupo de arquitectos para obtener una arquitectura estable, flexible, reusable y que sobre todo cumpliera con los requisitos funcionales y no funcionales necesarios para el funcionamiento del sistema. La arquitectura propuesta permitirá a los desarrolladores implementar cualquier tipo de simulador de procesos químicos con relativa facilidad y con tiempos de desarrollo cortos, aumentando así su productividad y eficiencia.

PALABRAS CLAVE: Arquitectura de software, simuladores.

TABLA DE CONTENIDOS

FIGURAS

TABLAS

Introducción

Antecedentes

Una de las vías más exitosas para lograr la eficiencia en la industria es a través de herramientas de la ingeniería de procesos como son la simulación, el análisis y la síntesis de procesos apoyada en técnicas de inteligencia artificial, minería de datos e integración de masa y energía a los procesos; complementadas con análisis ambientales y económicos.

Desde la década de los años 60 se desarrollaron y aplicaron 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, Flexible FLOWSHEET de la Corporación Kellogg, PEDLAN de la Compañía Petrolera MOBIL. También en esos años en el sector académico de Canadá y USA se crearon el PACER y el SPEEDUP. Algunos de los Simuladores mencionados dieron lugar a nuevas versiones o nuevos desarrollos, muchos de ellos han sido utilizados en con fines académicos en Cuba, por ejemplo en el ISPJAE se han utilizado simuladores como el GEMCS, GASP II, CHEMCAD, HYSYS, SUGARS, ASPEN PLUS y otros.

En Cuba se han desarrollado simuladores y se han aplicado a procesos completos o subprocesos como los relacionados con la producción de etanol, refinación de petróleo y de sulfato ferroso. En el ICIDCA se dispone de un Simulador, orientado a ecuaciones, del proceso azucarero y de alcohol 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 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 ó 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.

Todo lo logrado en este sentido está limitado fuertemente porque: 1) los software cubanos carecen de varias de las facilidades de sus similares extranjeros, sobretodo de las necesarias para el análisis y la síntesis de los resultados de las simulaciones de procesos completos y 2) los software extranjeros solo pueden usarse en Cuba e internacionalmente para fines académicos

Situación Problémica: En Cuba en la mayoría de los casos se emplean copias no autorizadas de simuladores extranjeros para la simulación de procesos. Por otra parte todos los software cubanos carecen de las facilidades computacionales de sus similares extranjeros y si la tienen son específicos de un tipo de industria, además carecen de facilidades para la síntesis de los resultados de las simulaciones de procesos completos y de técnicas de integración de energía y masa; sin contar que la manera en que han sido implementados impide extender sus funcionalidad o mejorarlos sin tener que rehacerlos totalmente, ni siquiera reutilizar las facilidades que incluyen.

Problema Científico: ¿Cómo desarrollar la arquitectura robusta, flexible y reusable para un software que permita la simulación de cualquier proceso de la industria química?
Objeto: Proceso de desarrollo de un simulador de procesos para la industria química cubana.
Objetivo: Obtener la arquitectura del sistema para la simulación de procesos de la industria química cubana garantizando que esta sea robusta, flexible y reusable.
Para lograr este objetivo se proponen los siguientes objetivos específicos:

1. Hacer un estudio de las arquitecturas existentes y evaluar hasta que punto pueden ayudar a realizar el trabajo.
2. Determinar las funcionalidades más importantes desde el punto de vista arquitectónico.
3. Establecer la arquitectura inicial del sistema.
4. Establecer la estrategia de cómo evolucionará la arquitectura base para obtener un producto de calidad.

Campo de Acción: Desarrollo de la arquitectura del simulador de procesos para la industria química cubana
Hipótesis: Si el grupo de arquitectos tiene un buen dominio de la tecnología a utilizar y hace un correcto análisis de los requisitos se puede obtener una arquitectura robusta, flexible y reusable y que cumpla con las funcionalidades requeridas.
El documento esta estructurado en tres capítulos. El primero trata los elementos relacionados con la fundamentación teórica y el estudio del estado del arte. En él se hace un análisis de los principales conceptos de la arquitectura de software, arquitectos de software y sobre los estilos y patrones

arquitectónicos. Se hace un estudio de arquitecturas para simuladores y de la situación del proyecto productivo antes de iniciar el trabajo.

El segundo capítulo muestra la solución propuesta para resolver el problema, detallada según las vistas arquitectónicas propuestas por RUP, también se analizan los patrones de diseño y arquitectónicos utilizados y la tecnología de implementación seleccionada.

En el capítulo tres se hace un análisis crítico de la solución propuesta de acuerdo a los atributos de calidad propuestos en la norma ISO 9126 y se evalúa el impacto de la arquitectura sobre el proyecto productivo.

Capítulo 1: Estado del arte y fundamentación teórica.

En este capítulo se hace un análisis de las diferentes corrientes arquitectónicas a nivel mundial, el estudio de patrones y estilos arquitectónicos, y de las principales tareas que debe cumplir un arquitecto. El lector encontrará también un tópico relacionado con la arquitectura de un simulador de procesos químicos, acompañado de un análisis crítico de esa arquitectura.

La arquitectura de software.

Por arquitectura se entiende el reparto de funciones en diferentes módulos o aplicaciones de lo que constituye el software y el tránsito de información entre ellos o comunicaciones. La arquitectura de software es la estructura de más alto nivel de un sistema de software. Está a un nivel tal de abstracción que el sistema puede ser visto como un todo. La estructura debe soportar la funcionalidad requerida para el sistema. De esta forma, el comportamiento dinámico del sistema debe ser tenido en cuenta al momento de diseñar la arquitectura. La estructura debe estar diseñada de acuerdo a los requerimientos funcionales y no funcionales, como: desempeño, seguridad y confiabilidad, así como flexibilidad y extensibilidad a fin de poder adaptar la funcionalidad en el futuro a un costo razonable. La arquitectura del software abarca decisiones importantes sobre:

- La organización del sistema software.
- Los elementos estructurales que compondrán al sistema y sus interfaces, junto con sus comportamientos, tal y como se especifican en las colaboraciones entre estos elementos.
- La composición de los elementos estructurales y del comportamiento en subsistemas progresivamente más grandes.
- El estilo de la arquitectura que guía esta organización: los elementos y sus interfases, sus colaboraciones y su composición **[1]**.

La arquitectura del software aporta una visión abstracta de alto nivel, posponiendo el detalle de cada uno de los módulos definidos a pasos posteriores del diseño. Su objetivo principal es aportar elementos que ayuden a la toma de decisiones y, al mismo tiempo, proporcionar conceptos y un lenguaje común que permitan la comunicación entre los equipos que participen en un proyecto. Esta no solo esta relacionada con la estructura y el comportamiento del sistema, sino que esta también dictada por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y

compromisos económicos y tecnológicos, y la estética. Por lo que la relación siguiente hace una recopilación de los objetivos más importantes que debe cubrir un diseño de arquitectura:

- **Dar cobertura a la funcionalidad**: el diseño del software debe en primer lugar centrarse en dar cabida a todas las funcionalidades que se especifican, o sea, debe incluir dentro de su diseño la solución a los requisitos que le dan funcionalidad al sistema.
- **Facilitar el desarrollo del proyecto de software**: un software puede ser un proyecto complejo donde intervengan docenas de personas y se extienda durante años. Es un objetivo crucial que el reparto de funciones entre los distintos módulos permita un desarrollo independiente, pero coordinado, de los mismos. La programación orientada a objetos y a componentes permite, por su filosofía de origen, una programación modular a la vez independiente pero coordinada.
- **Optimizar el uso de los recursos del hardware**: los recursos se refieren a la capacidad de cálculo de los procesadores, al uso de la memoria, al almacenamiento y acceso a los discos, a la capacidad de generación gráfica y a la capacidad de comunicaciones.
- **Permitir el intercambio entre módulos**: para mejorar y ampliar las capacidades del software en el futuro es preciso que la arquitectura sea lo suficientemente abierta para poder sustituir un módulo sin que se vean afectados demasiados los demás.

Evidentemente muchos de estos aspectos están condicionados por distintos factores que son los que permiten y modelan el alcance de los objetivos anteriores. He aquí algunos de ellos:

- **Recursos de hardware**: es evidente que la capacidad del hardware en general condiciona el reparto de funciones del software. También sucede que en algunos casos se plantea la tesitura de realizar algunas funciones mediante software o a través de un hardware específico, como puede ser en temas de generación gráfica, el acceso a datos, o la capacidad de cálculo.
- **Capacidad del software**: el lenguaje de programación y las capacidades intrínsecas de los sistemas operativos pueden condicionar de manera definitiva una arquitectura u otra. La capacidad para generar componentes o la ejecución remota podrían ser ejemplos de este tipo de condicionantes.
- **Capacidad presupuestaria**: bajo este concepto se engloba la capacidad económica, que condicionará hardware y número y experiencia de programadores, y la disponibilidad temporal, un desarrollo rápido exige una arquitectura sencilla.

La arquitectura dirige el desarrollo de los sistemas software y contribuye a que estos se lleven a cabo en los límites establecidos de costos y tiempo, y fundamentalmente debe garantizar que se cumpla con los requisitos funcionales y no funcionales de los usuarios. La arquitectura es el resultado del trabajo durante todo el ciclo de vida del proyecto, y principalmente en las primeras iteraciones, de un grupo de trabajo encabezado por el arquitecto (o grupo de arquitectura, en dependencia de las dimensiones del proyecto).

Los arquitectos de software hoy.

El rol del arquitecto de software implica articular la visión arquitectónica, conceptualizando y experimentando con enfoques de arquitecturas alternativas, creando modelos, componentes y documentos de especificación de interfaces, validando la arquitectura contra los requerimientos. Este rol involucra no sólo las actividades técnicas mencionadas, sino también otras de naturaleza más política y estratégica a fin de definir la arquitectura correcta para el problema del cliente, dados los objetivos de negocio de la organización. En esta área las actividades incluidas son: creación de la estructura a un alto nivel del software, plantear las posibles evoluciones tecnológicas y determinar sus consecuencias en la estrategia técnica y en el enfoque de la arquitectura planteada. El arquitecto debe ser capaz de:

1. **Descomponer la aplicación en capas, al menos, lógicas**. La descomposición en capas lógicas es fundamental, y de hecho debe ser tal que pueda acomodarse a más de una descomposición en capas físicas, en la medida en que la aplicación -por razones de escalabilidad y/o rendimiento- deba re desplegarse.
2. **Descomponer cada capa lógica en componentes**. Esto es, asignar claramente las responsabilidades de ejecución, ya sea en clases concretas o abstractas, o quizás también en interfaces que posean contratos completamente definidos, de manera tal que los desarrolladores de lógica de aplicación completen su implementación.
3. **Seleccionar tecnologías y/o frameworks de implementación**. El arquitecto debe ser un experto en manipular la complejidad. En la tarea de seleccionar las tecnologías a utilizar el arquitecto debe ser capaz de adaptarlas de manera que se aprovechen la potencia y las funcionalidades que estas ofrecen pero tratando de evitar los mecanismos complejos o tratándolos a un nivel superior de la arquitectura de forma que no se tenga que obligar a todo el equipo a interactuar con las partes más complejas.
4. **Realizar una prueba de concepto de la arquitectura**. La prueba de concepto es una etapa necesaria para validar la arquitectura, para decidir si seguir adelante o someterla a ajustes o revisiones

ulteriores. Pero fundamentalmente, es un elemento que al arquitecto le debe proveer confianza en sus decisiones, y a la vez inspirar confianza en el resto de los interesados en la aplicación.

5. **Brindar algunos casos de uso de referencia**. Tiene como finalidad explicar como juegan, en la práctica, la arquitectura y los componentes a implementar por los desarrolladores. Cuanto más representativos y emblemáticos sean los casos de uso, tanto mejor. Esto le va a dar no sólo retroalimentación al arquitecto respecto de la arquitectura que definió, sino que además le va a permitir ganar confianza respecto de cuán asimilable es su arquitectura.

A diferencia de un programador, el arquitecto de software debe dominar la mayor cantidad de tecnologías de software y prácticas de diseño, para así poder tomar decisiones adecuadas para garantizar el mejor desempeño, reuso, robustez, portabilidad, flexibilidad, escalabilidad y mantenibilidad de las aplicaciones. Estas decisiones sobre la estructura y dinámica de la aplicación son plasmadas en una notación formal estandarizada como lo es UML; sobre todo si se utilizan las nuevas tecnologías, en especial con los lenguajes orientados a objetos.

El arquitecto de software es el líder técnico del equipo, el rol natural al que debe aspirar un programador experimentado que desea tomar decisiones técnicas relevantes en el desarrollo de un sistema. Es el principal tomador de decisiones respecto a la manera en que será construida la aplicación por los programadores del equipo. El líder de proyecto se apoya totalmente en este rol para alcanzar el éxito del proyecto optimizando el uso de la tecnología para desarrollar la solución correcta que proporcionará valor real a sus usuarios y al negocio al que le dará soporte.

Hay dos formas de convertirse en arquitecto: aprendiendo a definir las soluciones con base en la propia experiencia, o reutilizando el conocimiento de los expertos a nivel mundial plasmado en patrones de arquitectura y diseño.

Es común cuando se busca en este mundo del desarrollo de software escuchar sobre la rivalidad entre arquitectos y desarrolladores. En general los programadores alegan que los arquitectos definen todo a muy alto nivel, presentando a los clientes y directivos del proyecto diseños muy interesantes y convincentes, pero que a la larga, implementarlos es prácticamente una utopía y son precisamente los desarrolladores los que tienen que escribir las líneas de código para materializar el diseño. Un arquitecto de software real debe ser capaz de interactuar con los desarrolladores y convertirse en uno de ellos, estar en contacto directo con ellos y establecer una retroalimentaron constante para lograr que el esqueleto de la arquitectura inicial vaya llenándose con músculos y que al final de este largo proceso de hacer un software se logre un producto de calidad. Es ideal que el arquitecto haya sido programador, entonces,

estará en condiciones de saber si lo que propone en sus diagramas es posible materializarlo en líneas de código. El arquitecto es el líder técnico del proyecto y tiene que estar al lado de los programadores velando que las cosas se hagan de acuerdo como se ha especificado en la arquitectura, es importante también su presencia para ayudar y guiar a los desarrolladores novatos que aun no conocen la arquitectura.

Estilos y patrones arquitectónicos.

Anteriormente se comentaba que existen dos formas de convertirse en arquitecto, “aprendiendo a definir las soluciones con base en la propia experiencia, o reutilizando el conocimiento de los expertos a nivel mundial plasmado en patrones de arquitectura y diseño.” El primer elemento es incuestionable, la experiencia adquirida en el trabajo durante muchos años en la definición de arquitecturas es una carta de triunfo a la hora de enfrentarse a un nuevo proyecto. El arquitecto verá aparecer antes sus ojos situaciones por las que ya pasó anteriormente, y será capaz de tomar decisiones teniendo en cuenta sus triunfos y fracasos en proyectos anteriores. El segundo factor no deja de ser importante, el arquitecto debe conocer la mayor cantidad de patrones y estilos posible, y no solo conocerlos, sino dominarlos. Si bien es cierto que por ambas vías se llega a arquitecto, se considera que lo mejor es tener ambas cosas, el mejor arquitecto es aquel con más experiencia y que domina la mayor cantidad de patrones, de esta manera tendrá en su poder los elementos que fue adquiriendo durante su trabajo más la experiencia de muchos otros arquitectos abarcada en los patrones.

En la bibliografía referida al estudio de los patrones arquitectónicos se suele llamar de la misma manera a los patrones y estilos, pero la mayoría de los autores los ven de manera separada. Ambos se refieren a formas de estructurar los sistemas y como relacionar los componentes de estos, la diferencia radica en el nivel de abstracción en que se aplican. Los estilos están en un plano de abstracción más alto, definiendo como configurar la arquitectura, mientras los patrones están más cercanos al diseño, incluso podría decirse que más cercano a algo físico, pues estos pueden representarse mediante código en un lenguaje de programación determinado.

Según Mary Shaw y David Garlan en **[2]** un estilo arquitectónico define a una familia de sistemas en términos de un patrón de organización estructural. Específicamente, un estilo arquitectónico determina el vocabulario de componentes y conectores que puede ser usado así como un conjunto de restricciones de cómo pueden ser combinados. Robert T. Monroe en **[3]** plantea que un estilo arquitectónico provee una colección de elementos edificadores del diseño en bloque, reglas y restricciones para componer los

bloques constructivos, y las herramientas para analizar y manipular los diseños creados en el estilo. Los estilos generalmente proveen guía y análisis para crear una clase amplia de arquitecturas en un dominio específico donde los patrones se enfocan en solucionar los problemas más pequeños, más específicos dentro de un estilo dado.

Algunos de los estilos arquitectónicos más difundidos son los que se muestran a continuación:

- Estilos de Flujo de Datos
- Tubería y filtros
- Estilos Centrados en Datos
- Arquitecturas de Pizarra o Repositorio
- Estilos de Llamada y Retorno
- Modelo Vista Controlador (MVC)
- Arquitecturas en Capas
- Arquitecturas Orientadas a Objetos
- Arquitecturas Basadas en Componentes
- Estilos de Código Móvil
- Arquitectura de Máquinas Virtuales
- Estilos heterogéneos
- Sistemas de control de procesos
- Arquitecturas Basadas en Atributos
- Estilos Peer-to-Peer
- Arquitecturas Basadas en Eventos
- Arquitecturas Orientadas a Servicios
- Arquitecturas Basadas en Recursos

En **[4]** Frank Buschmann define los patrones arquitectónicos como sigue: “Un problema particular recurrente del diseño que surge en un contexto de diseño específico y presenta esquema genérico bien probado para la solución de dicho problema. Esta solución es especificada describiendo sus componentes constitutivos las responsabilidades de estos componentes, sus relaciones y la forman en que colaboran”. Los patrones arquitectónicos están relacionados con la interacción de los objetos dentro o entre niveles arquitectónicos, están dirigidos a la solución de problemas arquitectónicos como adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento, etc. Mientras las soluciones

propuestas tienen que ver con patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de funcionalidad, entre otros aspectos. En la actualidad los patrones se encuentran muy difundidos y se les puede encontrar por cientos, a continuación se listan los referidos en [4] y que son los más populares y conocidos por los arquitectos:

- **Layers**: ayuda a estructurar aplicaciones que puede estar descompuestas en grupos de subtareas en las cuales cada grupo de subtareas esta en un nivel particular de abstracción.
- **Pipes and Filtres**: modelan una estructura para los sistemas que procesan una corriente de datos. Cada fase de elaboración es encapsulada en un filtro. Los datos por tuberías entre filtros adyacentes. Combinando los filtros se pueden crear familias de sistemas relacionados.
- **Blackboard**: es útil para problemas para los cuales ninguna de las estrategias deterministas de solución son conocidas. En Blackboard varios subsistemas especializados instrumentan su conocimiento para fortalecer una solución posiblemente parcial o aproximada.
- **Broker:** puede usarse para estructurar sistemas distribuidos del software con componentes desacoplados que interactúan por invocaciones remotas de servicio. Un componente del agente es responsable de coordinar comunicación, algo semejante como reenviar demandas múltiples, como para excepciones y resultados transmisores.
- **Model View Controller (MVC):** divide una aplicación interactiva en tres componentes. El modelo contiene los datos y funcionalidades de fondo. La vista exhibe información para el usuario. Los controladores manipulan las entradas del usuario. La vista y el controlador conjuntamente comprenden la interfaz de usuario. Un mecanismo de propagación de cambio asegura consistencia entre la interfaz de usuario y el modelo.
- **Presentation-Abstraction-Control (PAC):** define una estructura para los sistemas interactivos del software en forma de una jerarquía de agentes cooperadores. Cada agente es responsable de un aspecto específico de funcionalidad de la aplicación y consta de tres componentes: La presentación, la abstracción, y el control. Esta subdivisión separa los aspectos de interfaz persona-ordenador del agente de su corazón funcional y su comunicación con otros agentes.
- **Microkernel:** se aplica a los sistemas del software que deben poder adaptarse a requisitos de sistema cambiantes. Separa un corazón funcional mínimo de funcionalidad extendida y partes creadas por el usuario. El microkernel también sirve de un conector para enchufar estas extensiones y coordinar su colaboración.

- **Reflection:** provee un mecanismo para el comportamiento y estructura cambiante de sistemas del software dinámicamente. Soporta la modificación de aspectos fundamentales, como las estructuras de tipo y los mecanismos de llamada de función. En este patrón, una aplicación es dividida en dos partes. Un meta nivel provee información acerca de propiedades seleccionadas de sistema y hace el software consciente de sí mismo. Un nivel de base incluye la lógica aplicativa. Su implementación se fundamenta en el meta nivel.

Como se puede observar existen los elementos teóricos suficientes como para enfrentarse a la concepción de una arquitectura, a juzgar por la cantidad de experiencia acumulada en patrones y estilos debe existir alguna estas soluciones prefabricadas que se ajuste o pueda ser usada para definir la arquitectura del simulador que es el objeto de este trabajo. A continuación se hace un análisis de sistemas parecidos y sus arquitecturas

Arquitecturas para simuladores.

La arquitectura está más relacionada con la concepción, prestaciones y desarrollo del software que con la funcionalidad final del simulador. Obviamente arquitectura y funcionalidad están íntimamente ligadas y la primera se orienta dependiendo de la segunda, pero la arquitectura tiene otros condicionantes y objetivos. Independientemente de otros criterios más sutiles, como la extensión de la filosofía cliente-servidor, el factor más importante a considerar en el diseño de una arquitectura es su grado de concentración o su grado de distribución. Una arquitectura concentrada es aquella en la que hay una o pocas aplicaciones que concentran cada una un número elevado de funcionalidades y funcionan en uno o pocos ordenadores con poco flujo de comunicación exterior, entiéndase por aplicación el módulo de software que puede funcionar autónomamente en un solo ordenador. Una arquitectura distribuida tiene varias o muchas aplicaciones funcionando en varios ordenadores con un flujo importante de comunicaciones.
La arquitectura concentrada permite desarrollos simples de muy corto alcance. Aunque debido al incremento de potencia de la computación personal permite resultados casi profesionales, la arquitectura concentrada se queda corta rápidamente en simuladores complejos. Cuando se quieren implementar simuladores complejos como los simuladores a tiempo real resulta casi absurdo pensar en una arquitectura concentrada, la distribuida ofrece múltiples beneficios y sin duda es la candidata perfecta. Entre sus principales ventajas están:

- Evita la sobrecarga de procesador con cálculos sobre los modelos matemáticos y generación de la escena.
- Permite una mayor reutilización del código: al ser compartimentos más o menos estancos, las mayores variaciones se realizan en interfaz de usuario.
- El uso de ordenadores personales reduce el coste inicial de implantación.
- Los ordenadores personales son altamente fiables, se reparan fácilmente y se sustituyen de forma inmediata.
- Es software empleado es de gran difusión y se encuentra fácilmente software desarrollado y personal cualificado.
- El uso del mismo tipo de ordenador para tareas distintas permite un coste de mantenimiento más reducido.

Un ejemplo importante a mencionar cuando se habla de arquitecturas distribuidas y simulación de procesos es el Standard CAPE-OPEN, plataforma que ofrece interoperatividad e integración de componentes para la simulación de procesos físicos y químicos. Es multiplataforma y utiliza importantes frameworks y middlewares como CORBA, COM, J2EE, EJB, etc., orientada a componentes, tiene una concepción de interfaces orientadas a objetos y usa técnicas de reingeniería y encapsulación. Entre los ambientes para la simulación de procesos desarrollados sobre esta plataforma están ProsimPlus de la compañía ProSim SA, simulador estacionario para la producción de oxígeno, INDISS de RSI, este último para la simulación dinámica y otros como Aspen Engineering Suite, Process Engineering Suite, HYSYS, INDISS, ProSim Plus, gPROMS.

Al hablar de simuladores estacionarios y no muy complejos, como el que se trata en este trabajo es muy apropiado el uso de una arquitectura concentrada, pues se evitan inconvenientes como:

- Necesidad de adquirir y aprender un software para las comunicaciones entre los distintos ordenadores.
- Problemas de organización del tráfico de información para garantizar la consistencia de las comunicaciones.
- Necesidad de un hardware de red de suficiente fiabilidad.
- Los problemas aparejados a la depuración de los programas en arquitecturas distribuidas en los cuales se dificulta enormemente.

En estos casos es conveniente proponer una arquitectura propia que se adapte a las condiciones de lo que se quiere lograr con el simulador en materia de funcionalidad y que garantice un desempeño adecuado cuando este terminado el producto. Lo más importante no seria el grado de concentración sino que esta se estructure de manera eficiente y se puedan aprovechar mejor los recursos, además de elementos tan importantes como la reusabilidad, interoperatividad, flexibilidad, mantenebilidad, robustez, etc. Otro factor fundamental a tener en cuenta es el hecho de que sea multiplataforma, lo que amplia las posibilidades de uso del simulador, sobre todo en la actualidad mundial donde cada vez se hace más popular el uso de otros sistemas operativos a parte de la plataforma de Windows, como son los de la plataforma Linux.

Situación del proyecto.

En la etapa de concepción del proyecto de desarrollo del simulador de procesos de la industria química se manejaba la idea de construir el nuevo sistema sobre una versión del simulador que ya existía en plataforma MS-DOS y programado en el lenguaje de programación Pascal, el simulador STA (Sistema Termoazúcar). Era interés de los clientes que se reutilizaran todas las units programadas en Pascal de manera que no se perdieran los modelos que ya estaban incluidos y probados, validados por varios años de uso del simulador con resultados satisfactorios. Se hizo un estudio completo del STA del que se llegó a la conclusión de que no era posible la reutilización de ese sistema si se quería crear un simulador que cumpliera con todos los requisitos que pedían y que además tuviera una arquitectura robusta, reusable y flexible. Entre las limitaciones fundamentales de la arquitectura del STA estaba la falta de una estructura bien definida en la que se pudieran identificar las diferentes partes del simulador. Por otra parte las concepciones para la realización de la simulación no se ajustaban a las nuevas condiciones que dictaban los requisitos. La forma de tratar elementos tan importantes como las corrientes y la comunicación entre los modulos de cálculo son demasiado inflexibles.

Capítulo 2: Descripción y análisis de la solución propuesta.

Hasta el momento se han analizado los elementos teóricos necesarios para enfrentarse a la concepción de la arquitectura del sistema para la simulación de la industria química cubana. Ha llegado la hora de describir dicha arquitectura. Cuando se describe una arquitectura un elemento fundamental es el concepto de vista, según Reynoso en **[5]** “una vista es un subconjunto resultante de practicar una selección o abstracción sobre una realidad, desde un punto de vista determinado”. En otras palabras la vista es un conjunto de artefactos de documentación resultantes de analizar el sistema desde un punto de vista. Usualmente se usa la analogía entre la arquitectura de edificios y la arquitectura del software, analogía que no es tan exacta, pero a la hora de analizar las vistas es muy útil, de la misma manera que los arquitectos de edificios definen vistas para diferentes aspectos de los edificios como el cableado eléctrico, la plomería, la ventilación, la disposición de las habitaciones, etc. los arquitectos de software deben describir la arquitectura de un sistemas desde distintos aspectos del sistema, tales como la estructura del sistema atendiendo a los elementos de software que lo componen, la disposición de estos elementos en el hardware, etc. Existen varias definiciones de las diferentes vistas que componen un sistema software, estas van desde 3 hasta 36, como es el caso de John Zachman, que las propone basado en seis niveles (alcance, empresa, sistema lógico, tecnología, representación detallada y funcionamiento empresarial) y seis aspectos (datos, función, red, gente, tiempo, motivación). En general todas se refieren a aspectos fundamentales como la composición del sistema, la distribución de los elementos componentes en el hardware, la especificación de los procesos y flujos de información y el sistema visto como componentes físicos en ficheros. En el marco de la política de desarrollo de software de la Universidad de las Ciencias Informáticas, que definen como metodología a RUP, en este trabajo se describirá la arquitectura en términos de las 4+1 vistas propuestas por esta metodología: (1) La vista de casos de uso, como la perciben los usuarios, analistas y encargados de las pruebas; (2) la vista de diseño que comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución; (3) la vista de procesos que conforman los hilos y procesos que forman los mecanismos de sincronización y concurrencia; (4) la vista de implementación que incluye los componentes y archivos sobre el sistema físico; (5) la vista de despliegue que comprende los nodos que forma la topología de hardware sobre la que se ejecuta el sistema **[1]**.

Los requisitos en la arquitectura.

Los casos de uso juegan un papel fundamental en la concepción de la arquitectura de un sistema, en las primeras etapas de desarrollo de un software el esfuerzo es mayor en la captura de los requisitos y en la comprensión de lo que quiere el cliente. Diseñar la arquitectura basándose en los requisitos garantiza que esta sea funcional. Los casos de uso son formas de expresar los requisitos funcionales del sistema que se quiere desarrollar. Cuando se concibe la arquitectura se toman los casos de usos más significativos y a partir de ellos se define un sistema que soporte dichas funcionalidades. En esta sección se darán a conocer los requisitos funcionales del sistema para la simulación de la industria química cubana y se describirán los casos de uso arquitectónicamente significativos así como el porque de su clasificación como tal. Además se listaran los requisitos no funcionales, que también juegan un papel fundamental en la arquitectura.

Requisitos funcionales:

- Mostrar información
    - Mostrar información de los módulos
    - Mostrar información de las corrientes
- Insertar información
    - Insertar información corrientes
    - Insertar información módulos
- Convertir unidades de medida
- Crear DFI
    - Adicionar módulos al DFI
    - Conectar módulos del DFI
    - Copiar módulos
    - Cortar módulos
    - Pegar módulos
    - Mover los módulos en el DFI
    - Eliminar módulos
    - Rotar los modulos
- Analizar sensibilidad
- Realizar análisis sobre corridas múltiples

- Reportar resultados
    - Reportar todos los resultados
    - Reportar resultados seleccionados
- Analizar economía
- Simular
    - Determinar orden de cálculo
    - Calcular parámetros de los módulos
    - Calcular parámetros de las corrientes
- Identificar módulos
- Identificar corrientes
- Exportar datos a otros ficheros de texto
- Exportar DFI como imagen
- Mostrar en el DFI información de la simulación
- Calcular indicadores
    - Calcular indicadores por área
    - Calcular indicadores por equipos
    - Calcular indicadores globales
- Salvar toda la información manejada en el simulador

Requisitos no funcionales:

- El sistema debe ser una aplicación de escritorio.
- El sistema sebe ser multiplataforma
- La interfaz debe ser amigable y fácil de usar
- Debe estar protegido contra copias no autorizadas
- Permitir integrarse con otros formatos de aplicaciones conocidas
- Tener una paleta de figuras estándar
- Destacar con colores distintos las corrientes distintas por naturaleza
- Destacar con colores distintos los módulos cuya información ha sido suministrada
- Permitir exportar el diagrama como una imagen en formato jpg, bmp, etc.

Conocidos los requisitos es hora de ver los casos de uso que fueron clasificados como arquitectónicamente significativos:

- **Caso de uso Crear DFI.** Este caso de uso agrupa las funcionalidades relacionadas con la construcción de los Diagramas de Flujo de Información, aquí se incluye agregar módulos al diagrama, conectarlos, poder acomodarlos al gusto del usuario así como las operaciones básicas de cualquier editor tradicional, dígase cortar, copiar, pegar, deshacer y rehacer, etc. Desde el punto de vista de la arquitectura este caso de uso es tiene una importancia elevada, pues las funcionalidades que implican constituye la cara del simulador hacia el usuario, es la interfaz de comunicación entre el simulador y los que lo van a usar. Técnicamente, poder soportar esta serie de requisitos representa un trabajo fuerte y decisivo en la continuidad de la arquitectura, además que permitirá comprobar la validez de otras funcionalidades fundamentales del sistema.
- **Caso de uso Simular.** La misión principal del simulador es el cálculo de las propiedades físicas y químicas de los equipos y de las corrientes que fluyen entre ellos. Precisamente calculando esta serie de parámetros en un determinado orden se puede hablar de que el sistema es capaz de simular determinado proceso. La implementación de este caso de uso en la arquitectura inicial es determinante para comprobar cuan válida es la arquitectura.
- **Caso de uso Insertar Información.** Para lograr la simulación no solo se necesita tener un DFI y vías para calcular determinados parámetros. También son necesarios datos de entrada para la realización de dichos cálculos y para lograr la completitud del DFI. El hecho de entrar datos implica que de alguna manera deben ser almacenados estos datos, y he aquí donde aflora el porque se clasificó este caso uso como arquitectónicamente significativo, pues este implica la toma de decisiones importantes sobre la estructura del almacenamiento de los datos, así como cuestiones de accesibilidad a estos.
- **Caso de uso Reportar Resultados.** La simulación de un proceso arroja gran cantidad de información que debe ser organizada para una mejor comprensión de ella y que sea más fácil de analizar por los usuarios. En la arquitectura base se implementara parte de este caso de uso, permitiendo solamente un reporte condensado que muestre los datos que resultan de la operación de simular. La inclusión de este en la arquitectura base permitirá evaluar la capacidad del sistema de calcular de manera correcta y completa, es decir, que se obtienen todos los parámetros esperados por los usuarios.

- **Caso de uso Convertir Unidades.** Como se conoce las propiedades físicas y químicas de flujos y equipos tienen asociada una unidad de medida determinada que permite mejorar la calidad de la información que brinda el valor numérico asociado a esa propiedad. Existen distintos sistemas de unidades para designar las magnitudes, sistemas que son usados indistintamente por los especialistas, a veces mezclando los sistemas en que se dan los datos. El simulador que se pretende hacer debe dar la posibilidad de que el usuario suministre los datos y los reciba en la unidad de medida que prefiera, esto, en contraste con que los cálculos de las propiedades internamente se realicen con unidades específicas. La implementación de las conversiones entre unidades entre sistemas de medida o incluso dentro del mismo puede convertirse en un problema serio si no es bien tratado, esta es la motivación principal para incluirlo dentro de la arquitectura base.
- **Caso de uso Salvar Información.** Uno de las funcionalidades que tienen la mayoría de las aplicaciones es la de poder almacenar y recuperar la información que se está manejando, o sea salvar y cargar. En caso de la presente arquitectura se debe poder almacenar toda la información visual que contenga el DFI, los valores suministrados por el usuario, los valores calculados, el estado de los módulos, la información del DFI referente al autor, fecha, descripción entre otros, los análisis económicos del DFI, etc.

En el análisis de los requisitos se pueden identificar conceptos importantes que constituyen el sistema. Entre los elementos encontrados se encuentran el concepto de módulo, refiriéndose a los equipos de la industria; las corrientes, para referirse a los flujos que se mueven entre los equipos de la fabrica; los puertos que constituyen los puntos de conexión entre módulos a través de corrientes. También emerge como concepto importante el DFI, elemento que representa a la fábrica en su totalidad como un sistema en el que interactúan los diferentes equipos mediante las conexiones que se establecen entre ellos.

Selección de tecnologías de implementación.

La tecnología usada para el desarrollo de la arquitectura propuesta fue la Plataforma .NET, y para la selección de la misma se tuvieron en cuenta algunos aspectos los cuales serán enumerados a continuación:

- **Microsoft .Net:** es, de acuerdo con la definición de Microsoft, una plataforma que comprende servidores, clientes y servicios. Esta plataforma es una implementación basada en estándares

abiertos como SOAP, WSDL, o sea que es una plataforma que puede intercambiar con otras plataformas ya que trabaja sobre los Standard que rigen el mundo del desarrollo. Lo que permitirá futuras integraciones con proyecto desarrollados en otras plataformas.

- **Modelo de programación consistente:** A todos los servicios y facilidades ofrecidos por el CLR de .NET se accede de la misma forma: a través de un modelo de programación orientado a objetos. Esto es una diferencia importante respecto al modo de acceso a los servicios ofrecidos por algunos sistemas operativos actuales (por ejemplo, los de la familia Windows), en los que a algunos servicios se les accede a través de llamadas a funciones globales definidas en DLLs y a otros a través de objetos (objetos COM en el caso de la familia Windows). Esto permite el uso de estos servicios además de aprovechar las potencialidades de un lenguaje puramente orientado a objetos.
- **Eliminación del "infierno de las DLLs":** En la plataforma .NET desaparece el problema conocido como "infierno de las DLLs" que se da en los sistemas operativos actuales de la familia Windows, problema que consiste en que al sustituirse versiones viejas de DLLs compartidas por versiones nuevas puede que aplicaciones que fueron diseñadas para ser ejecutadas usando las viejas dejen de funcionar si las nuevas no son 100% compatibles con las anteriores. En la plataforma .NET las versiones nuevas de las DLLs pueden coexistir con las viejas, de modo que las aplicaciones diseñadas para ejecutarse usando las viejas podrán seguir usándolas tras instalación de las nuevas. Además facilita el desarrollo de la arquitectura por componentes.
- **Ejecución multiplataforma:** El CLR de .NET actúa como una máquina virtual, encargándose de ejecutar las aplicaciones diseñadas para la plataforma .NET. Es decir, cualquier plataforma para la que exista una versión del CLR podrá ejecutar cualquier aplicación .NET. Microsoft ha desarrollado versiones del CLR para la mayoría de las versiones de Windows: Windows 95, Windows 98, Windows ME, Windows NT 4.0, Windows 2000, Windows XP y Windows CE. Por otro lado Microsoft ha firmado un acuerdo con Corel para portar el CLR a Linux y también hay terceros que están desarrollando de manera independiente versiones de libre distribución del CLR para Linux, este proyecto es conocido como el proyecto Mono. Esta facilidad permite la migración futura hacia otros SO, dado que la arquitectura del CLR está totalmente abierta, es posible que en el futuro se diseñen versiones del mismo para otros sistemas operativos.
- **Integración de lenguajes:** Desde cualquier lenguaje para el que exista un compilador que genere código para la plataforma .NET es posible utilizar código generado para la misma usando

cualquier otro lenguaje tal y como si de código escrito usando el primero se tratase. Microsoft ha desarrollado un compilador de C# que genera código de este tipo, así como versiones de sus compiladores de Visual Basic (Visual Basic.NET) y C++ (C++ con extensiones gestionadas) que también lo generan y una versión del intérprete de JScript (JScript.NET) que puede interpretarlo. La integración de lenguajes está dada en que es posible escribir una clase en C# que herede de otra escrita en Visual Basic.NET que, a su vez, herede de otra escrita en C++ con extensiones gestionadas, o sea, desde el punto de vista del programador, el entorno .NET ofrece un solo entorno de desarrollo para todos los lenguajes que soporta, ejemplo Visual Basic, C++, C#, Visual J#, Fortran, Cobol. Esto da gran interoperabilidad entre los lenguajes y los desarrollares, puesto que no constituye una limitante en la adquisición de personal especializado en un lenguaje especifico.

- **Seguridad de tipos:** El CLR de .NET facilita la detección de errores difíciles de localizar comprobando que toda conversión de tipos que se realice durante la ejecución de una aplicación .NET se haga de modo que los tipos origen y destino sean compatibles. Esto lo posibilita un lenguaje fuertemente tipado, lo cual permite una arquitectura más sólida.
- **Tratamiento de excepciones:** En el CLR todos los errores que se puedan producir durante la ejecución de una aplicación se propagan de igual manera: mediante excepciones. Esto es muy diferente a como se venía haciendo en los sistemas Windows hasta la aparición de la plataforma .NET, donde ciertos errores se transmitían mediante códigos de error en formato Win32, otros mediante HRESULTs y otros mediante excepciones. El CLR permite que excepciones lanzadas desde código para .NET escrito en un cierto lenguaje se puedan capturar en código escrito usando otro lenguaje, e incluye mecanismos de depuración que pueden saltar desde código escrito para .NET en un determinado lenguaje a código escrito en cualquier otro. Por ejemplo, se puede recorrer la pila de llamadas de una excepción aunque ésta incluya métodos definidos en otros módulos usando otros lenguajes.

Vista lógica.

Paquetes del diseño.

El diseño juega un papel fundamental en la definición de la arquitectura de cualquier sistema. Durante el desarrollo de esta actividad ya se ve el sistema con un mayor nivel de detalle y se es más explicito en

como se construirá este. Para una mayor comprensión del diseño se construyeron paquetes que contienen las diferentes clases, la selección de los paquetes se hizo siguiendo el criterio de las funcionalidades que deben garantizar las clases y tratando de separar los elementos de interfaz de usuario y los de negocio. Los paquetes diseñados se muestran en la figura 1.

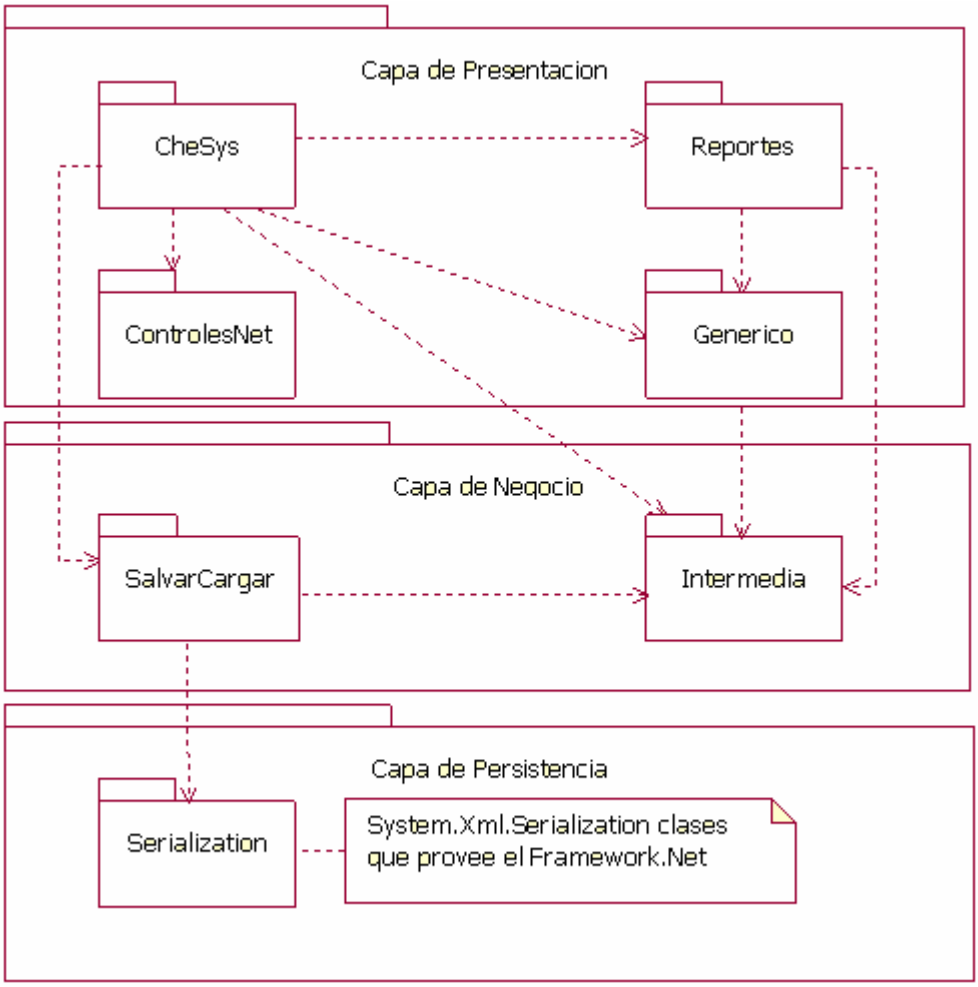

Figura 1 : Diagrama de paquetes del diseño.

Generico contiene el conjunto de clases que garantizan el funcionamiento del editor gráfico. Proporciona elementos claves como los módulos, líneas, puertos, el panel donde se dibujan los DFI, los elementos que permiten la visualización de la información contenida en los equipos y corrientes así como los formularios generales para la entrada de información por parte del usuario que interactúa con el simulador. Además define los mecanismos más importantes del editor gráfico para el manejo de los módulos y

conexiones, como son las operaciones de cortar, pegar, eliminar, insertar, deshacer/rehacer, rotar en diferentes direcciones, etc. Dentro de este componente se hace un uso extensivo de los controles de usuarios que provee la plataforma .NET, los puertos, módulos y conexiones son controles de usuarios, que encajan en una jerarquía de clases, donde las funcionalidades comunes se van quedando en la parte más alta de dicha jerarquía y se van especializando en cada uno de ellos las funcionalidades especificas que los hacen únicos. Estas relaciones se analizarán con mayor énfasis cuando se entre al diseño de cada una de estas clases. La razón fundamental de Generico es la encapsulación de funcionalidades, el hecho de que el grueso de los elementos visuales de la aplicación estén agrupados en este componente garantiza que se pueda cambiar fácilmente la interfaz del simulador sin afectar el negocio que se encuentra encapsulado en Intermedia, cuando se requiera cambiar la tecnología de interfaz gráfica solo hay que sustituir la implementación de Generico.

Si bien Generico juega un papel protagónico en la parte gráfica del simulador, **Intermedia** tiene otro tan o más importante en la parte del negocio, este componente provee los mecanismos para el cálculo de las propiedades de módulos y corrientes. Provee un paquete de funciones para el cálculo de propiedades físico-químicas generales y para la conversión de unidades. Aquí se definen las interfaces que rigen la comunicación entre los diferentes módulos de cálculo así como los tipos de corrientes que puede manejar el simulador, estas corrientes están estrechamente ligadas a la tecnología que se este simulando, en la actualidad solo están definidas las corrientes del proceso de obtención y refinación de azúcar. Este componente provee la estructura general de un módulo y sus puertos desde el punto de vista del negocio, entonces un módulo determinado se especificará en un componente independiente, el cual contendrá una clase que es una especificación de la clase módulo que está en Intermedia. Dentro de Intermedia se encuentran submódulos que encapsulan funcionalidades afines, los paquetes General, Negocio, Interfaces y Parámetros constituyen estos submódulos.

Chesys constituye el agente integrador del simulador, este componente contiene la ventana principal de la aplicación con los elementos gráficos que permiten la manejabilidad del simulador, como son menús, barras de herramientas y de estado, la paleta de componentes que muestra los modelos matemáticos disponibles para simular, así como la posibilidad de acceso a las funcionalidades adicionales del simulador.

La simulación genera gran cantidad de información que debe ser presentada al usuario de forma coherente, con este fin se creó el componente **Reportes**, que contiene los elementos que permiten el procesamiento de la información obtenida de la simulación para presentársela al usuario de la forma más

apropiada, incluso posibilitando que este la pueda personalizar. Aquí se usaron las facilidades que ofrece el Crystal Reports (CR), el cual es una potente herramienta para la confección de reportes. El CR está especialmente orientado a información proveniente de bases de datos, por lo que fue necesaria la adaptación de los resultados del simulador para poder usar las facilidades de CR, en este componente entonces se definen los mecanismos que permiten dicha adaptación. Reportes permite además la confección de reportes específicos y especializados que son característicos de las distintas industrias que se estén simulando.

Para el tratamiento de los ficheros de la aplicación se concibió el componente **SalvarCargar**. La plataforma .NET ofrece una serie de métodos para el trabajo con ficheros y la serialización de objetos, en SalvarCargar se usa este potente mecanismo y se definen un conjunto de clases que permiten salvar los proyectos de simulación creados con la aplicación y cargarlos posteriormente.

El diseño debe garantizar que el sistema soporte todos los requisitos, funcionales y no funcionales. La mejor forma de asegurarse de que se están cumpliendo con todos los requisitos es diseñar basándose en los casos de uso, es decir, hacer las realizaciones de los de uso en el diseño. En lo que sigue se verán las clases de diseño fundamentales que dan solución a los casos de uso más significativos.

Realización caso de uso Crear DFI.

Un diagrama de flujo de información (DFI) es el elemento visual más importante de la simulación. Está formado por tres elementos fundamentales, los módulos, los puertos y las conexiones. Los módulos son las representaciones de los equipos reales de la fábrica, los puertos son los puntos por los que se conectan los módulos unos con otros y las conexiones son la líneas que conectan a los puertos. Para la definición de estas clases se usaron las funcionalidades del Framework.Net, todos estos elementos se definieron como controles de usuario. En la persecución de este objetivo se estableció una jerarquía de clases que empieza en la clase CtrMovil, la cual agrupa las funcionalidades para poder arrastrar un control sobre un panel en tiempo de ejecución. Siguiendo la descendencia jerárquica aparecen las clases CtrModulo y CtrPuertoEnlace, CtrModulo tiene todos los elementos que le permiten representar a un equipo real, la imagen representativa del equipo real, el menú contextual para acceder a diferentes operaciones que se puedan efectuar sobre el módulo, la lista de imágenes que aparecerán en los distintos estados, así como la definición de estos estados, contiene también una lista con la referencia a sus puertos y un atributo para relacionarlo con la parte del negocio. En la otra rama del árbol jerárquico,

CtrPuertoEnlace contiene las funcionalidades comunes de los puertos y conexiones, esto es, los puertos y líneas se pueden conectar entre ellos, hasta cierto punto se pueden tratar como elementos del mismo tipo, pues tienen que responder de la misma manera a la hora de conectarse, moverse, notificar al control adyacente de su cambio de posición o tamaño, etc. Cada CtrPuertoEnlace contiene las referencias de los enlaces que tiene a cada lado y es capaz de notificar a estos elementos sus cambios de estado, así como propagar información visual que es común a un par de puertos y la conexión que los une como es el nombre de la corriente. Siguiendo la jerarquía por CtrPuertoEnlace se encuentran CtrPuerto y CtrEnlace, la primera define las características que diferencian a los puertos de las conexiones y contiene la imagen del puerto, el menú contextual y una referencia al módulo al que pertenece, también tiene un atributo para referirse al negocio del puerto. En esta clase se definen todos los tipos de puertos posibles: entrada, salida, seudo entrada, seudo salida, lector y escritor, así como los permisos de conexión entre ellos. CtrEnlace encapsula las propiedades y funcionalidades que definen a las conexiones en general, dejando los detalles a CtrEnlaceHorizontal y CtrEnlaceVertical que son la que finalmente representan los dos tipos de líneas que se pueden encontrar en un diagrama. Otra clase fundamental necesaria para garantizar la construcción de un DFI es el panel sobre el que se grafica el diagrama, aquí entra en escena pnlDiagrama otro control de usuario, pero que en este caso hereda de la clase Panel del Framework.Net, aquí se definen los mecanismos para distintas funcionalidades del editor gráfico como cortar, copiar y pegar módulos, las acciones de hacer y deshacer entre otras. Los equipos arrastrados hacia el diagrama son almacenados en el arreglo módulos. En pnlDiagrama se implementa el importante método Simular, función que itera por cada uno de los módulos mandándolos a calcularse; también define la función Ordenar que halla automáticamente el orden de cálculo del DFI. La figura 2 muestra el diagrama de clases correspondiente a la realización de este caso de uso en el diseño.

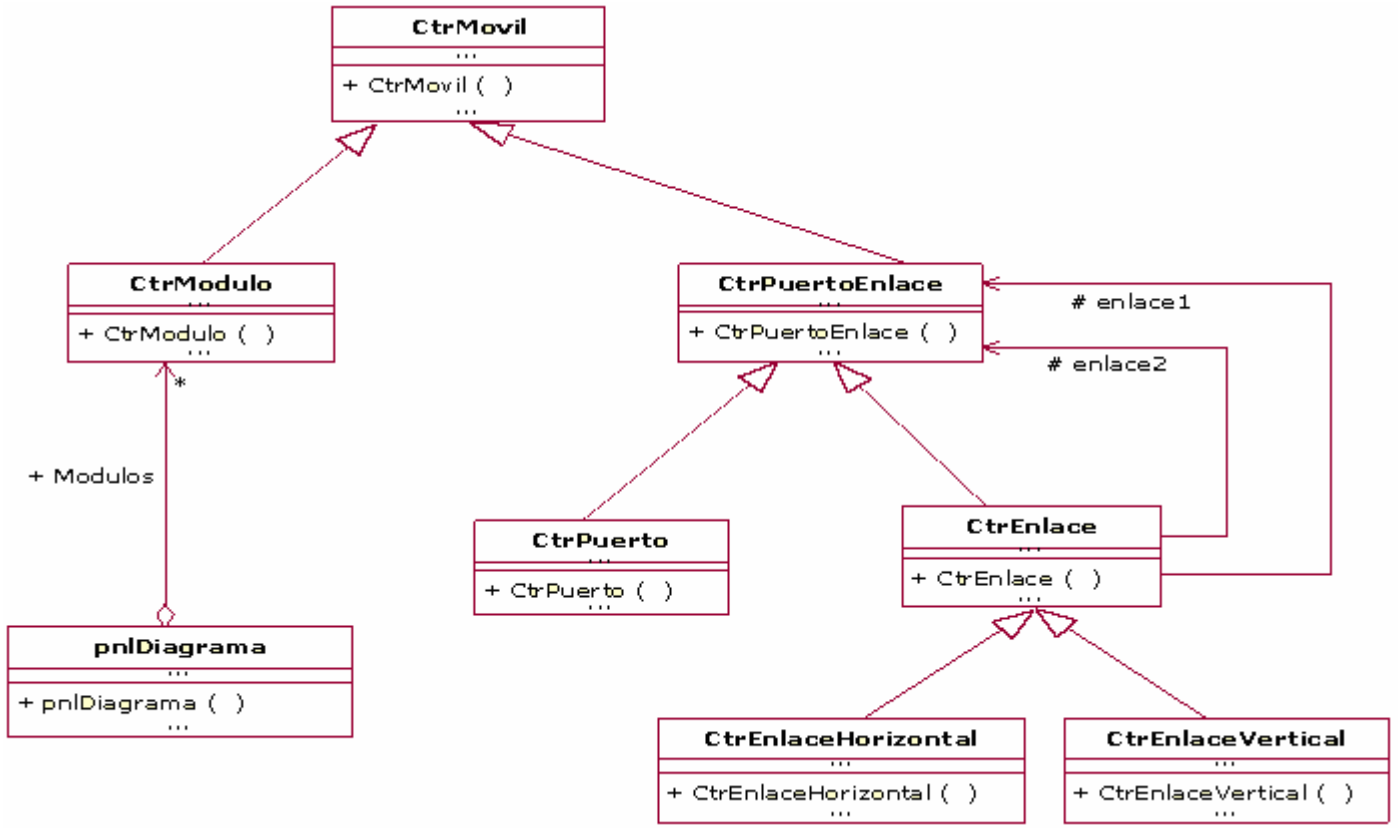

Figura 2: Realización caso de uso Crear DFI. Clases del diseño.

Nombre: CtrMovil	**Tipo:** Control de usuario	
Descripción: Clase que encapsula las funcionalidades de movimiento de los controles en un área.		
Atributo	**Tipo**	**Visibilidad**
Arrastrar	bool	protected
PuntoFijo	Point	protected
id	string	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
getID	string	public
Descripción: Obtiene el identificador de un control.		
setID	void	public
Descripción: Establece el identificador de un control.		
GetImage	Image	public
Descripción: Obtiene una imagen del control para ser exportado en un formato de imagen		

Tabla 1: Descripción de la clase CtrMovil.

Nombre: CtrModulo
Descripción: Es una abstracción de un equipo real, representa las características gráficas de un equipo.

Atributo	**Tipo**	**Visibilidad**
colorNormal	Color	private
colorSeleccionado	Color	private
NegocioModulo	NegModulo	public
seleccionado	bool	private
estado	eEstados	private
aRotacion	eAngulo	private
fliped	bool	private
mensaje	string	private
transformaciones	ArrayList	private
ventana	frmPropiedadesModulo	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
IniModulo()	void	protected
Descripción: Inicializa los elementos de la clase, establece la conexión con el NegModulo correspondiente y los puertos del módulo.		
MostrarInformacion()	void	public
Descripción: Muestra los datos contenidos en el módulo.		
CargarModulo()	void	public
Descripción: Reconstruye un objeto a partir de un fichero de datos.		
Get_ModuloSerializado()	NegModuloSerializable	public
Descripción: Obtiene un prototipo del módulo con los datos necesarios para reconstruirlo posteriormente.		
CargarImagen()	void	public
Descripción: Carga la imagen correspondiente al estado en que esta el equipo.		
Calcular()	void	public
Descripción: Calcula los parámetros del módulo y sus corrientes de salida.		
Eliminarme()	void	public
Descripción: Elimina un objeto.		
Seleccionar()	void	public
Descripción: Marca el módulo como seleccionado		
SetModuloNegPuerto()	void	public
Descripción: Establece la conexión entro los puertos gráficos con sus negocio puertos.		
Identificador()	bool	public
Descripción: Establece el identificador del equipo y especifica si pudo cambiarlo o no.		
MostrarIdentificador()	void	public
Descripción: Hace visible la etiqueta que identifica el módulo.		
MostrarVentana()	void	public
Descripción: Muestra la ventana de entrada de datos al equipo.		
ShowInfo()	void	public
Descripción: Muestra los datos contenidos en el negocio módulo.		
ObtenerInformacionPuertos()	InfoTemp[]	private
Descripción: Obtiene la información contenida en todos sus puertos.		
FlipModulo()	void	public
Descripción: Hace un espejo vertical del equipo.		
GetInfo()	NegInfo[]	public
Descripción: Obtiene los datos almacenados en el negocio módulo.		
CambiarAngulo()	void	private
Descripción: Actualiza el ángulo de rotación correspondiente al estado del equipo cuando se le aplica una transformación.		

Rotar90()	void	public
Descripción: Hace un espejo vertical de la imagen del equipo.		
ActualizarConexiones()	void	private
Descripción: Actualiza las conexiones del equipo después de cualquier transformación.		
CargarImagenListo()	void	public
Descripción: Chequea si el equipo esta listo para calcular y carga la imagen correspondiente a este estado.		
PuertosListos()	bool	private
Descripción: Chequea si todos los puertos están listos para calcular.		
Ayuda()	void	public
Descripción: Muestra la ayuda del equipo.		
SetNombrePuertos()	void	public
Descripción: Establece un nombre especificado a todos los puertos del equipo.		
CargarSTA()	void	public
Descripción: Entra los datos del módulo a partir de un fichero de STA.		
SetNewModulo()	void	public
Descripción: Establece como negocio módulo del equipo uno especificado.		
Seleccionado	bool	public
Descripción: Obtiene o establece el estado de selección del equipo.		
GetPuertos	CtrPuerto[]	public
Descripción: Devuelve un arreglo con los puertos del equipo.		
Estado	eEstados	public
Descripción: Obtiene o establece el estado en que esta el equipo.		
Nombre	string	public
Descripción: Obtiene o establece el nombre del equipo.		
Ventana	frmPropiedadesModulo	public
Descripción: Obtiene o establece la ventana asociada al equipo.		
Fliped	bool	public
Descripción: Obtiene o establece el estado de flip del equipo.		
Transformaciones	eTransformaciones[]	public
Descripción: Obtiene o establece las transformaciones del equipo.		
GetImage	Image	public
Descripción: Obtiene una imagen como representación de todo el equipo.		
btnAceptar_Click()	void	public
Descripción: Método que se subscribe al evento OnClick del botón aceptar de la forma asociada al equipo.		

Tabla 2: Descripción de la clase CtrModulo.

Nombre: CtrPuertoEnlace	**Tipo: control de usuario**	
Descripción: Clase que describe los comportamientos comunes entre enlaces y puertos		
Atributo	**Tipo**	**Visibilidad**
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
RefrescarNombre ()	void	public
Descripción: Actualiza el nombre del enlace o puerto.		
Enlazar()	void	public
Descripción: Enlaza un enlace con otro especificado. Hace el enlace en las dos direcciones.		
EnlazarBack()	void	protected

Descripción: Enlaza un enlace con otro especificado. Hace el enlace en una sola dirección.		
DesEnlazar()	void	public
Descripción: Desenlaza un enlace con otro especificado. Lo hace en las dos direcciones.		
DesEnlazarBack()	void	protected
Descripción: Desenlaza un enlace con otro especificado. Lo hace en una sola dirección.		
Notificarme()	void	public
Descripción: Permite notificar entre enlaces los cambios de posición, tamaño, etc.		
Nombre	string	public
Descripción: Obtiene o establece el nombre de la conexión.		
getLargo()	int	public
Descripción: Obtiene el largo del enlace.		
GetAlgunPunto()	Point	public
Descripción: Obtiene el punto por donde se conecta visualmente un enlace.		

Tabla 3: Descripción de la clase CtrPuertoEnlace.

Nombre: CtrPuerto	**Tipo:** Control de usuario	
Descripción: Clase que encapsula las funcionalidad y propiedades de un puerto.		
Atributo	**Tipo**	**Visibilidad**
puerto	CtrPuerto	private
enlace	CtrEnlace	private
tipo	eTipoPuerto	private
sentido	eSentidoPuerto	private
corriente	eTipoCorriente	private
ventanaSiempre	bool	private
negocioPuerto	NegPuerto	private
ConexionIniciada	CtrPuerto	public
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
IniFlechasGraficas()	void	private
Descripción: Establece el color y forma de la imagen del puerto de acuerdo a su estado.		
RefrescarNombre()	void	public
Descripción: Actualiza el nombre su nombre y el de toda la conexión asociada.		
AceptarValor()	void	public
Descripción: Le asigna a su negocio puerto el valor especificado.		
EntregarValor()	void	public
Descripción: Entrega al puerto con el que esta conectado su valor. Garantiza la comunicación.		
EliminarEnlace()	void	public
Descripción: Se elimina a si mismo y notifica para que se elimine la conexión asociada.		
Enlazar()	void	public
Descripción: Enlaza un puerto con un enlace especificado y el enlace con el puerto.		
EnlazarBack()	void	protected
Descripción: Enlaza un puerto con un enlace especificado.		
DesEnlazarBack()	void	protected
Descripción: Desenlaza un puerto del enlace especificado.		
SePuedeConectarCon()	bool	private
Descripción: Chequea si el puerto se puede conectar con un puerto especificado.		
ConectableCon()	bool	private

Descripción: Chequea que el tipo del puerto y de un puerto especificado sean compatibles para ser conectados.		
IgualTipoDeCorriente()	bool	public
Descripción: Chequea el negocio módulo del puerto y el de un puerto especificados son compatibles.		
Vincular()	bool	protected
Descripción: Vincula el puerto con un puerto especificado y viceversa.		
VincularBack()	void	protected
Descripción: Vincula el puerto con un puerto especificado.		
DesVincular()	void	protected
Descripción: Desvincula el puerto del puerto especificado y viceversa.		
DesVincularBack()	void	protected
Descripción: Desvincula el puerto del puerto especificado.		
GetNuevoPunto()	Point	private
Descripción: Devuelve el punto de conexión del puerto.		
Conectar()	bool	public
Descripción: Conecta el puerto con un puerto especificado.		
Notificarme()	void	public
Descripción: Notifica el puerto de cambios en la conexión asociada o el equipo.		
RefrescarLabel()	bool	public
Descripción: Actualiza la etiqueta que muestra el nombre asociado a la corriente que manipula.		
ObtenerVentana()	frmPropiedades	public
Descripción: Obtiene la ventana asociada al puerto.		
MostrarVentana()	void	public
Descripción: Muestra la ventana asociada al puerto,		
ResetNegocio()	void	public
Descripción: Resetea el negocio puerto asignándole valor vacío.		
GetInfo()	NegInfo[]	public
Descripción: Obtiene la información contenida en el negocio puerto.		
Nombre	string	public
Descripción: Obtiene o establece el nombre del puerto.		
ActualizarConexion()	void	public
Descripción: Actualiza la conexión asociada.		
Ventana	frmPropiedades	public
Descripción: Obtiene y establece la ventana asociada al puerto.		
Enlace	CtrEnlace	public
Descripción: Obtiene y establece el enlace continuo al puerto.		
GetPunto	Point	public
Descripción: Obtiene el punto de conexión del puerto.		
Sentido	eSentidoPuerto	public
Descripción: Obtiene y establece el sentido del puerto, cuando se establece el sentido llama al método IniFlechasGraficas()		
TipoCorriente	eTipoCorriente	public
Descripción: Obtiene y establece el tipo de corriente del puerto, cuando se establece el sentido llama al método IniFlechasGraficas()		
Tipo	eTipoPuerto	public
Descripción: Obtiene y establece el tipo del puerto, cuando se establece el sentido llama al método IniFlechasGraficas()		
Mensaje	string	public
Descripción: Obtiene y establece mensaje asociado al puerto.		
Puerto	CtrPuerto	public

Descripción: Obtiene el puerto con el que esta conectado.		
GetImage	Image	public
Descripción: Obtiene la imagen que representa al puerto.		

Tabla 4: Descripción de la clase CtrPuerto.

Nombre: CtrEnlace	**Tipo:** Control de usuario	
Descripción: Clase que encapsula las funcionalidades y propiedades comunes de los enlaces que componen una conexión.		
Atributo	**Tipo**	**Visibilidad**
colorNormal	Color	private
colorSeleccionado	Color	private
colorTenue	Color	private
enlace1	CtrPuertoEnlace	protected
enlace2	CtrPuertoEnlace	protected
ancho	int	private
oculto	bool	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
RestaurarColor()	void	protected
Descripción: Establece el color del enlace de acuerdo a un estado anterior.		
RefrescarNombre()	void	public
Descripción: Actualiza el nombre de la conexión.		
UbicarLabel()	void	protected
Descripción: Posiciona la etiqueta que muestra el nombre de la conexión.		
Seleccionar()	void	public
Descripción: Selecciona el enlace.		
SeleccionarBack()	void	protected
Descripción: Selecciona toda la conexión a que pertenece el enlace.		
OcultarEnlace()	void	public
Descripción: Oculta la conexión a que pertenece el enlace.		
MostrarEnlace()	void	public
Descripción: Muestra la conexión a que pertenece el enlace.		
EliminarEnlace()	void	public
Descripción: Elimina el enlace y toda la conexión.		
Unir()	void	protected
Descripción: Relaciona el enlace y otro especificado mutuamente.		
UnirBack()	void	protected
Descripción: Relaciona el enlace con un enlace especificado.		
EvitarNulos()	void	protected
Descripción: Garantiza que cuando se cruce la conexión sobre si misma o se superpongan enlaces estos se eliminen.		
Get_EnlaceSerializado()	NegLineaSerializable	public
Descripción: Obtiene un prototipo del enlace para ser guardado en un fichero.		
Cargar_Linea()	void	public
Descripción: Reconstruye un enlace a partir de un objeto NegLineaSerializable recuperado de un fichero.		
EstaConectado()	bool	public
Descripción: Devuelve si el enlace esta conectado o no.		
Nombre	string	public

Descripción: Obtiene o establece el nombre de la conexión.		
RefrescarLabel()	bool	public
Descripción: Actualiza el texto de la etiqueta que nombra la conexión.		
ColorNormal	Color	public
Descripción: Propiedad estática que obtiene y establece el color considerado normal para todos los objetos CtrEnlace.		
ColorTenue	Color	public
Descripción: Propiedad estática que obtiene y establece el color que toman los enlaces cuando están ocultos.		
ColorSeleccionado	Color	public
Descripción: Propiedad estática que obtiene y establece el color que toman los enlaces cuando están seleccionados.		
Ancho	int	public
Descripción: Propiedad estática que obtiene y establece el ancho de todos los objetos de tipo enlace.		
GetPunto1	Point	public
Descripción: Obtiene el punto inicial del enlace.		
GetPunto2	Point	public
Descripción: Obtiene el punto final del enlace.		
GetImage	Image	public
Descripción: Obtiene una imagen que representa al enlace		
SetPosicionLongitud()	void	protected
Descripción: Establece la posición y el tamaño del enlace a partir de los puntos de inicio y fin.		

Tabla 5: Descripción de la clase CtrEnlace.

Nombre: CtrEnlaceVertical	**Tipo:** Control de usuario	
Descripción: Es una especialización de la clase CtrEnlace, en ella se define el comportamiento especifico de los enlaces verticales.		
Atributo	**Tipo**	
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
UbicarLabel()	void	public
Descripción: Sobrescribe el método de CtrEnlace, definiendo como ubicar la etiqueta del nombre en los enlaces verticales.		
SetPosicionLongitud()	void	public
Descripción: Sobrescribe el método de CtrEnlace, define como posicionar un enlace vertical.		
Unir()	void	public
Descripción: Especifica como vincular un enlace vertical con un enlace especificado y viceversa.		
UnirBack()	void	public
Descripción: Especifica como vincular un enlace vertical con un enlace especificado**.**		
Notificarme()	void	public
Descripción: Establece que acciones cuando el enlace es notificado de algún cambio en sus enlaces adyacentes.		
Largo	int	public
Descripción: Obtiene la longitud del enlace		
GetPunto1	Point	public
Descripción: Obtiene el punto de inicio del enlace.		
GetPunto2	Point	public

Descripción: Obtiene el punto de fin del enlace.

Tabla 6: Descripción de la clase CtrEnlaceVertical.

Nombre: CtrEnlaceHorizontal	**Tipo:** Control de usuario	
Descripción: Es una especialización de la clase CtrEnlace, en ella se define el comportamiento especifico de los enlaces horizontales.		
Atributo	**Tipo**	
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
UbicarLabel()	void	public
Descripción: Sobrescribe el método de CtrEnlace, definiendo como ubicar la etiqueta del nombre en los enlaces horizontales.		
SetPosicionLongitud()	void	public
Descripción: Sobrescribe el método de CtrEnlace, define como posicionar un enlace horizontal.		
Unir()	void	public
Descripción: Especifica como vincular un enlace horizontal con un enlace especificado y viceversa.		
UnirBack()	void	public
Descripción: Especifica como vincular un enlace horizontal con un enlace especificado**.**		
Notificarme()	void	public
Descripción: Establece que acciones cuando el enlace es notificado de algún cambio en sus enlaces adyacentes.		
Largo	int	public
Descripción: Obtiene la longitud del enlace		
GetPunto1	Point	public
Descripción: Obtiene el punto de inicio del enlace.		
GetPunto2	Point	public
Descripción: Obtiene el punto de fin del enlace.		

Tabla 7: Descripción de la clase CtrEnlaceHorizontal.

Nombre: pnlDiagrama	**Tipo:** Control de usuario	
Descripción: Clase que representa el área sobre la que se grafican los DFI, es la clase contenedora de módulos y conexiones.		
Atributo	**Tipo**	**Visibilidad**
adicionando	bool	private
posicion	Point	private
url	string	private
mycursor	Cursor	private
identificador	CtrIdentificador	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
AddModuloDiagramaNormal()	void	private
Descripción: Agrega un nuevo módulo al diagrama.		
EliminarModulo()	void	public
Descripción: Elimina el módulos seleccionado en el diagrama.		
CopiarParaPapelera()	void	public

Descripción: Copia el módulo seleccionado a la papelera.		
CortarParaPapelera()	void	public
Descripción: Corta el módulo seleccionado a la papelera.		
Simular()	void	public
Descripción: Método que itera por los módulos contenidos en el diagrama mandándolos a calcular.		
Ordenar()	void	private
Descripción: Método que ordena los módulos del diagrama antes de simular para garantizar que los cálculos sean fiables.		
ModuloSeleccionado	CtrModulo	public
Descripción: Obtiene el módulo que esta seleccionado en el diagrama.		
pnlDiagrama_ControlAdded()	void	public
Descripción: Define las acciones a realizar cuando se le agrega un nuevo control al diagrama.		
pnlDiagrama_ControlRemoved()	void	public
Descripción: Define las acciones a realizar cuando se elimina un control al diagrama.		
IdentificarDiagrama()	void	public
Descripción: Permite establecer la identificación del diagrama.		
Pegar()	Void	public
Descripción: Adiciona al diagrama el módulo, copiado o cortado y que se encuentra en el Clipboard.		
EliminarTodo()	Void	public
Descripción: Elimina todos los controles que estén sobre el diagrama.		
URL	string	public
Descripción: Obtiene o establece la url donde esta salvado el proyecto que representa el diagrama.		
Identificador	CtrIdentificador	public
Descripción: Obtiene o establece el identificador del diagrama.		
Conexiones	Connection[]	public
Descripción: Obtiene todas las conexiones que hay en el diagrama.		
Posicion	Point	public
Descripción: Obtiene la posicion del diagrama.		
IdentificadorVisible	bool	public
Descripción: Establece la visibilidad del identificador del diagrama.		
AdicionandoModulos	bool	public
Descripción: Obtiene o estable el estado del diagrama relacionado con la adición de módulos.		

Tabla 8: Descripción de la clase pnlDiagrama.

Todas estas clases colaboran para soportar todos los escenarios del caso de uso Crear DFI. A continuación se muestra un diagrama de secuencia que representa el escenario específico de agregar un módulo al diagrama.

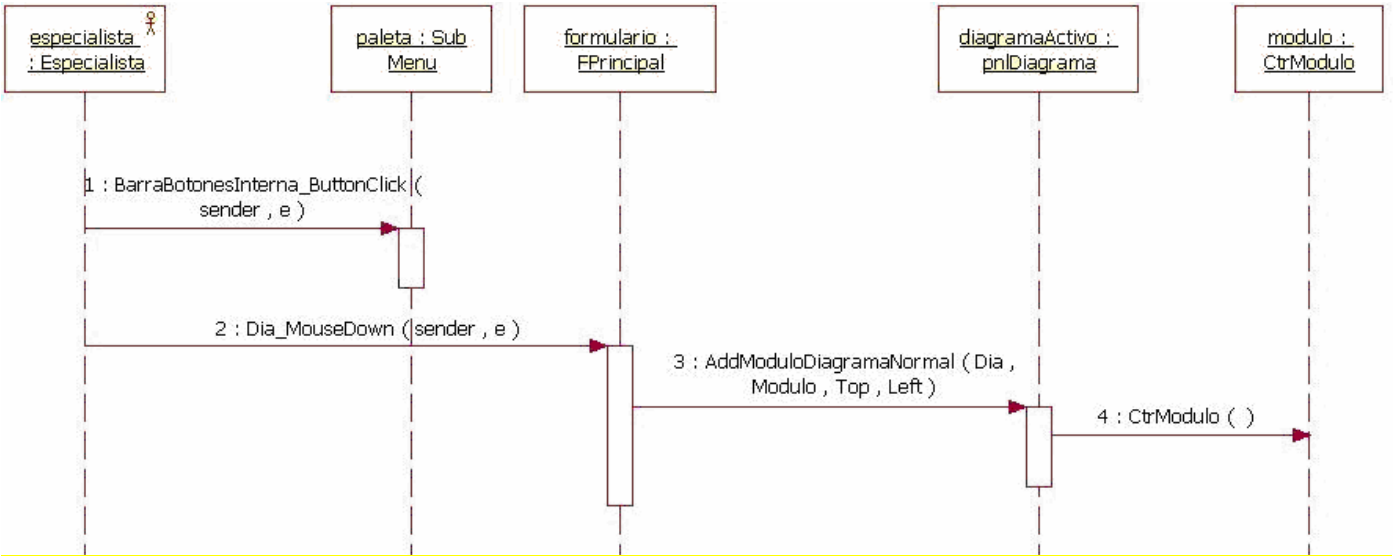

Figura 3: Realización caso de uso Crear DFI. Diagrama de secuencia.

El especialista selecciona en la paleta de módulos el que desea agregar al diagrama, después de hacer su elección da click sobre el área de dibujo, el panel de dibujo detecta la acción de click y crea un nuevo módulo y lo muestra en la posición donde se produjo el impacto del puntero del Mouse.

Realización caso de uso Simular.

La misión principal de la simulación es el cálculo de los parámetros de las corrientes de salida y de los parámetros adicionales de los módulos a partir de las corrientes de entrada y los parámetros particulares de los módulos. En primer lugar se debe tener un soporte de datos para almacenar toda esa información, esto no significa que se este pensando ya en una base de datos relacional o algo por el estilo, puede ser cualquier estructura como una lista o un arreglo. Hasta este punto se necesitan mecanismos para almacenar datos cuya existencia no tiene que ser obligatoriamente persistente, estos se pueden almacenar en la memoria de trabajo mientras se opera sobre ellos. Las clases que contendrán los datos de entrada y salida del simulador son los módulos y corrientes. Para esto se creó en primer lugar la estructura (struct) doublez, la cual define el tipo de datos con el que opera todo el sistema para hacer los cálculos, este tiene las propiedades Valor, que se refiere al valor numérico, Magnitud que dice la magnitud asociada a ese valor y UM la unidad de medida específica, esto se hizo con el objetivo de aumentar la calidad y completitud de la información que se maneja. Las clases creadas para manejar la información fueron NegModulo y NegPuerto, ambas tienen la propiedad pública Valor de tipo NegModulo y NegPuerto

respectivamente, mediante la cual se puede acceder a los datos de cada uno de ellos. Cada módulo y puerto tendrá un conjunto de parámetros que los diferenciarán de los demás, estos parámetros se definirán en el momento que se estén creando nuevos equipos y corrientes agregándoles atributos y propiedades. En el caso de los que sean reales o enteros se definirán del tipo doublez.

La simulación consiste en la elaboración de modelos matemáticos de los equipos, representando una aproximación de su comportamiento. Dichos modelos son implementados en la clase NegModulo. Esta clase tiene una serie de métodos y propiedades entre los que se encuentra Calcular, en el que se llama el método CalcularModulo (en el que se programan los modelos) y manda a cada uno de los puertos de salida a pasar su valor al puerto con el que está conectado. Los tipos de corrientes que aparecen en el simulador vienen dadas por la tecnología que se este simulando, en la primera etapa de desarrollo se incluyen los modelos de la industria azucarera, las muchas corrientes que existen en esta industria se pueden clasificar en dos grandes familias, flujos azucarados y no azucarados, la diferencia está en la cantidad de parámetros que contiene la corriente. Ante esta situación se definieron dos clases que heredan de NegPuerto, las cuales tienen ya establecidos los parámetros que lleva cada tipo de corriente, estas fueron NegPuertoAgua y NegPuertoAzucarado. De esta manera la creación de cualquier flujo del proceso de producción de azúcar se obtiene heredando de una de estas dos clases.

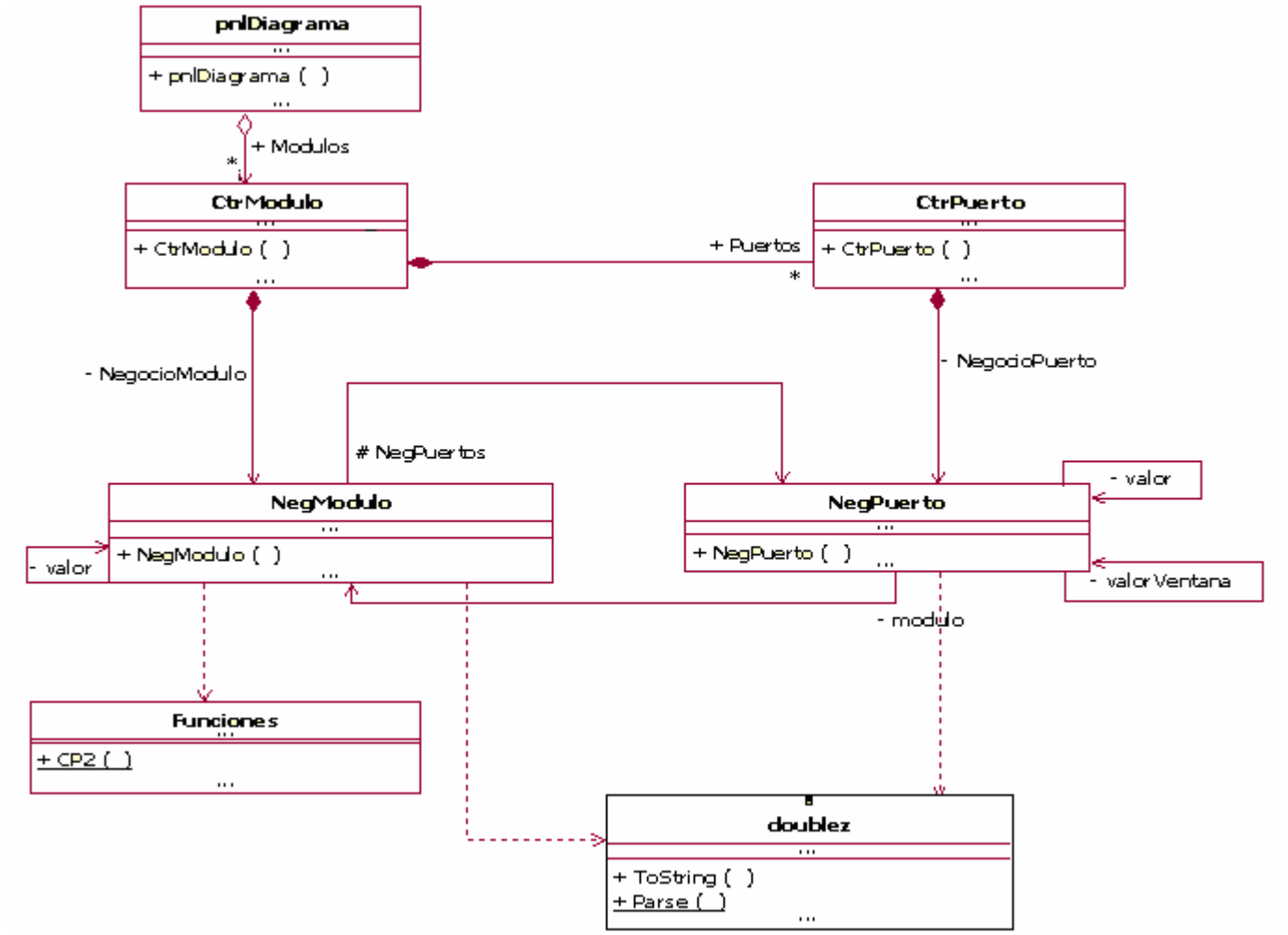

Figura 4: Realización caso de uso Simular. Clases del diseño.

Nombre: NegModulo		**Tipo:** clase
Descripción: Provee a los módulos del las campos y métodos necesarios para almacenar y manipular sus valores, así como las ecuaciones y cálculos matemáticos de los mismos.		
Atributo	**Tipo**	**Visibilidad**
NegPuertos	NegPuerto	private
name	string	private
valor	NegModulo	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
Calcular()	bool	public

Descripción: Se encarga de recorrer todos los puertos y llamar a la simulación de cada uno de ellos y además simular el módulo, calculando de esta manera los valores de la simulación. Este método es virtual y se redefine en cada módulo, el cual lo adapta a su forma de simular en concreto.		
CalculoModulo()	void	protected
Descripción: Calcula a través de los modelos matemáticos y los parámetros del módulo, y lleva a cabo el proceso de la simulación en el módulo. Este método es virtual.		
Compatibles()	bool	public
Descripción: Determina cuando un módulo tiene compatibilidad con otro, este método es virtual, puesto que cada módulo define aquellos que se podrán intercambiar información con el.		
GetInfo()	NegInfo	public
Descripción: Devuelve un arreglo de clases de tipo NegInfo que contiene la información de cada uno de los parámetros particulares del módulo y de sus puertos.		
Listo()	bool	public
Descripción: El valor que devuelve indica si el módulo esta listo o no para calcularse. Cada módulo en especifico tiene su propio concepto de listo por lo tanto el método es virtual.		

Tabla 9: Descripción de la clase NegModulo.

Nombre: NegPuerto		**Tipo:** clase
Descripción: Provee a los puertos de los campos y métodos necesarios para almacenar y manipular sus valores, así como las ecuaciones y cálculos matemáticos de los mismos.		
Atributo	**Tipo**	**Visibilidad**
módulo	NegModulo	private
name	string	private
valor	NegPuerto	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
Clone()	object	public
Descripción: Realiza una copia exacta de los valores contenidos en el negocio puerto.		
Compatibles()	bool	public
Descripción: Determina si otro puerto es compatible con el puerto.		
GetInfo	NegInfo	public
Descripción: Devuelve un arreglo de clases de tipo NegInfo que contiene la información de cada uno de los parámetros particulares del módulo y de sus puertos.		
Listo	bool	public
Descripción: Determina si el puerto esta listo con información o alguna conexión para ser calculado en la simulación.		
Resetear	void	public

Descripción: Elimina los valores almacenados en el puerto en ese momento.

Tabla 10: Descripción de la clase NegPuerto.

Nombre: Funciones		**Tipo:** Clase
Descripción: Contiene todos la funciones implementadas que son comunes a los módulos, y que su utilizan el cálculo de propiedades químicas.		
Atributo	**Tipo**	**Visibilidad**
-	-	-
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
CP2()	double	public
Descripción: Calcula de capacidad calorífica de las corrientes azucaradas.		
HJ2()	double	public
Descripción: Calcula de Entalpía específica de las corrientes azucaradas.		
Psat2()	double	public
Descripción: Calcula la Presión de Saturación		
EPEB()	double	public
Descripción: Calcula de la elevación del punto de ebullición		
TempSaturacion()	double	public
Descripción: Cálculo de los Temperatura de saturación		
EntalpiaLiquidoSaturado()	double	public
Descripción: Calcula la entalpía de liquido saturado(solo para el agua)		
EntropiaLiquidoSaturado()	double	public
Descripción: Cálculo de la entropía de liquido saturado(solo para el agua)		
EntalpiaEspecificaVaporSat()	double	public
Descripción: Calcula de la entalpía del vapor saturado(solo para el agua)		
EntropiaEspecificaVaporSat()	double	public
Descripción: Calcula la entropía del vapor saturado(solo para el agua)		
EntalpiaEspecificaVaporSobreCal()	double	public
Descripción: Cálculo de la entalpía del vapor sobrecalentado(solo para el agua)		
EntropiaEspecificaVaporSobreCal()	double	public
Descripción: Cálculo de la entropía del vapor sobrecalentado(solo para el agua)		
Stabl2()	void	public
Descripción: Tabla de vapor 2		
Pureza()	double	public
Descripción: Calcula de la Pureza		
Brix()	double	public
Descripción: Cálculo del Brix		

Tabla 11: Descripción de la clase Funciones.

El flujo de eventos que se lleva a cabo para cumplir la operación de simular por la clases de diseño se muestran en el diagrama de secuencia de la figura 5

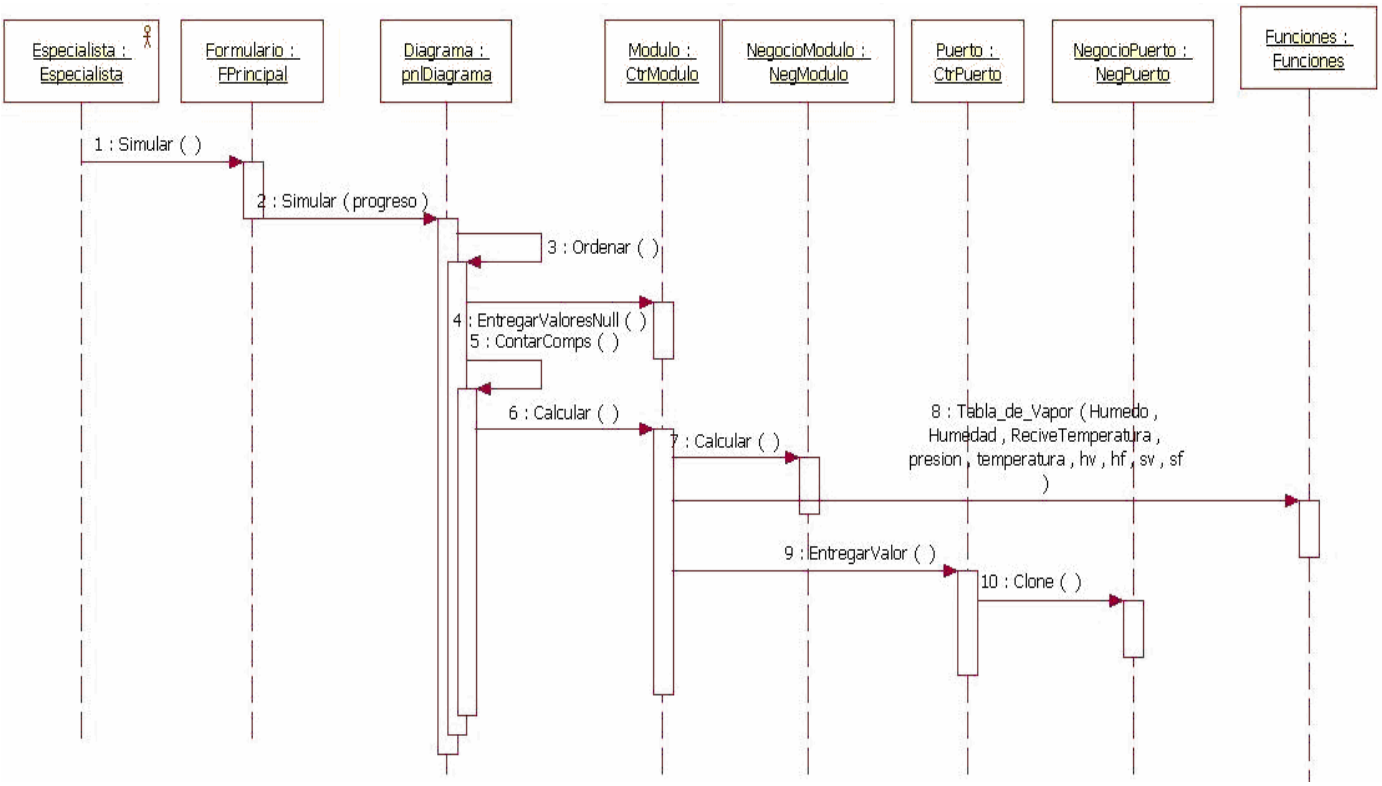

Figura 5: Realización caso de uso Simular. Diagrama de secuencia.

El usuario solicita a la aplicación que desea simular un diagrama determinado, la interfaz solicita al pnlDiagrama que comience el proceso de simulación. El diagrama ordena los módulos en un orden de cálculo lógico, carga los datos suministrados y pasa por los módulos ordenados llamando al método calcular de cada uno de ellos, el avance de la simulación es mostrado al usuario a través de una barra de progreso.

Realización caso de uso Insertar Información.

Entre el simulador y el usuario de este se establece una relación bien fuerte de intercambio de información. Si bien como resultado de los cálculos se devuelve un volumen de datos importante es necesario que el usuario introduzca un conjunto de datos iniciales para hacer esos cálculos. Ya teniendo el soporte de los datos (NegModulo y NegPuerto) es necesario definir el mecanismo de intercambio. Para

esto se crearon dos formularios generales de los cuales se hereda posteriormente para definir la ventana de entrada de datos a los módulos y puertos. Estos formularios contienen la propiedad virtual SetGetValores del tipo NegPuerto y NegModulo, dependiendo de si es la de puertos o la de módulos, esta es invocada cuando en tiempo de ejecución se muestra el formulario y cuando se presiona el botón aceptar de este. Las clases CtrModulo y CtrPuerto tienen definida una propiedad para vincularlos con la ventana de entrada de datos, y tienen definido el método MostrarVentana en el cual se invoca el formulario y se le pasa el NegPuerto o NegModulo asociado y se suscribe al evento Click del botón Aceptar de la ventana, de esta manera se garantiza que cuando se muestre la ventana esta refleje los datos que están almacenados y que cuando se presione el botón Aceptar los datos introducidos por el usuario se almacenen.

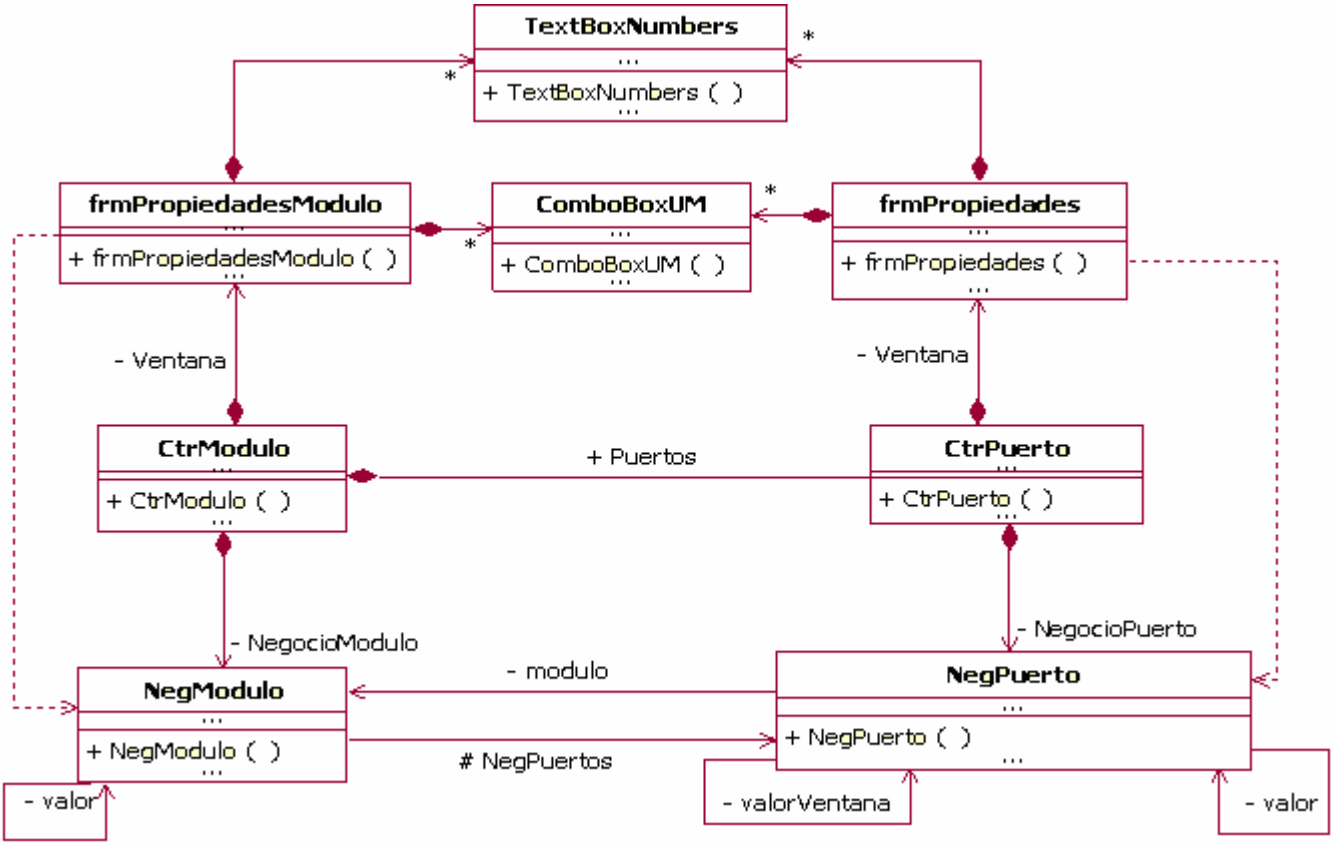

Figura 6: Realización caso de uso Insertar Información. Clases del diseño.

Nombre: TextBoxNumber	**Tipo:** Control de usuario	
Descripción: Control de usuario creado para la entrada de datos, garantiza que los datos numéricos tengan el formato correcto.		
Atributo	**Tipo**	**Visibilidad**
porciento	bool	private
negativos	bool	private
enteros	bool	private
mensaje	string	private
MostrarMensaje	string	private
Error	eTipoError	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Solo_Porciento()	bool	private
Descripción: Determina si la cadena entrada por el usuario es fraccionaria o no.		
Si_Pegan()	bool	private
Descripción: Validar que los datos pegados desde el Clipboard sean numéricamente válidos.		
Solo_Numeros()	bool	private
Descripción: Valida que la cadena introducida sean un número.		
ActualizarError()	void	private
Descripción: Actualiza el atributo error de acuerdo al estado del componente.		
ActualizarMensaje()	void	private
Descripción: Actualiza el mensaje que muestra el control cuando es señalado con el puntero.		
OnKeyPress()	void	private
Descripción: Especifica las acciones a realizar cuando se presiona una tecla en el área de edición de control.		
TextBoxNumbers_TextChanged()	void	private
Descripción: Define las acciones que se realizan cuando cambia el texto del control.		
PermitirNegativos	bool	public
Descripción: Obtiene y establece el valor del atributo negativos. Cuando este atributo esta en falso no se pueden introducir números negativos.		
Porciento	bool	public
Descripción: Obtiene y establece el valor del atributo porciento. Cuando esta en verdadero el control solo acepta valores entre 0 y 1.		
SoloEnteros	bool	public
Descripción: Obtiene y establece el valor del atributo enteros. Si esta en verdadero solo acepta números enteros.		
Valor	void	public
Descripción: Obtiene el valor numérico del dato introducido en el control.		

Tabla 12: Descripción de la clase TextBoxNumber.

Nombre: frmPropiedades	**Tipo:** formulario	
Descripción: Formulario general creado para la entrada de datos a los puertos. Define los mecanismos generales para la entrada de datos. Otros formularios que heredan de este establecen la entrada a los tipos de puertos específicos.		
Atributo	**Tipo**	**Visibilidad**
flujoCaña	double	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Validar()	bool	protected

Descripción: Se especifican las condiciones que se deben cumplir para considerar válida la entrada de datos.		
TextboxValidos()	bool	public
Descripción: Chequea que todos los campos de entrada de texto en el formulario tengan valores válidos.		
SetGetValores	NegPuerto	public
Descripción: Esta propiedad obtiene los datos que están almacenados en el puerto que contiene al formulario mostrándolos cuando el formulario es presentado, e introduce los datos ofrecidos por el usuario hacia el puerto.		
FlujoCaña	doublez	public
Descripción: Obtiene o establece el valor del atributo flujoCaña.		

Tabla 13: Descripción de la clase frmPropiedades.

Nombre: frmPropiedadesModulo	**Tipo:** formulario	
Descripción: Formulario general creado para la entrada de datos a los módulos. Define los mecanismos generales para la entrada de datos. Otros formularios que heredan de este establecen la entrada a los tipos de módulos específicos.		
Atributo	**Tipo**	**Visibilidad**
flujoCaña	double	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Validar()	bool	protected
Descripción: Se especifican las condiciones que se deben cumplir para considerar válida la entrada de datos.		
TextboxValidos()	bool	public
Descripción: Chequea que todos los campos de entrada de texto en el formulario tengan valores válidos.		
SetGetValores	NegModulo	public
Descripción: Esta propiedad obtiene los datos que están almacenados en el módulo que contiene al formulario mostrándolos cuando el formulario es presentado, e introduce los datos ofrecidos por el usuario hacia el módulo.		
FlujoCaña	doublez	public
Descripción: Obtiene o establece el valor del atributo flujoCaña.		

Tabla 14: Descripción de la clase frmPropiedadesModulo.

Este caso de uso tiene dos escenarios principales: entrar datos a los módulos y entrar datos a los puertos, los mecanismos para darle solución a ambos son bien parecidos, solo se diferencian las clases de los objetos que participan. La figura 7 muestra el diagrama de secuencia donde colaboran las clases cuando se le están entrando datos a un módulo. Para la entrada de datos a un puerto la secuencia es la misma solo que las clases que participan son otras.

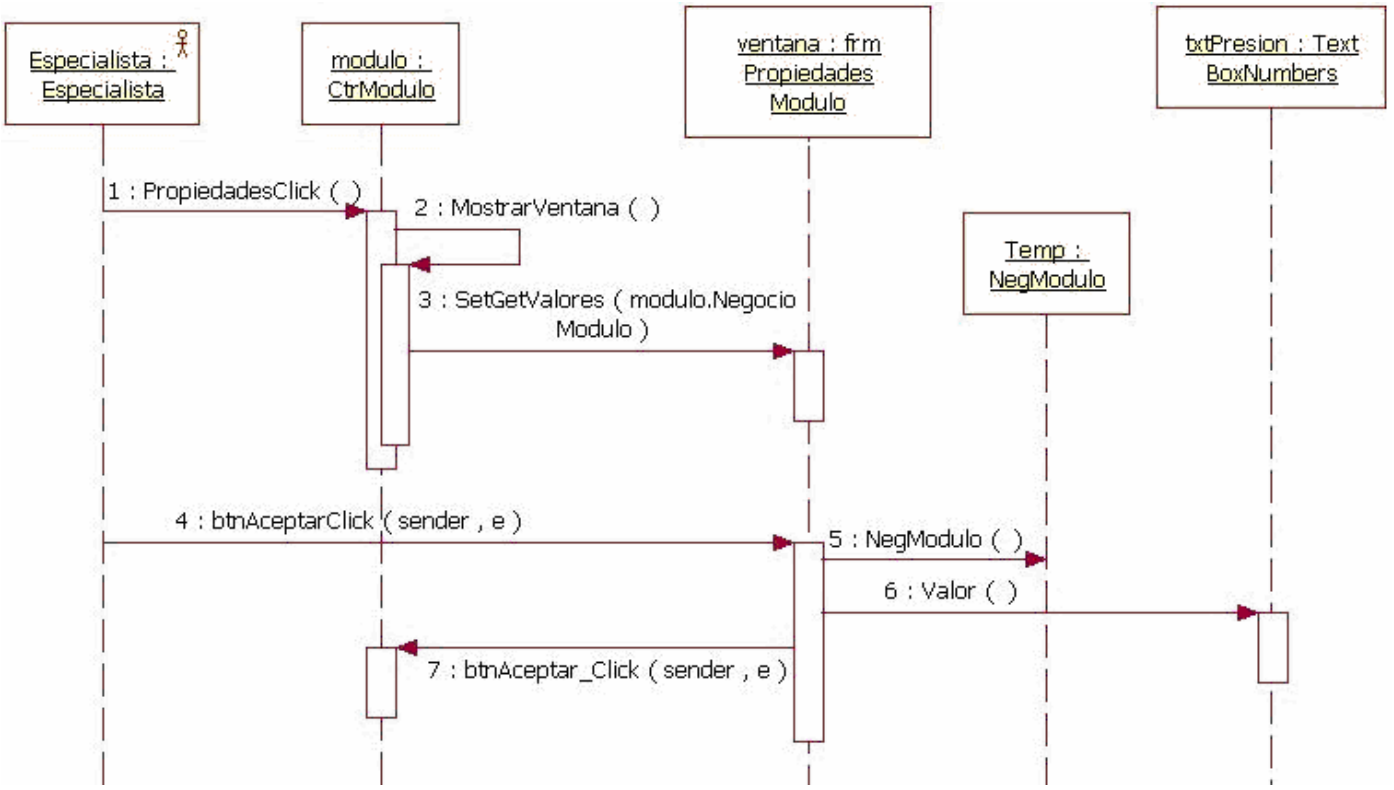

Figura 7: Realización caso de uso Insertar Información. Diagrama de secuencia.

El especialista accede a esta opción a través del menú contextual asociado al módulo dándole click en Propiedades. El módulo recibe esta acción y muestra la ventana pasándole su negocio módulo para que esta muestre los datos que estaban almacenados en él. El usuario modifica los campos de textos de la ventana y presiona el botón Aceptar. El formulario crea un nuevo negocio módulo con los nuevos valores introducidos por el usuario y se lo pasa al módulo. Para el escenario entrar datos a un puerto la secuencia es la misma pero los objetos participantes son de los tipos CtrPuerto, frmPropiedades y NegPuerto, en lugar de CtrModulo, frmPropiedadesModulo y NegModulo respectivamente.

Realización caso de uso Convertir Unidades.

Las propiedades físicas y químicas de los flujos, así como los parámetros adicionales de los equipos tienen asociadas magnitudes físicas que pueden ser expresadas en diferentes unidades de medida. Una funcionalidad básica que debe tener el simulador es permitir que el usuario pueda introducir los datos en cualquier sistema de unidades y usando la unidad que quiera, independientemente de que los cálculos se hagan con determinadas unidades. Para solucionar este problema se definió un conjunto de clases que

colaboran para convertir de una unidad de medida a otra así como permitir que se puedan introducir las unidades de medida en la forma que el usuario desee. En primer lugar esta la clase Conversion que tiene el método público Convertir, al cual se le pasan como parámetros el valor numérico y las unidades desde la que se quiere convertir y a la que se quiere convertir, también se definieron dos enumeradores (enum) eMagnitud y eUM, el primero para definir las magnitudes y el segundo para definir todas las unidades de medida, la relación entre las magnitudes y sus unidades de medida se establece por su índice dentro del enumerador. Conversion contiene además un conjunto de métodos privados que llevan de una unidad de medida a otra, los cuales son invocados dentro de Convertir. Ya más cerca de la interfaz de usuario esta la clase Correspondencia, que establece la correspondencia entre las unidades de medida dentro del enumerador y las cadenas de texto que se muestran en la interfaz gráfica, esto fue necesario hacerlo por el hecho de que muchas unidades de medida tienen asociado caracteres que no pueden ser usados para denominar las variables dentro del lenguaje de programación utilizado, como pueden ser los superíndices o símbolos especiales como en °C o °F. Para que se puedan manipular estos elementos de forma visual se definió un control de usuario especial, el ComboBoxUM, este componente hereda de la clase System.Windows.Forms.ComboBox tomando la apariencia de este. Lo nuevo en este control son las propiedades Magnitud y UM, así como la referencia a un campo numérico asociado. En tiempo de diseño visual, se le especifica que magnitud va a manejar y que campo numérico va a controlar, de esta manera, cuando se cambia la unidad de medida los valores numéricos son convertidos a la nueva unidad.

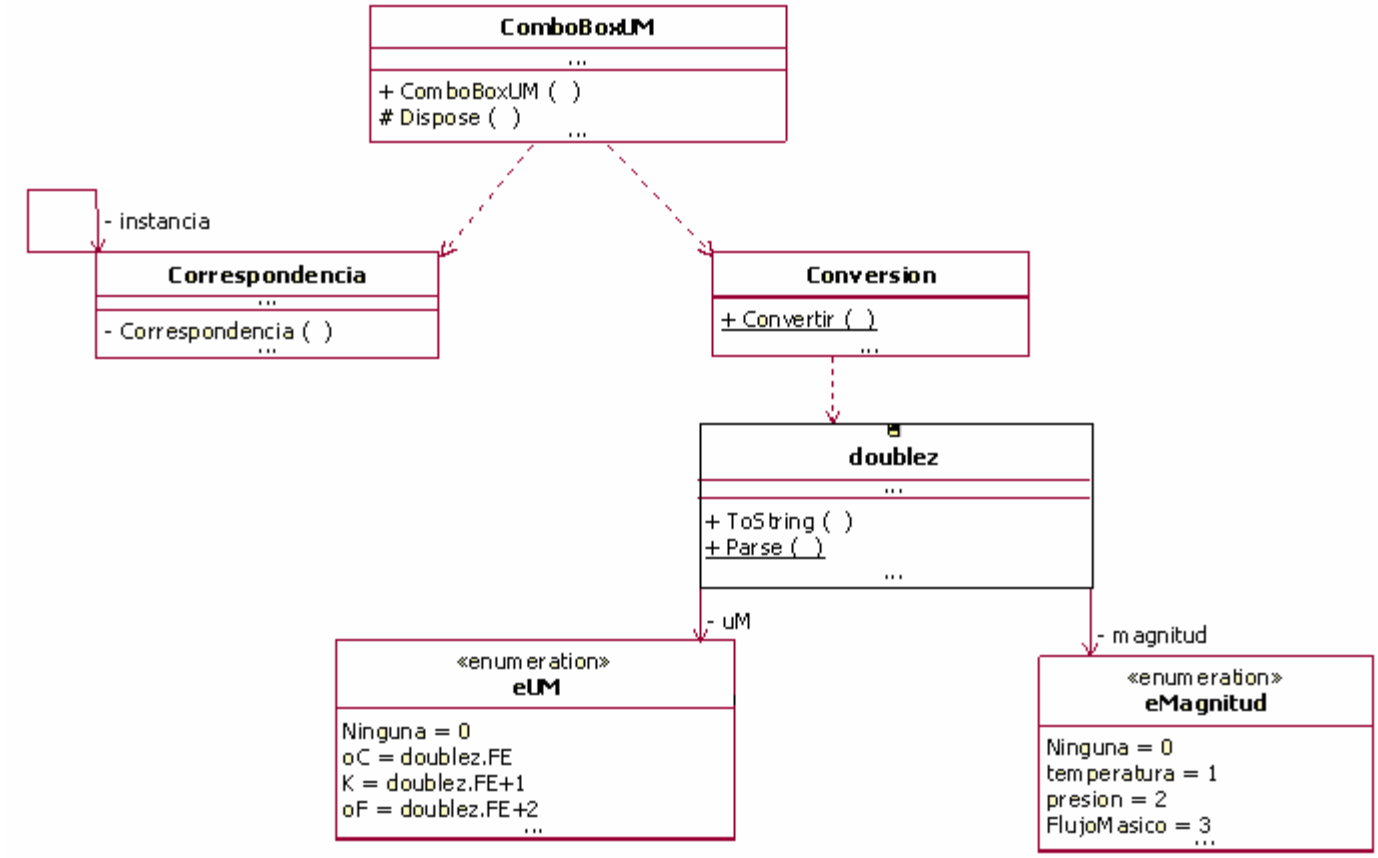

Figura 8: Realización caso de uso Convertir Unidades. Clases del diseño.

Nombre : Correspondencia	**Tipo:** clase	
Descripción: Se encarga de hacer las conversiones correspondientes entre las cadenas de texto y los enumerados que representan las unidades de medidas y las magnitudes.		
Atributo	**Tipo**	**Visibilidad**
Adim()	String	private
UM2String	Hashtable	private
String2UM	Hashtable	private
Métodos y propiedades		
Nombre:	**Tipo de retorno:**	**Visibilidad:**
get_Instancia()	Correspondencia	public

Descripción : Se encarga de que exista un única instancia de la clase Correspondiente		
CorrespondenciaDirecta()	void	private
Descripción: Llena la tabla Hash con la correspondencia entre los enumerados y su representación como una cadena.		
CorrespondenciaInversa()	void	private
Descripción: Llena la tabla Hash con la correspondencia entre las cadenas y su enumerados correspondientes.		
GetUM()	eUM	public
Descripción: Dada una cadena te devuelve la correspondiente unidad de medida.		
GetString()	String	public
Descripción: Dada una unidad de medida te devuelve la cadena correspondiente a dicha unidad de medida.		
GetMag2UM()	String	public
Descripción: Dada una magnitud te devuelve la cadena correspondiente a dicha magnitud.		

Tabla 15: Descripción de la clase Correspondencia

Nombre: doublez	**Tipo:** estructura	
Descripción: Representa un tipo numérico de doble precisión con otras facilidades de conversión de unidades de medidas.		
Atributo	**Tipo**	**Visibilidad**
magnitud	eMagnitud	private
valor	double	private
uM	eUM	private
umUtilizar	eUM	private
uM_Amostrar	eUM	private
Métodos y propiedades		
Nombre :	**Tipo de retorno:**	**Visibilidad:**
set_Magnitud()	void	Public
Descripción: Cambia los valores de la magnitud física asociada.		
set_UM()	Void	public
Descripción: Cambia el valor de la unidad de medida actual convirtiendo el valor, de la unidad que tiene a la unidad de medida pasada.		
Parse	Double	public
Descripción: Dada una cadena de texto la recorre buscando si constituye un número lo devuelve con ese formato		

Tabla 16: Descripción de la estructura Doublez

Nombre: ComboBoxUM		**Tipo:** Control de usuario
Descripción: Componente que tiene el comportamiento de una de una caja de texto en forma de lista, y contiene dentro de ella las unidades en las que esta dada la magnitud que se le seleccione representará.		
Atributo	**Tipo**	**Visibilidad**
magnitudes	eMagnitud	public
unidad	eMagnitud	public
valor_viejo	eUM	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
set_Magnitud	void	public
Descripción: Cambia la magnitud física que representa el control, establece y muestra los unidades de medidas asociadas a esa magnitud.		
get_Magnitud	eMagnitud	public
Descripción: obtiene la magnitud que esta representando el control.		
set_UM	void	public
Descripción: establece la unidad de medida a través a través de la correspondencia entre la cadena de texto y el valor del enumerado correspondiente para poderlo mostrar en el control.		
get_UM	eUM	public
Descripción: obtiene de la cadena de texto selecciona en el control y lo devuelve convertido como el valor de enumeración que le corresponde.		
Redondeo	double	private
Descripción: Redondea un número a las cifras significativas que se desea.		

Tabla 17: Descripción de la clase ComboBoxUM.

Nombre: eUM	**Tipo:** Enumerado
Descripción: Define las unidades de medidas que se manipulan en la arquitectura	

Tabla 18:Descripción del enumerado eUM.

Nombre: eMagnitud	**Tipo:** Enumerado
Descripción: las magnitudes que se manipulan en la arquitectura.	

Tabla 19: Descripción del enumerado eMagnitud.

Nombre: Conversion	**Tipo:** clase	
Descripción: contiene las funcionalidades para la conversión de unidades de medida.		
Atributo	**Tipo**	**Visibilidad**
-	-	-
Métodos y propiedades		
Nombre:	**Tipo de retorno:**	**Visibilidad:**
Convertir()	double	public
Descripción: Realiza la conversión de un valor de una unidad de medida a otra.		

Tabla 20: Descripción de la clase Conversion.

La figura 9 muestra el diagrama de secuencia que soporta el escenario convertir unidad de medida, cuando el usuario decide cambiar la unidad de medida en una entrada de datos.

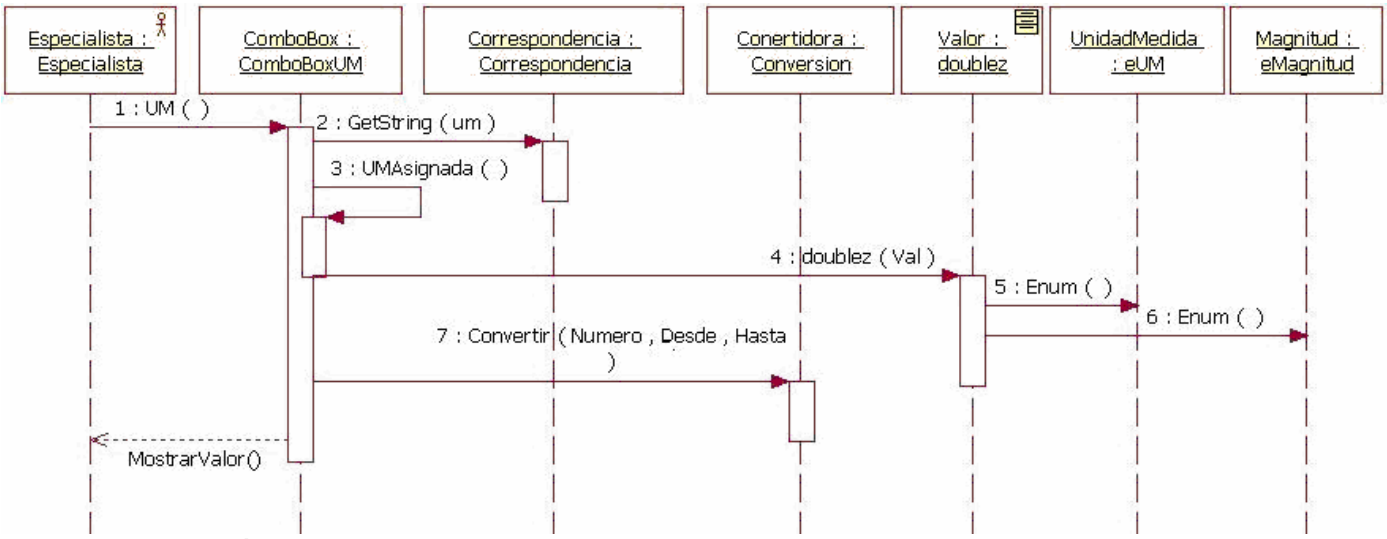

Figura 9: Realización caso de uso Convertir Unidades. Diagrama de secuencia.

El usuario solicita a la caja de texto desplegable de una entrada determinada que cambie su unidad de medida, esta solicita la cadena que debe mostrar a la clase Correspondencia, después llama al método que convierte el valor dado a través del método “Convertir” que provee la clase Conversion entre las unidades suministradas y el valor que es de tipo doublez, el resultado es mostrado al usuario.

Realización caso de uso Salvar Información.

Para salvar o cargar la información del DFI se debe tener en cuenta que se desea almacenar en el disco toda la información visual del diagrama para que este pueda ser reconstruido sin pérdida de modelado,

además todos los datos suministrados por el usuario y los resultados obtenidos en las simulaciones, los análisis económicos hechos al diagrama así como la información referente al autor, la fecha, la descripción, etc. Para lograr esto se utiliza un recurso que proporciona la plataforma de desarrollo .NET que es la serialización, que no es más que la representación en XML de objetos. Con la precondiciones planteadas el diagrama de clases del caso de uso Salvar/Cargar esta compuesto por 4 clases fundamentales, NegSerializador, NegSerializable, NegPanelSerialiable, NegModuloSerializable, NegLineaSerializable, las que son usadas por las clases pnlDiagrama, CtrModulo, CtrEnlace, de manera que cuando el usuario ejecuta la funcionalidad de salvar se recorren los módulos y líneas en busca los objetos generados por ellos para ser serializados. Para su funcionamiento la clases NegSerializador contiene los métodos Salvar y Cargar, los cuales reciben el camino donde será salvado o cargado el fichero, dicha clase es la que utiliza la serialización de objetos para almacenar en el fichero, el objeto que se serializa es creado de la clase NegLineaSerializable la cual almacena dentro de si todos aquellos objetos que se quieran guardar que en este caso son NegPanelSerialiable, NegModuloSerializable y NegLineaSerializable. Los objetos de dichas clases son generados por las clases CtrModulo, pnlDiagrama y ctrEnlace respectivamente y ellas depositan en estos objetos toda la información necesaria para que puedan ser recuperados cuando se cargan.

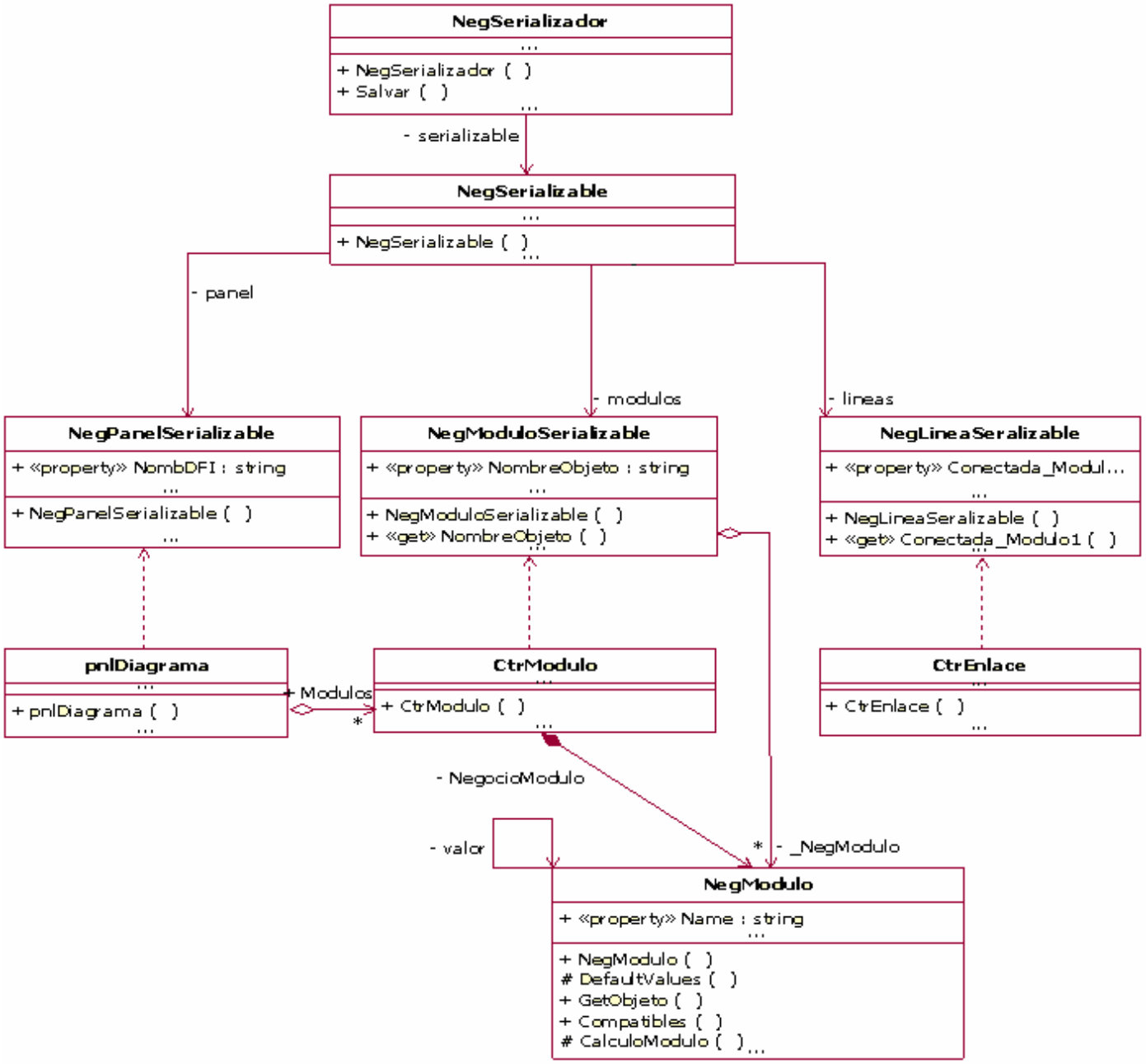

Figura 10: Realización caso de uso Salvar Información. Clases del diseño.

Nombre: NegSerializador	**Tipo:** clase
Descripción: contiene el objeto que se va a salvar y provee las funcionalidades para guardar o recuperar un fichero del disco a través de la serialización.	

Atributo	**Tipo**	**Visibilidad**
serializable	NegSerializable	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
Salvar()	bool	public
Descripción: Almacena en la dirección proporcionada lo que contenga en su campo Serializable		
Cargar()	bool	public
Descripción: Carga desde la dirección especificada el fichero en su campo Serializable.		

Tabla 21: Descripción de la clase NegSerializador.

Nombre: NegSerializable		**Tipo:** clase
Descripción: contiene dentro de si todos los elementos que vayan a ser guardados en el fichero.		
Atributo	**Tipo**	**Visibilidad**
módulos	NegModuloSerializable	private
líneas	NegLineaSeralizable	private
panel	NegPanelSerializable	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
-	-	-
Descripción: -		

Tabla 21: Descripción de la clase NegSerializable.

Nombre: NegPanelSerializable		**Tipo:** class
Descripción: Almacena la información necesaria para recuperar la información de un diagrama		
Atributo	**Tipo**	**Visibilidad**
nombDFI	string	private
nombCentral	string	private
descripcion	string	private
visible	bool	private
autor	string	private
left	int	private
top	int	private
r	byte	private
a	byte	private
b	byte	private

g	byte	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
-	-	-
Descripción: -		

Tabla 22: Descripción de la clase NegPanelSerializable.

Nombre: NegModuloSerializable		**Tipo:** class
Descripción: Contiene toda la información necesaria para almacenar un módulo.		
Atributo	**Tipo**	**Visibilidad**
nombreObjeto	string	private
nombreInstancia	string	private
id	string	private
x	int	private
y	int	private
fliped	bool	private
módulo	NegModulo	private
nombres	string	private
nombres_modulos	string	private
nombres_modulosEXT	string	private
nombres_puertosEXT	string	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
-	-	-
Descripción: -		

Tabla 23: Descripción de la clase NegModuloSerializable.

Nombre: NegLineaSeralizable		**Tipo:** clase
Descripción: contiene toda la información necesaria para almacenar una línea.		
Atributo	**Tipo**	**Visibilidad**
enlace1	string	private
enlace2	string	private
modulo1	string	private
modulo2	string	private
nombreObjeto	string	private
nombreInstancia	string	private

x	int	private
y	int	private
width	int	private
height	int	private
conectada_Modulo1	bool	private
conectada_Modulo2	bool	private
ancho	int	private
Métodos y propiedades		
Nombre	**Tipo de retorno**	**Visibilidad**
-	-	-
Descripción: -		

Tabla 24: Descripción de la clase NegLineaSerializable.

Los diagramas de las figuras 11 muestran los dos escenarios fundamentales de este caso de uso: salvar y cargar información.

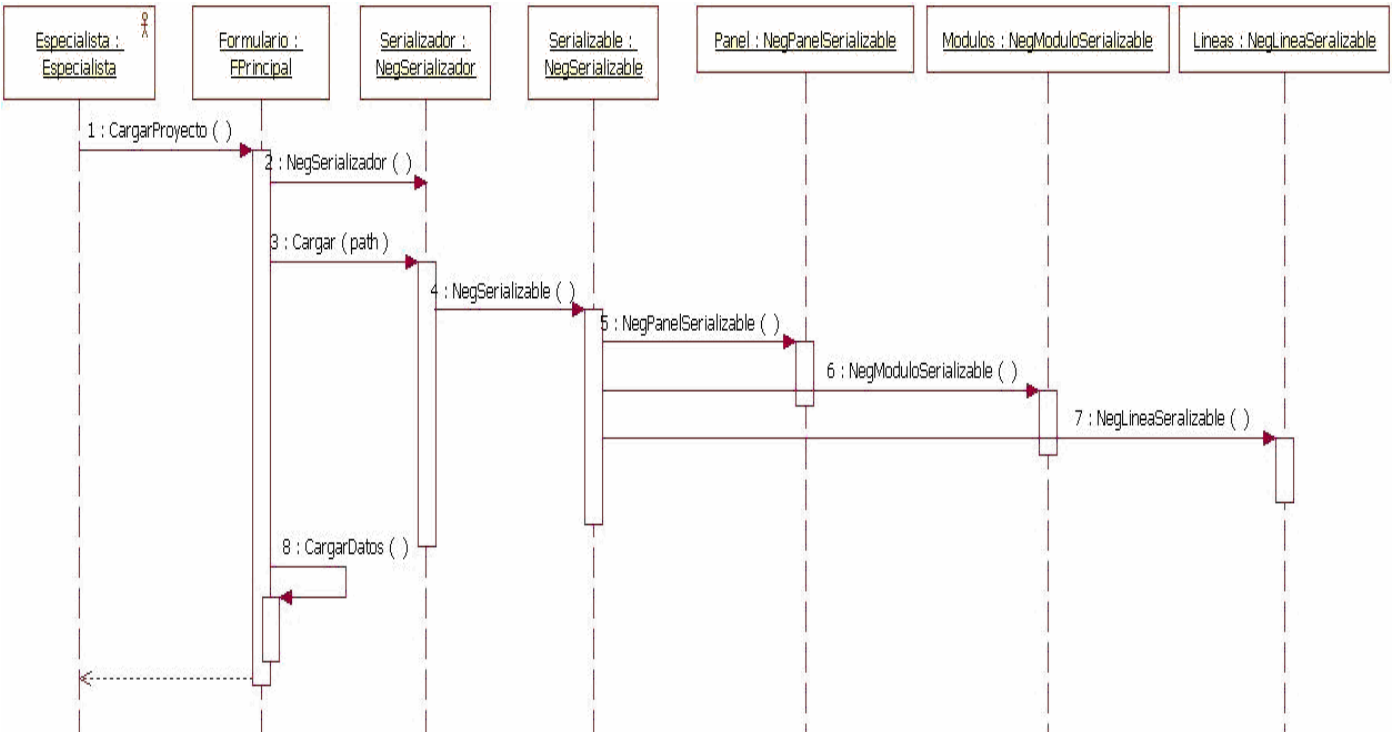

Figura 11: Realización caso de uso Salvar Información. Diagrama de secuencia, escenario cargar información.

El especialista a través de la interfaz del formulario solicita a la aplicación que desea cargar del disco duro un fichero, esta le suministra una interfaz donde puede buscar la ruta del fichero, posteriormente, la interfaz crea una instancia de la clase NegSerializador y solicita el método cargar y le suministra el camino definido por el usuario. El objeto deserealiza el fichero y devuelve las instancias de los objetos con las cuales la interfaz carga en un nuevo diagrama los componentes leídos del disco, y se los muestra al usuario.

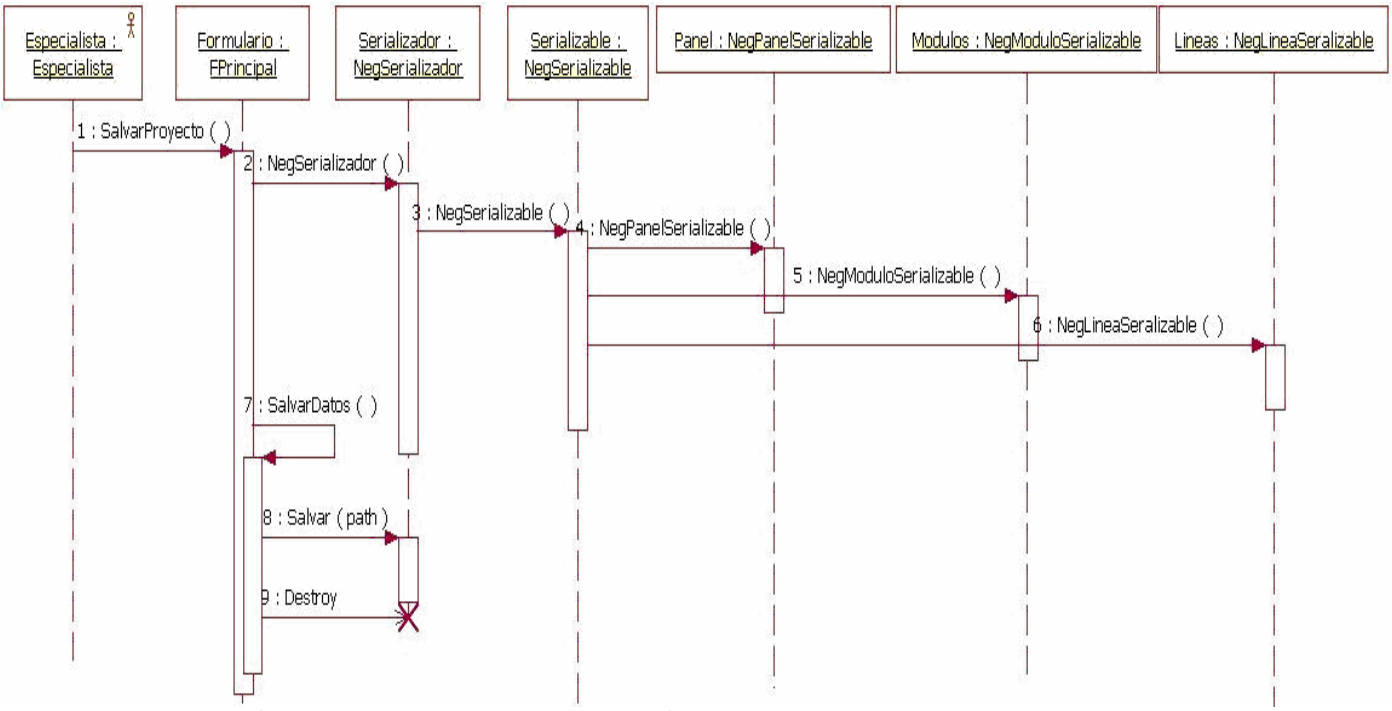

Figura 12: Realización caso de uso Salvar Información. Diagrama de secuencia, escenario salvar información.

El especialista a través de la interfaz del formulario solicita a la aplicación que desea salvar un diagrama determinado para el disco duro en un fichero, esta le suministra una interfaz donde puede buscar la ruta del fichero que desea salvar, posteriormente, la interfaz crea una instancia de la clase NegSerializador y solicita el método salvar y le suministra el camino definido por el usuario y el diagrama que desea guardar. El objeto convierte la información de los elementos del diagrama en un fichero serializable y después serializa en la ruta especificada, la aplicación notifica al usuario a través de una barra de progreso.

Realización caso de uso Reportar Resultados.

La gran cantidad de datos que devuelve el sistema luego de una simulación debe ser presentada al usuario de forma organizada y coherente. Toda esta información va estar almacenada en los módulos y solo hay que tomarla y mostrarla de alguna manera. Para dar solución a este caso de uso se aprovecharon las facilidades del Crystal Reports (CR), el cual permite hacer reportes e informes con facilidad y además da posibilidades de exportar los informes en distintos formatos. Pero existe un pequeño detalle, el CR es para reportar desde tablas de bases de datos, la tarea fundamental se centró entonces en definir un mecanismo que recopilara todos los datos y los ubicara en tablas para que estos fueran accedidos desde el CR, así surgen las clases Modulos, Parámetros, Corrientes y DatosReportes, las cuales colaboran para tomar los datos que resultan de la simulación y ubicarlos en DataSets. En el caso de los reportes más complejos, como el reporte de indicadores, se definieron las clases AsistenteReportes y GestorIndicadores, la primera es un formulario definido en forma de asistente que da la posibilidad al usuario de escoger que indicadores desea reportar. La segunda es donde se calculan los indicadores, aquí se definen tres arreglos para almacenar los indicadores separados por tipo, y los métodos para calcularlos son todos del tipo void, y sin parámetros, de manera que se puedan almacenar en un delegado (delegate) para ser calculados todos juntos al final del reporte, de esta manera el usuario puede navegar como quiera dentro del asistente de selección de los indicadores y solo se calcularan estos en el momento que él decida reportar. También está la clase Acceso_ReporteIndicadores que inserta los indicadores obtenidos en un DataSet para que puedan reportarse con el CR.

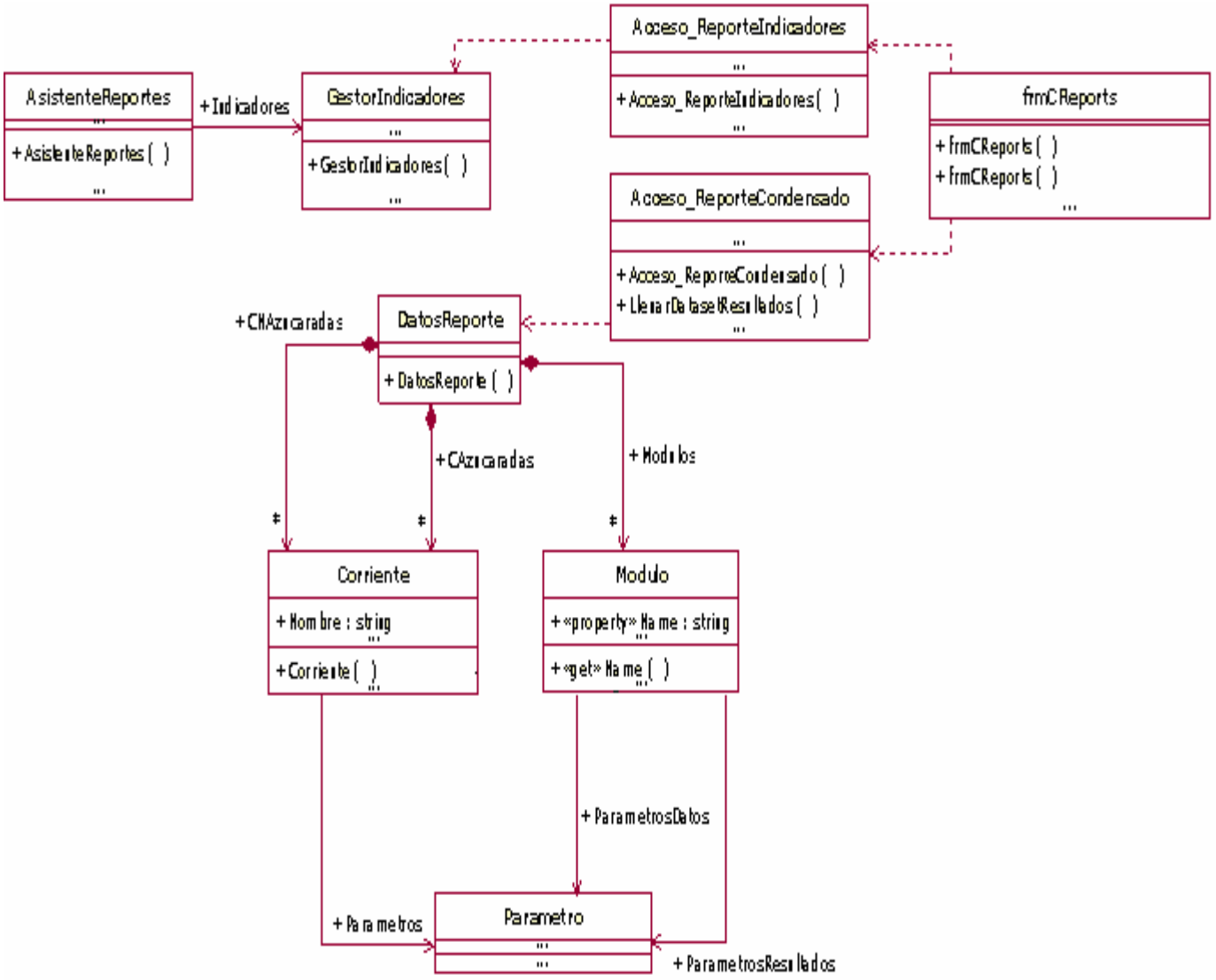

Figura 13: Realización caso de uso Reportar Resultados. Clases de diseño.

Nombre: DatosReporte	**Tipo:** clase	
Descripción: Clase creada para extraer los datos de un diagrama y darle un formato más comprensible para insertar en tablas.		
Atributo	**Tipo**	**Visibilidad**
CAzucaradas	Corriente[]	Public
CNAzucaradas	Corriente[]	Public
Modulos	Modulo[]	Public
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
DatosReporte()		public

Descripción: Al constructor de la clase se la pasa un pnlDiagrama y este llena los arreglos de corrientes azucaradas, no azucaradas y módulos.

Tabla 25: Descripción de la clase DatosReporte.

Nombre: Modulo	**Tipo:** clase	
Descripción: Creada para almacenar los datos de un negocio módulo de forma simple.		
Atributo	**Tipo**	**Visibilidad**
nombre	string	private
ParametrosDatos	Parametro[]	public
ParametrosResultados	Parametro[]	public
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Nombre	string	public
Descripción: Obtiene o establece el valor del atributo nombre.		

Tabla 26: Descripción de la clase Modulo.

Nombre: Corriente	**Tipo:** clase	
Descripción: Creada para almacenar los datos de una corriente de forma simple.		
Atributo	**Tipo**	**Visibilidad**
nombre	string	private
entrada	bool	private
Parametros	Parametro[]	public
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Nombre	string	public
Descripción: Obtiene o establece el valor del atributo nombre.		
Entrada	bool	public
Descripción: Obtiene o establece el valor del atributo entrada, si esta en verdadero indica que la corriente es de entrada.		

Tabla 27: Descripción de la clase Corriente.

Nombre: Parametro	**Tipo:** clase	
Descripción: Representa un parámetro determinado, ya sea de corrientes o módulos.		
Atributo	**Tipo**	**Visibilidad**
valor	double	private
nombre	string	private
um	string	private
rango	string	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
Valor	double	public
Descripción: Obtiene o establece el valor del atributo valor.		
Nombre	string	private

Descripción: Obtiene o establece el valor del atributo nombre.		
UM	string	private
Descripción: Obtiene o establece el valor del atributo um.		
Rango	string	private
Descripción: Obtiene o establece el valor del atributo rango.		

Tabla 28: Descripción de la clase Parametro.

Nombre: GestorIndicadores	**Tipo:** clase	
Descripción: Clase que gestiona al cálculo de los indicadores.		
Atributo	**Tipo**	**Visibilidad**
ParametroGlobales	ArrayList	public
ParametroArea	ArrayList	public
ParametroEquipos	ArrayList	public
Diagrama	pnlDiagrama	public
AguaTecnologica	NegPuertoAgua	public
Areastransferencia	NegInfo	public
Bagazo	NegPuertoAzucarado	public
Bdisponible	NegPuertoAzucarado	public
Bsobrante	NegPuertoAzucarado	public
Calentadores	NegCool[]	public
Canna	NegPuertoAzucarado	public
CondPuros	NegPuertoAgua[]	public
Consumome	NegPuertoAgua	public
Consvapor	NegPuertoAgua[]	public
ConsvaporEvap	NegPuertoAgua[]	public
ConsvaporTachos	NegPuertoAgua[]	public
DemElectrica	double	public
EfecTermodinamica	NegInfo	public
EficBruta	NegInfo	public
Eners	NegEner[]	public
Evaporaciones	NegPuertoAgua[]	public
Evaporadores	NegEvap[]	public
Extraccion	NegPuertoAgua	public
FacPerdCalorTacho	NegInfo	public
JAlcalizado	NegPuertoAzucarado	public
JAlcalizadoAF	NegPuertoAzucarado	public
JClaro	NegPuertoAzucarado	public
Meladura	NegPuertoAzucarado	public
PReposicion	double	public
Pans	NegPan[]	public
ProdElectrica	NegInfo[]	public
TempAguaACalderas	NegInfo	public
VaporAltaRed	NegPuertoAgua	public
VaporExp	NegPuertoAgua[]	public
VaporRedEsc	NegPuertoAgua	public
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**

Calculados()	bool	private
Descripción: Determina si un parámetro especificado ya fue calculado.		
BagazoSobrante()	void	public
Descripción: Calcula el % de bagazo sobrante.		
BagazoDispoCana()	void	public
Descripción: Calcula el % de bagazo disponible con relación a la caña molida.		
VaporConsumido()	void	public
Descripción: Calcula relación entre el vapor consumido y la caña molida		
VaporExpulsado()	void	public
Descripción: Calcula el flujo de vapor expulsado a la atmósfera.		
VaporDirecto()	void	public
Descripción: Calcula la relación entre el vapor directo y el vapor consumido.		
VaporEscp()	void	public
Descripción: Calcula la relación entre el vapor de escape y el de extracción.		
DemandaElect()	void	public
Descripción: Calcula la relación entre la demanda eléctrica de la fábrica y la caña molida.		
ProducEspec()	void	public
Descripción: Calcula la relación entre la producción eléctrica de la fábrica y la caña molida.		
AguaRepCalderas()	void	public
Descripción: Calcula el flujo del agua de reposición en las calderas.		
ConsVaporEvap()	void	public
Descripción: Calcula el consumo de vapor en los evaporadores.		
ConsVaporTachos()	void	public
Descripción: Calcula el consumo de vapor en la estación de tachos.		
FactorPerdCalor()	void	public
Descripción: Calcula el factor de perdidas de vapor en los tachos.		
AguaTecn()	void	public
Descripción: Calcula la relación entre el agua tecnológica y la caña molida.		
BrixJugoClaro()	void	public
Descripción: Calcula el brix del jugo claro.		
BrixMeladura()	void	public
Descripción: Calcula el brix de la meladura.		
TempInicialJugo()	void	public
Descripción: Calcula la temperatura inicial del jugo.		
TempJugoCalent()	void	public
Descripción: Determina la temperatura del jugo a la salida del calentador.		
RelaCoefGlobalCalor()	void	public
Descripción: Halla la relación entre el coeficiente de transferencia de calor calculado y el de Hugot.		
VAbsCoefGlobalCalor()	void	public
Descripción: Determina el valor absoluto del coeficiente de transferencia de calor calculado.		
VelocidadJugo()	void	public
Descripción: Calcula la velocidad del jugo en los calentadores.		
EfectividadTermo()	void	public
Descripción: Calcula la efectividad termodinámica en los calentadores.		
DifTempJugo()	void	public
Descripción: Calcula la diferencia de temperatura entre el jugo que entra y sale en los calentadores.		
TasaEvaporación()	void	public
Descripción: Determina la tasa de evaporación de un evaporador.		
Economía()	void	public

Descripción: Determina la economía de un evaporador.		
TasaEvaporaciónME()	void	public
Descripción: Determina la tasa de evaporación de un múltiple efecto.		
EconomiaME()	void	public
Descripción: Determina la economía de un múltiple efecto.		
HumedadBagazo()	void	public
Descripción: Calcula la humedad en el bagazo.		
TenmperaturaCombustion()	void	public
Descripción: Calcula la temperatura de los gases de combustión.		
TempAguaCalderas()	void	public
Descripción: Determina la temperatura del agua en las calderas.		
EficBruta()	void	public
Descripción: Calcula la eficiencia bruta de los generadores de vapor.		
GenerBruto()	void	public
Descripción: Calcula el índice de generación bruto de los generadores de vapor.		
GenerNeto()	void	public
Descripción: Calcula el índice de generación neto de los generadores de vapor.		

Tabla 29: Descripción de la clase GestorIndicadores.

Nombre: AsistenteReportes	**Tipo:** formulario	
Descripción: Formulario que sirve de asistente para el cálculo de los indicadores. Hace los cálculos mediante el atributo GestorIndicadores que tiene.		
Atributo	**Tipo**	**Visibilidad**
Indicadores	GestorIndicadores	public
rIniciado	bool	private
modulos	ArrayList	private
CV	bool	private
Canna	bool	private
BD	bool	private
completado	bool	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
OperacionActual()	void	private
Descripción: Calcula el indicador relacionado con la hoja del asistente que se acaba de configurar.		
LLenarArbol()	void	private
Descripción: Establece los indicadores que pueden ser calculados por el asistente.		
FiltrarLista()	void	private
Descripción: Filtra la lista de corrientes de acuerdo con un tipo de corriente especificado.		
FiltrarArbol()	void	private
Descripción: Filtra el árbol de módulos de acuerdo con un tipo de módulo especificado.		
RestablecerLista()	void	private
Descripción: Restablece la lista de corrientes.		
RestablecerArbol()	void	private
Descripción: Restablece el árbol de módulos.		
ClasificarModulos()	void	private
Descripción: Clasifica los modulos agregándolos en diferentes listas según el tipo.		
siguiente_Click()	void	private

Descripción: Especifica las operaciones que se hacen cada vez que se presiona el botón siguiente del asistente.		
finalizar_Click()	void	private
Descripción: Especifica las operaciones que se hacen cuando se presiona el botón finalizar del asistente. Se mandan a calcular todos los indicadores.		

Tabla 30: Descripción de la clase AsistenteReportes.

Nombre: Acceso_ReporteIndicadores	**Tipo:** clase	
Descripción: Clase que a partir de un objeto GestorIndicadores llena un dataset para poder reportar con el Crystal Reports.		
Atributo	**Tipo**	**Visibilidad**
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
LlenarDataset()	DataIndicadores	public
Descripción: Llena un dataset del tipo DataIndicadores a partir de un objeto GestorIndicadores.		

Tabla 31: Descripción de la clase Acceso_ReporteIndicadores.

Nombre: Acceso_ReporteCondensado	**Tipo:** clase	
Descripción: Clase que a partir de un objeto DatosReporte llena un dataset para poder reportar con el Crystal Reports.		
Atributo	**Tipo**	**Visibilidad**
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
LlenarDatasetResultados()	DataCondensado	public
Descripción: Llena un dataset con los datos calculados durante la simulación.		
LlenarDatasetDatos()	DataCondensado	public
Descripción: Llena un dataset con los datos suministrados por el usuario.		

Tabla 32: Descripción de la clase Acceso_ReporteCondensado.

Nombre: frmCReports	**Tipo:** formulario	
Descripción: Formulario que muestra los reportes, contiene un crystalReportViewer que permite visualizar los reportes.		
Atributo	**Tipo**	**Visibilidad**
crystalReportViewer	CrystalReportViewer	private
Métodos y propiedades		
Nombre:	**Tipo de retorno**	**Visibilidad**
GenerarReportCondensadoResultados()	void	public
Descripción: Genera un reporte con la información generada por simulador.		
GenerarReportCondensadoDatos()	void	public
Descripción: Genera un reporte con la información suministrada por el usuario.		
GenerarReportIndicadores()	void	public
Descripción: Genera un reporte con la información de los indicadores calculados.		

Tabla 33: Descripción de la clase fmCReports.

Este conjunto de clases colaboran para dar cumplimiento a los requisitos funcionales y no funcionales relacionados con el reporte de los datos manejados por el simulador. La figura 14 muestra el diagrama de secuencia para los escenarios mostrar reporte condesado de datos y mostrar reporte condensado de resultados.

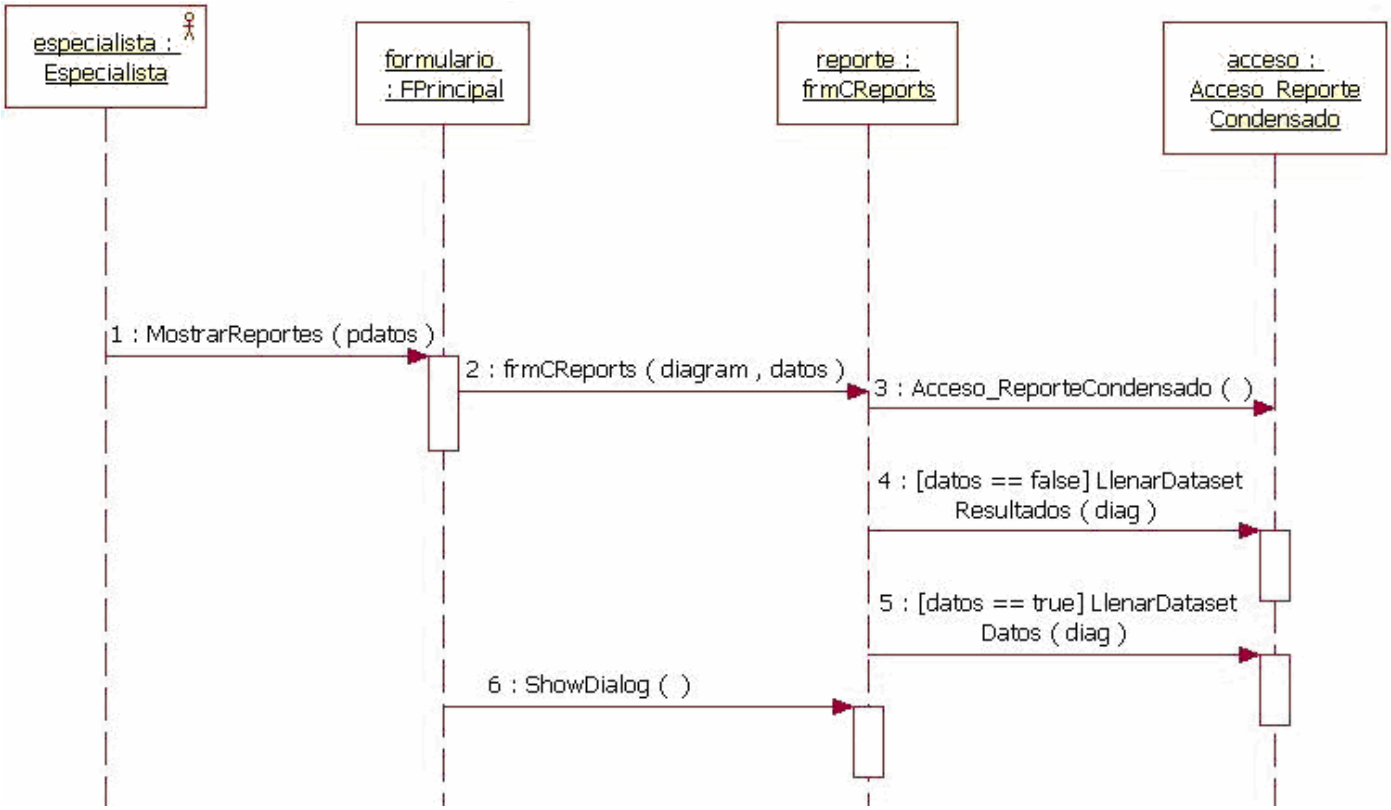

Figura 14: Realización caso de uso Reportar Resultados. Diagrama de secuencia.

El especialista desde el formulario principal puede escoger entre ver el reporte de los datos que el suministró y el reporte de los datos calculados por el simulador. El formulario recibe la orden de mostrar un reporte condensado, crea un formulario de visualización de reportes pasando el DFI que quiere reportar y especificando si es datos o resultados lo que solicitó el usuario. El reporte crea un objeto Acceso_ReporteCondensado() y en dependencia del tipo de reporte llama al método correspondiente. Finalmente el formulario principal indica al reporte que se muestre.

Vista de implementación.

La vista de implementación muestra el software como los elementos físicos que lo componen: ficheros, componentes, librerías, etc. El sistema se estructuró como un grupo de librerías de vínculo dinámico (dll) que son integradas en un ejecutable que constituye el núcleo del simulador. Esta estructura posibilita la flexibilidad del sistema y la reusabilidad. Para la selección de los componentes se siguió el criterio de encapsular las funcionalidades afines y separar la interfaz de usuario del negocio. Los componentes más importantes creados fueron Chesys, Generico e Intermedia. Generico ofrece los elementos principales para garantizar el procesamiento gráfico, Intermedia encapsula las clases e interfaces del negocio y Chesys constituye el núcleo de la aplicación donde se ensamblan y muestra la interfaz del usuario, o sea, contiene el formulario principal donde aparecen los menús, barrar de herramientas y estado, paleta de módulos, editor gráfico, etc. A estos elementos principales se integran otros componentes como los reportes, el tratamiento de ficheros, y cada uno de los modelos matemáticos que posibilita el proceso de la simulación de la fábrica, los cuales fueron modelados de manera independiente, posibilitando que se pueden agregar o quitar de la aplicación general con relativa facilidad. Como se puede observar existe una traza directa entre los paquetes de diseño y los elementos que integran el diagrama de componentes.

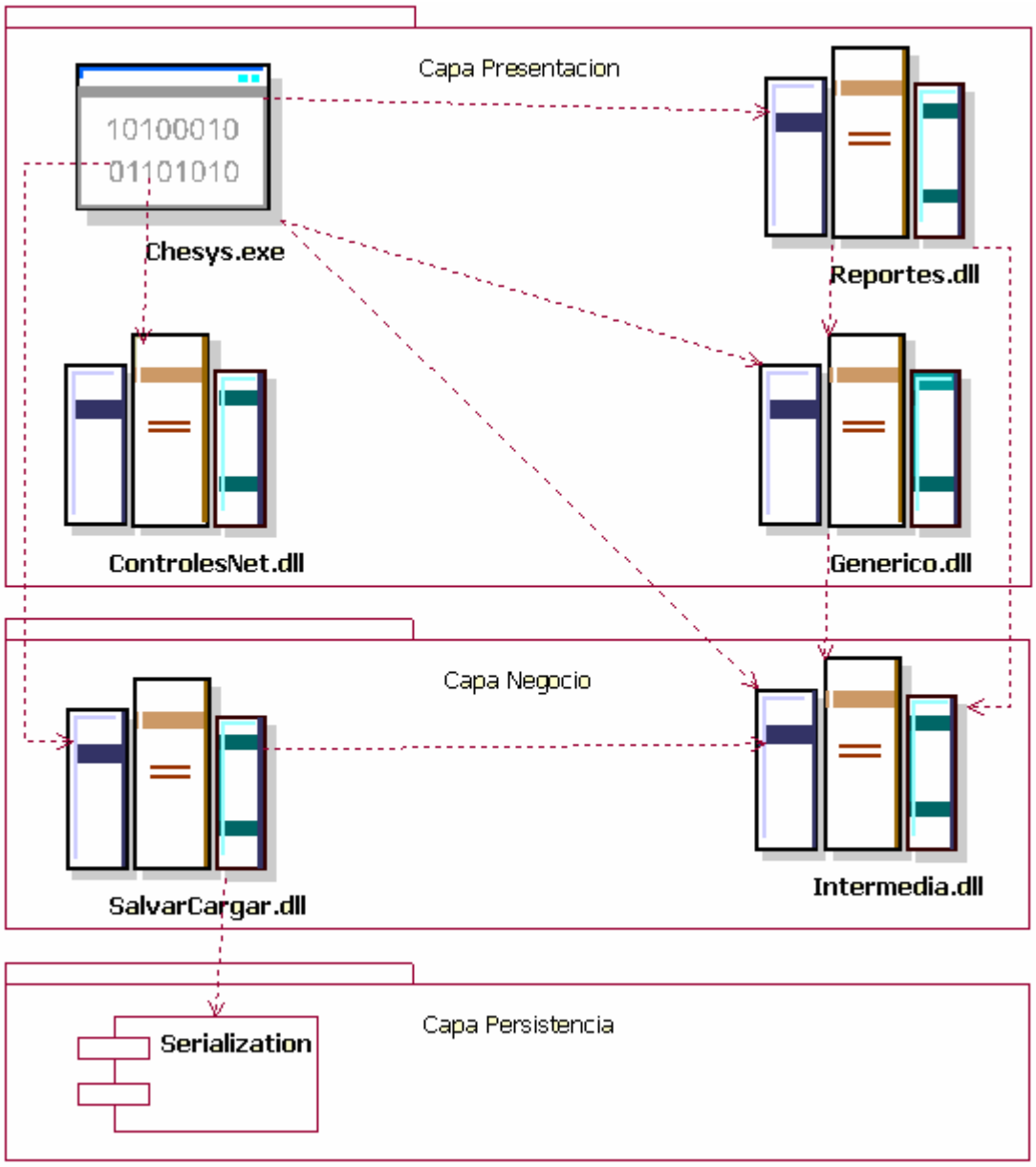

Figura 15: Diagrama de componentes.

Vista de despliegue.

La vista de despliegue permite ver el sistema ya distribuido sobre nodos de procesamiento, servidores, estaciones de trabajo o dispositivos especiales. En esta vista se muestran las unidades de

implementación asignadas a dichos nodos. En el caso particular del sistema que se discute en este trabajo no tiene mucha relevancia mostrar o no esta vista, esto se debe a que el sistema esta concebido como una aplicación de escritorio cuyos componentes corren sobre la misma computadora. En el diagrama de despliegue solo se tendría la computadora como nodo de procesamiento y una impresora como dispositivo, la cual se usaría para la impresión de los reportes o de las imágenes de los diagramas exportados.

Uso de patrones.

En el desarrollo de una arquitectura hay dos factores de éxito importantes, la experiencia que tenga el arquitecto del desarrollo de sistemas anteriores y el dominio que tenga de los patrones y estilos arquitectónicos. Es mucha la experiencia acumulada en los patrones y sería una ingenuidad dejar pasar las ventajas que esto pueda ofrecer, sobre todo cuando el arquitecto se enfrenta por primera vez a la concepción de un sistema. En el desarrollo de este trabajo se tuvieron en cuenta patrones como el patrón Capas, el sistema fue estructurado como un conjunto de componentes que se encuentran en tres capas, tres niveles de abstracción distintos. La capa de interfaz contiene los subsistemas Chesys, Generico y Reportes, en la capa de negocio están SalvarCargar e Intermedia y en la tercera capa, la de persistencia esta el componente System.XML.Serialzation que provee la plataforma .NET. Tal y como especifica el patrón estos paquetes se relacionan con los paquetes de la misma capa o de la inferior, en ninguno de los casos hay relaciones desde la capa de negocio hacia la de presentación. En la figura 16 se muestran las relaciones entre los subsistemas.

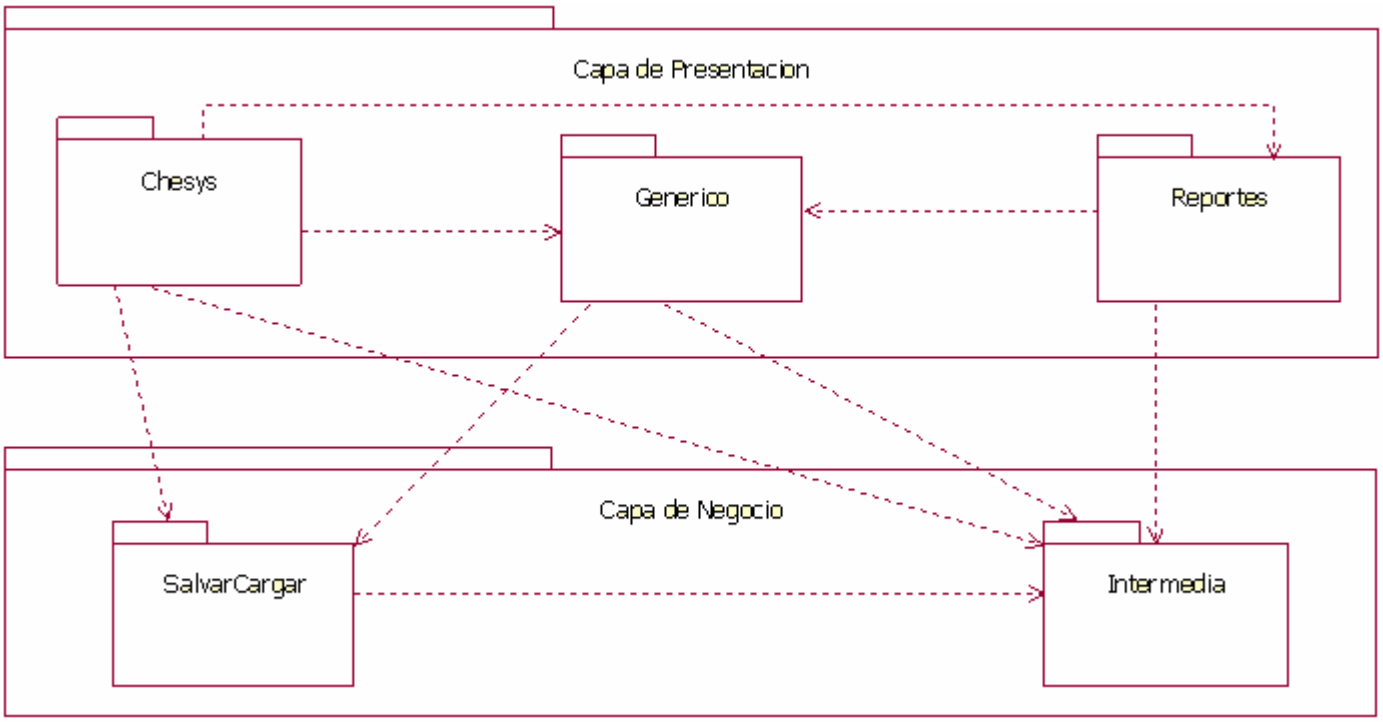

Figura 16: Patrón Capas.

Esta estructura hace totalmente independiente el negocio de la interfaz de usuario, elemento que puede ser aprovechado para la implementación de la interfaz de usuario usando otras tecnologías sin perder el negocio. Uno de los factores que influyó en la selección de este patrón es el hecho de que se pretende desarrollar una versión del simulador sobre plataforma Linux. Ya hace unos años que se viene desarrollando el proyecto Mono, el cual desarrolla una versión del Framework.Net para Linux. En caso extremo de que sea imposible la migración total de la interfaz de usuario, no se perdería ninguna de las facilidades del negocio, pudiéndose construir el producto sobre esta base.

También es posible la sustitución del mecanismo de persistencia sin que haya que hacer cambios significativos en los demás componentes del sistema, el único que se vería afectado seria SalvarCargar y como este tiene bien definidas las interfaces de comunicación con los demás no sería gran problema un cambio en la estructura interna de este.

Visto el sistema desde otra perspectiva se puede observar que está constituido por un conjunto de unidades que se combinan para procesar una serie de datos de entrada y producir una salida. Entonces podemos verlo como un sistema de tuberías y filtros, los filtros son cada uno de los módulos de cálculo

que están encapsulados en paquetes independientes y las tuberías el mecanismo de transferencia de datos establecido para la comunicación entre dichos paquetes.

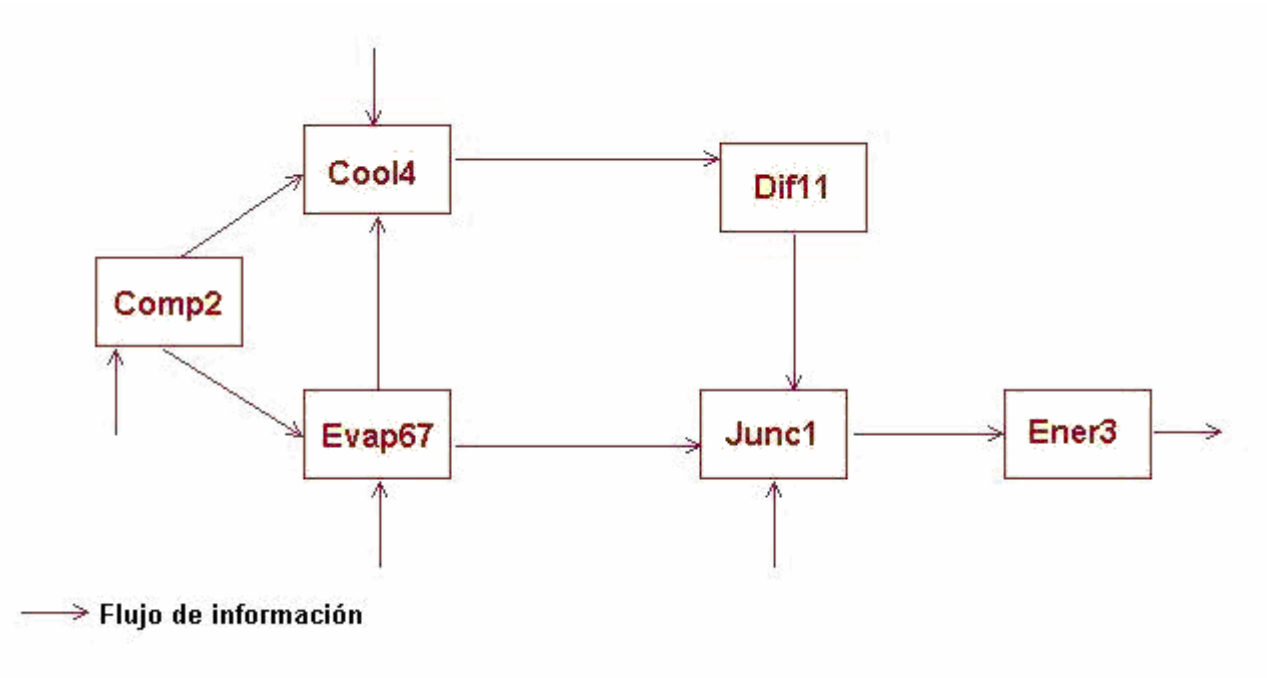

Figura 17: Patrón Tuberías y Filtros.

Cada módulo representa un equipo real de la industria, cuando una corriente pasa a través de un equipo, son transformados algunos de los parámetros de dicha corriente así como parámetros específicos del módulo. Cada DFI que se grafique en el simulador traerá una nueva configuración de los filtros, y los resultados a la salida de todos los filtros dependerá lógicamente de los datos de entrada y de la combinación que se le haya dado a los módulos en el diagrama. La concepción de simulación que tienen los modelos matemáticos fue determinante en la selección de este patrón, ya que su estructura modular-secuencial es perfecta para encapsular en filtros independientes las funcionalidades de cada uno de los equipos de una fábrica.

Patrones de diseño.

En el proceso de obtención de la arquitectura el diseño juega un papel importante y seria imperdonable no mencionar los patrones de diseño cuando se describen los patrones usados en la arquitectura. Al final todos esos patrones usados en el diseño influyen directamente en la calidad del diseño arquitectónico.

Según Craig Larman en [9] un patrón es una descripción de un problema y su solución que recibe un nombre que puede emplearse en otros contextos indicando la manera de utilizarlo en circunstancias diversas. A lo largo de todo el diseño propuesto se usaron patrones para resolver problemas que se fueron presentando. En primer lugar los patrones GRASP (General Responsibility Assignment Software Patterns) los cuales se usan ya hace bastante tiempo no tanto como patrones netamente, sino como principios generales básicos para lograr un buen diseño. Experto, Creador y Controlador están presentes en muchas de las clases que componen la arquitectura, de la misma manera que estuvieron presentes los principios de alta cohesión y bajo acoplamiento en la toma de decisiones importantes relacionadas con la tarea de diseñar el sistema. Por otro lado fueron usados también con grandes resultados los patrones de la pandilla de los cuatro, GOF. Algunos como Fachada, Singleton, Observador y Comando entre otros aparecen con frecuencia en el diseño de algunos de los subsistemas.

El Singleton es utilizado para garantizar que exista solo una instancia de una clase en todo el sistema. En el contexto del simulador de procesos sucede algo así con las unidades de medida definidas para la aplicación, estas son comunes para todo. Las unidades fueron definidas mediante un enumerador, el cual contiene todas las unidades posibles, pero la forma en que están especificadas no es la más apropiada para presentarlas en una interfaz de usuario, por esa razón se definió una clase que convirtiera las unidades de medidas en cadenas de textos que tuvieran un formato adecuado. Esta clase se llamó Correspondencia y para su definición se utilizó el patrón Singleton, garantizando así que solo existiera una instancia de esta clase para convertir el formato de texto de las unidades.

El formulario principal de la aplicación contiene múltiples proyectos abiertos a la vez y tiene acceso a todos ellos, sin embargo ninguno de los diagramas que representan a los proyectos conoce al formulario. En la ventana principal existen otros componentes como son las barras de estado y progreso y algunos botones de las barras de herramientas que tienen que ser notificados de cambios que ocurren en los diagramas. Para resolver esta encrucijada se utilizó el patrón Observador, el formulario principal es un observador de cada diagrama, cuando ocurre algún cambio significativo en uno de ellos el formulario es notificado mediante eventos y este a su vez actualiza a los demás elementos que deben reflejar los cambios.

Una funcionalidad imprescindible en todo editor, ya sea de textos o gráfico es la posibilidad de deshacer y rehacer acciones. En la realización de este importante requisito se utilizó el patrón Comando para definir el mecanismo de deshacer y deshacer operaciones. Las posibles acciones que se pueden hacer sobre un diagrama fueron modeladas como comandos, los cuales son almacenados en una pila, de manera que

pueden ser accedidos en orden contrario a como fueron hechas las acciones por el usuario. Así se podrá ir y venir a través de las operaciones realizadas.

El patrón Fachada fue también utilizado, en especial para disminuir la dependencia entre subsistemas de diseño y garantizar puntos de acceso común a dichos componentes.

Capítulo 3: Análisis de los resultados.

Durante la fase de elaboración de ciclo de desarrollo del software se obtiene la arquitectura inicial del sistema, la cual implementa los casos de uso más significativos y lo requisitos que representan un mayor riesgo para el desarrollo. En este punto se tiene definida la arquitectura base del simulador de la industria química cubana, el cual soporta las funcionalidades de la creación de diagramas de flujo de información, la entrada de datos, la simulación, la posibilidad de reportar los resultados y de guardar la información que provee el usuario que interactúa con el sistema. Queda por delante la creación de la biblioteca de módulos de cálculo que operará el sistema. Durante esta etapa la arquitectura inicial queda sometida a prueba, lo que permitirá corroborar o no si la arquitectura soporta los requisitos funcionales y si cumple con los diferentes atributos de calidad relacionados con esta. El primer elemento es fundamental, una arquitectura que no soporte los requisitos del cliente no es funcional, es uno de los principales puntos de evaluación de la calidad de la arquitectura. El segundo elemento, los atributos de calidad, están relacionados con los requisitos no funcionales y tienen igualmente una gran relevancia. Por otro lado el sistema debe ser flexible a cambios, fácil de extender, reusable y robusto.

Existen diferentes métodos de evaluación de las arquitecturas, entre ellos Software Architecture Analysis Method (SAAM) [7] propuesto por Rick Kazman, Len Bass, Gregory Abowd y Mike Webb quienes proponen tres perspectivas para el análisis de la arquitectura y un método de cinco pasos para el análisis. Por otro lado Architecture Tradeoff Analysis Method (ATAM) el cual esta diseñado para producir como respuestas las metas comerciales tanto del sistema como de la arquitectura, y usar esas metas y la participación de los stakeholders para centrar la atención de los evaluadores en la porción de la arquitectura que es esencial para el cumplimientos de dichas metas [8]. Un concepto importante en la evaluación de la arquitectura son los atributos de calidad, los cuales constituyen los criterios de evaluación. En este trabajo los atributos seleccionados para hacer la evaluación se corresponden con los establecidos en el estándar ISO 9126 para la medida de calidad del software:

Característica	**Descripción**
Funcionabilidad	**Adecuación** Capacidad del producto software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados.

	Exactitud Capacidad del producto software para proporcionar los resultados o efectos correctos o acordados, con el grado necesario de precisión. **Interoperabilidad** Capacidad del producto software para interactuar con uno o más sistemas especificados. **Seguridad de acceso** Capacidad del producto software para proteger información y datos de manera que las personas o sistemas no autorizados no puedan leerlos o modificarlos, al tiempo que no se deniega el acceso a las personas o sistemas autorizados **Cumplimiento funcional** Capacidad del producto software para adherirse a normas, convenciones o regulaciones en leyes y prescripciones similares relacionadas con funcionalidad
Fiabilidad	**Madurez** Capacidad del producto software para evitar fallar como resultado de fallos en el software. **Tolerancia a fallos** Capacidad del software para mantener un nivel especificado de prestaciones en caso de fallos software o de infringir sus interfaces especificados. **Capacidad de recuperación** Capacidad del producto software para reestablecer un nivel de prestaciones especificado y de recuperar los datos directamente afectados en caso de fallo. **Cumplimiento de la fiabilidad** Capacidad del producto software para adherirse a normas, convenciones o regulaciones relacionadas con al fiabilidad.
Usabilidad	**Capacidad para ser entendido** Capacidad del producto software que permite al usuario entender si el software es adecuado y cómo puede ser usado para unas tareas o condiciones de uso particulares. **Capacidad para ser aprendido**

	Capacidad del producto software que permite al usuario aprender sobre su aplicación. **Capacidad para ser operado** Capacidad del producto software que permite al usuario operarlo y controlarlo. **Capacidad de atracción** Capacidad del producto software para ser atractivo al usuario. **Cumplimiento de la usabilidad** Capacidad del producto software para adherirse a normas, convenciones, guías de estilo o regulaciones relacionadas con la usabilidad.
Eficiencia	**Comportamiento temporal** Capacidad del producto software para proporcionar tiempos de respuesta, tiempos de proceso y potencia apropiados, bajo condiciones determinadas. **Utilización de recursos** Capacidad del producto software para usar las cantidades y tipos de recursos adecuados cuando el software lleva a cabo su función bajo condiciones determinadas. **Cumplimiento de la eficiencia** Capacidad del producto software para adherirse a normas o convenciones relacionadas con la eficiencia.
Mantenibilidad	**Capacidad para ser analizado** Es la capacidad del producto software para serle diagnosticadas deficiencias o causas de los fallos en el software, o para identificar las partes que han de ser modificadas. **Capacidad para ser cambiado** Capacidad del producto software que permite que una determinada modificación sea implementada. **Estabilidad** Capacidad del producto software para evitar efectos inesperados debidos a modificaciones del software. **Capacidad para ser probado**

	Capacidad del producto software que permite que el software modificado sea validado. **Cumplimiento de la mantenibilidad** Capacidad del producto software para adherirse a normas o convenciones relacionadas con la mantenibilidad.
Portabilidad	**Adaptabilidad** Capacidad del producto software para ser adaptado a diferentes entornos especificados, sin aplicar acciones o mecanismos distintos de aquellos proporcionados para este propósito por el propio software considerado. **Instalabilidad** Capacidad del producto software para ser instalado en un entorno especificado. **Coexistencia** Capacidad del producto software para coexistir con otro software independiente, en un entorno común, compartiendo recursos comunes. **Capacidad para reemplazar** Capacidad del producto software para ser usado en lugar de otro producto software, para el mismo propósito, en el mismo entorno. **Cumplimiento de la portabilidad** Capacidad del producto software para adherirse a normas o convenciones relacionadas con la portabilidad.

Tabla 34: Atributos de calidad según ISO 9126.

Estos elementos están enfocados al producto final, a las características externas del producto cuando esta en operación pero indiscutiblemente la arquitectura tiene una gran influencia en el cumplimiento de estos atributos. Entonces el análisis en este capítulo estará dirigido a verificar si la arquitectura propuesta cumple o no con estas características.

Funcionalidad.

En primer lugar la arquitectura tiene que ser funcional, garantizar que el sistema cumpla con todos los requisitos del usuario. El desarrollo de la arquitectura a partir de la selección de casos de usos arquitectónicamente significativos posibilita que esta se desarrolle dirigida por esos casos de usos, es

decir que se piense con lo que quiere el usuario en mente. Una vez que se tiene la arquitectura inicial, en las siguientes iteraciones va creciendo esta arquitectura con la implementación de nuevos casos de usos, garantizando que todo el tiempo se trabaje para cumplir con los requerimientos.

La adecuación se define como la capacidad del producto de software para proporcionar un conjunto apropiado de funciones para tareas y objetivos de usuario especificados. En la arquitectura propuesta se dieron solución a las propuestas hecha por los usuarios y se tuvieron en cuenta todos los requisitos funcionales y no funcionales que tendría el software. La arquitectura desarrollada presenta una buena adecuación puesto que en ella esta soportado todo lo necesario para desarrollar simuladores con características semejantes. La construcción de de DFI, la creación de reportes, manejo y conversión de unidades de medidas de distintas magnitudes, las simulación matemática de los modelos, requisitos que son soportados en la arquitectura propuesta y da la medida de la adecuación del producto.

La exactitud es la capacidad del producto software para proporcionar los resultados o efectos correctos o acordados, con el grado necesario de precisión, es decir cuan precisa es la arquitectura en cuanto al resultado final de una acción solicitada al producto. La arquitectura esta diseñada para cumplir con este parámetro de calidad en toda su extensión, los requisitos planteados fueron diseñados en función de la exactitud, ejemplo de ellos son los cálculos que debe manejar la aplicación en las conversiones de unidades de medidas, las herramientas de diseño deben además poseer una idea clara de los resultados modelados, los reportes emitidos no pueden contener errores ya que de ellos se toman importantes decisiones por parte de los usuarios del producto.

La interoperabilidad es la capacidad del producto software para interactuar con uno o más sistemas especificados, o sea la manera de poder enviar, compartir y manipular información con otros sistemas, en este parámetro podemos resaltar elementos como la manera en que son manejados los ficheros que se guardan en la aplicación los cuales son en formato XML, que es un modo de lenguaje de texto compatible y manejado por la mayoría de la aplicaciones hoy en día, otra funcionalidad es la posibilidad de exportar los diferentes diagramas a formatos de imágenes (jpg, bmp), lo cual permite al usuario trabajar con el diagrama en otros sistemas que se encarguen de la manipulación de imágenes. Los reportes son otro de los elementos que demuestran la presencia de este atributo de calidad en la arquitectura, ya que los reportes a los cuales arroja el simulador se pueden exportar a otros formatos como pdf, doc.

La seguridad de acceso no es más que la capacidad del producto software para proteger información y datos de manera que las personas o sistemas no autorizados no puedan leerlos o modificarlos, al tiempo que no se deniega el acceso a las personas o sistemas autorizados. Este atributo no es muy necesario

resaltarlo puesto que es más aplicable a arquitecturas donde se maneje información en bases de datos o datos confidenciales, en el caso de la arquitectura propuesta no es necesario este atributo, puesto que no se maneja información que haya que proteger en ese sentido.

Fiabilidad.

La fiabilidad del sistema está relacionada con la capacidad de este para reaccionar ante fallos internos o externos que puedan afectarlo, en este sentido existen dos elementos fundamentales la madurez y la tolerancia a fallos. La madurez es la capacidad del producto software para evitar fallar como resultado de fallos en el software, o sea es la capacidad de respuesta del software ante anomalías del producto. En el caso de la arquitectura propuesta, este atributo es ampliamente tratado, ejemplo de ello es la manera que son tratadas las excepciones que pueden darse, para ello existe un tratamiento de errores, los cuales son capturados y mostrados en una ventana común que es capaz de conectar al usuario con una explicación en la ayuda sobre el error al tiempo que este es tratado internamente en caso de que la solución no este en manos del usuario. La manipulación y entrada de números es uno de los pilares de la aplicación y la arquitectura provee componentes que se encargan de capturar y procesar la entradas no válidas al sistema.

La tolerancia a fallos es la capacidad del software para mantener un nivel especificado de prestaciones en caso de fallos del software o de infringir sus interfaces. La arquitectura tiene definidos componentes que no permiten la entrada de cadenas no válidas, y en caso de que el usuario se valga de otros recursos para lograrlo, se le enseña a través de recursos visuales, donde está el error y no procesará la información hasta que esta sea entrada correctamente.

Usabilidad.

Muchas veces se piensa que la usabilidad es un atributo al que se le da solución en la capa de presentación y no tiene más influencia en el resto de la arquitectura. Y también son muchas las veces en que hay que decirle al cliente que no se puede hacer algo relacionado con la usabilidad porque afectaría cosas que van más allá de la capa de presentación. Aunque la mayoría de los elementos relacionados con la facilidad de uso del programa están encapsulados en la capa de presentación existen otros más generales que son tratados a otros niveles. Un aspecto importante del simulador relacionado con la usabilidad es la posibilidad de que los usuarios puedan entrar los datos en el sistema de medidas que

deseen, aunque esto en parte se soluciona en el componente Generico, el núcleo de esta funcionalidad se resuelve en Intermedia. Si en algún momento tiene que ser sustituida la interfaz de usuario no se perdería el núcleo funcional de la conversión de unidades. Otro aspecto importante esta relacionado con darle información al usuario de la naturaleza de los errores que pueda presentar el programa, con este fin se definió una estrategia de tratamiento de errores que se integran a la ayuda del programa, cuando ocurre un error el usuario es notificado y se le da posibilidad de que consulte la ayuda relacionada con ese error. La facilidad con que el usuario puede manejar los DFI se debe en gran medida a la utilización de controles de usuario para representar los elementos que aparecen en un diagrama. Estos controles son fáciles de manipular y se puede tener un mayor control sobre su comportamiento. Además de que proveen una apariencia agradable al sistema.

Eficiencia.

El Simulador de Procesos de la Industria Química Cubana es un software de escritorio, que no usa recursos de red ni necesita procesadores demasiado potentes ni con gran capacidad de almacenamiento como servidores. Las características necesarias para que el software funcione correctamente son los requerimientos que impone el Framework.NET, pues es la única precondición que debe cumplirse para instalar el software. Estando en esta situación la preocupación que se tuvo en cuenta mientras se diseñaba la arquitectura fueron el consumo de memoria y de tiempo del procesador. El uso de controles de usuario de .NET para la definición del editor gráfico es un punto que compromete un poco el consumo de memoria, pero esta decisión fue puesta en una balanza, por un lado la rapidez y facilidad para la creación de nuevos componentes para el editor gráfico y del otro el uso de un poco más de memoria, evidentemente la decisión fue de sacrificar una cantidad pequeña de memoria y quedarse con todas las facilidades que ofrecen los controles de usuario. Para la optimización del uso del CPU se usaron algoritmos eficientes a la hora de realizar procedimientos complejos que necesitaran una capacidad de procesamiento mayor, el uso de eventos y delegados también propició la disminución del consumo de tiempo de procesador al usarlos como alternativa a componentes que hacen chequeo todo el tiempo como son los ActionList que consumen gran cantidad de tiempos de reloj.

Portabilidad.

La portabilidad es un aspecto importante del software, y las decisiones arquitectónicas tienen una gran influencia en esta característica. La primera decisión importante en aras de que el producto fuera portable fue la selección de la tecnología de implementación. Con el Framework de .NET se pueden desarrollar aplicaciones para Windows y permite instalar el programa en cualquier ambiente que tenga instalado el Framework. Por otra parte el producto puede ser ejecutado sobre plataforma Linux, teniendo como precondición que ya este instalado el Mono, que es el equivalente del Framework en Linux. El Mono todavía es una herramienta precaria que no está al nivel de su homólogo de .NET, tiene aún problemas para el soporte de controles creados en .NET y para la comprensión de los ficheros de recursos, lo que hace muy difícil la reutilización de algunas partes de la interfaz gráfica creada en Windows. La arquitectura en capas posibilidad que se pueda hacer con mayor facilidad una migración a Linux, el hecho de separar la interfaz de usuario de la capa de negocio hace posible que en caso necesario se sustituya completamente la capa de presentación sin que las capas inferiores sufran cambios.

Mantenebilidad.

Este atributo constituye un buen medidor de la robustez y flexibilidad del producto, y son fundamentales las decisiones arquitectónicas que se tomen. La arquitectura que se analiza en este trabajo tiene como característica fundamental su extensibilidad, esta fue diseñada de manera que se pueda simular cualquier tecnología bajo la concepción modular-secuencial, resulta relativamente fácil la inclusión de una nueva tecnología de simulación y nuevos módulos de cálculo. El hecho de que esté basada en componentes con interfaces bien definidas posibilita la sustitución de alguno de estos componentes con la menor influencia en los demás. En la actualidad la persistencia se basa en la creación de ficheros utilizando la serialización, si en un futuro se quiere usar una base de datos solo hay que cambiar el componente de acceso a datos SalvarCargar, sin necesidad de que los demás se vean afectados.

Creciendo la arquitectura.

La característica fundamental de la arquitectura es su extensibilidad. Por la manera en que fue definida es posible la implementación de simuladores del tipo modular secuencial. Los equipos de la industria encapsulados en modelos matemáticos fueron concebidos como componentes independientes que se le pueden agregar o quitar a la aplicación. La definición de un nuevo módulo se hace fácil y rápidamente, el

nivel de dificultad dependerá de la complejidad del modelo matemático. El nuevo componente se obtiene siguiendo una serie de pasos definidos, heredando de clases existentes, implementando interfaces, definiendo la apariencia visual del nuevo módulo y sobrescribiendo métodos. El procedimiento para la inclusión de un nuevo equipo al simulador se describe a continuación.

El primer paso es la creación de un nuevo control heredado el cual es una especialización de la clase CtrModulo del paquete Generico. Este control se personaliza visualmente incluyéndole la imagen que representa al equipo real de la fábrica y agregándole los puertos, la cantidad de puertos dependerá lógicamente de las características específicas de cada equipo. En caso de que el módulo tenga parámetros particulares se agrega un formulario que hereda de la clase frmPropiedadesModulo (de Generico) y se diseña de acuerdo con las particularidades del módulo.

Una vez definidas las clases que garantizan la parte visual, se pasa a definir las clases de negocio, en primer lugar se define la clase que contendrá los modelos matemáticos, dicha clase hereda de NegModulo (de Intermedia), dentro de esta se sobrescribe el método CalculoModulo en el cual se implementan los cálculos necesarios para determinar los parámetros de las corrientes de salida y los parámetros adicionales de equipo. También se redefine el método GetInfo() para especificar que elementos son los que se muestran como información del módulo cuando el usuario lo requiera y el método Listo, especificando que condiciones se deben cumplir para se pueda calcular. Hasta este punto se cuenta con las clases visuales y de procesamiento de información, solo falta fijar los puntos y políticas de comunicación de un módulo con los demás, para ello se definen tantas clases de negocio como puerto tenga el equipo. Estas clases van a heredar de NegPuerto (de Intermedia) y van implementar una interfaz de flujo en dependencia del tipo de corriente que procesa el puerto específico, en el caso de las entrada, se redefine el método Compatibles.

Se crear un formulario cuando el módulo tiene parámetros particulares que deben ser introducidos por el usuario para garantizar los cálculos. Dicho formulario se va a diseñar utilizando para la entrada de datos numéricos el componente TextBoxNumber y el componente ComboBoxUM para la entrada de las unidades de medida asociadas a cada uno de los parámetros. En esta clase es necesario sobrescribir el método SetGetValores donde se define como se leen los datos entrados por el usuario y se le pasan a la clase que representa el negocio del módulo.

Cuando se hayan definido todas las clases del nuevo componente, se procede a relacionar las clases de negocio con la interfaz gráfica, para ellos se sobrescribe IniModulo en el control heredado, en este método se vincula el control gráfico con la clase de negocio que hereda NegModulo y cada uno de los puertos

gráficos con su correspondiente puerto del negocio. En el caso de que el módulo requiera un formulario para la entrada de datos este es el momento de decirlo.

Como se puede observar resulta muy fácil la inclusión de un nuevo equipo al simulador, si se quisiera incluir toda una nueva tecnología a simular, por ejemplo, la tecnología de la refinación de petróleo, solo sería necesario definir en Intermedia los tipos de corrientes que se usan en el proceso de refinación. Cuando se definen los tipos de corriente se están definiendo las interfaces que regirán las interacciones entre los futuros módulos.

Introduciendo una nueva tecnología de simulación

Cuando se quiere implementar una nueva tecnología de simulación el primer paso consiste en la definición de las interfaces de comunicación entre los futuros módulos de cálculo. Para ello se hace un análisis de los tipos de corrientes que existen en el proceso que se desea simular, por ejemplo en el caso de la obtención de azúcar crudo intervienen dos tipos de corrientes fundamentales, las azucaradas y las corrientes de agua-vapor. De este análisis se desprende que se deben definir en el componente Intermedia dos nuevas interfaces, que representarán a estos dos tipos de flujos: IFluAzucarado e IFluAgua. También es necesario definir los componentes de dichas corrientes, los cuales se representarán en el sistema también como interfaces que proveerán los métodos para acceder a dichos componentes. De forma general todas las corrientes tienen cuatro componentes comunes: presión, temperatura, flujo de agua y flujo másico total, las corrientes de agua-vapor no tienen ningún componente adicional, mientras las azucaradas tienen además flujo de sólidos solubles sacarosa, flujo de sólidos solubles no sacarosa, flujo de fibras y flujo de sólidos insolubles no fibras. Para representar que las corrientes contienen un determinado componente, esto se expresa haciendo que la interfaz que representa a la corriente implemente la interfaz que representa al componente. La jerarquía de clases que resulta de este análisis se muestra en la figura 18.

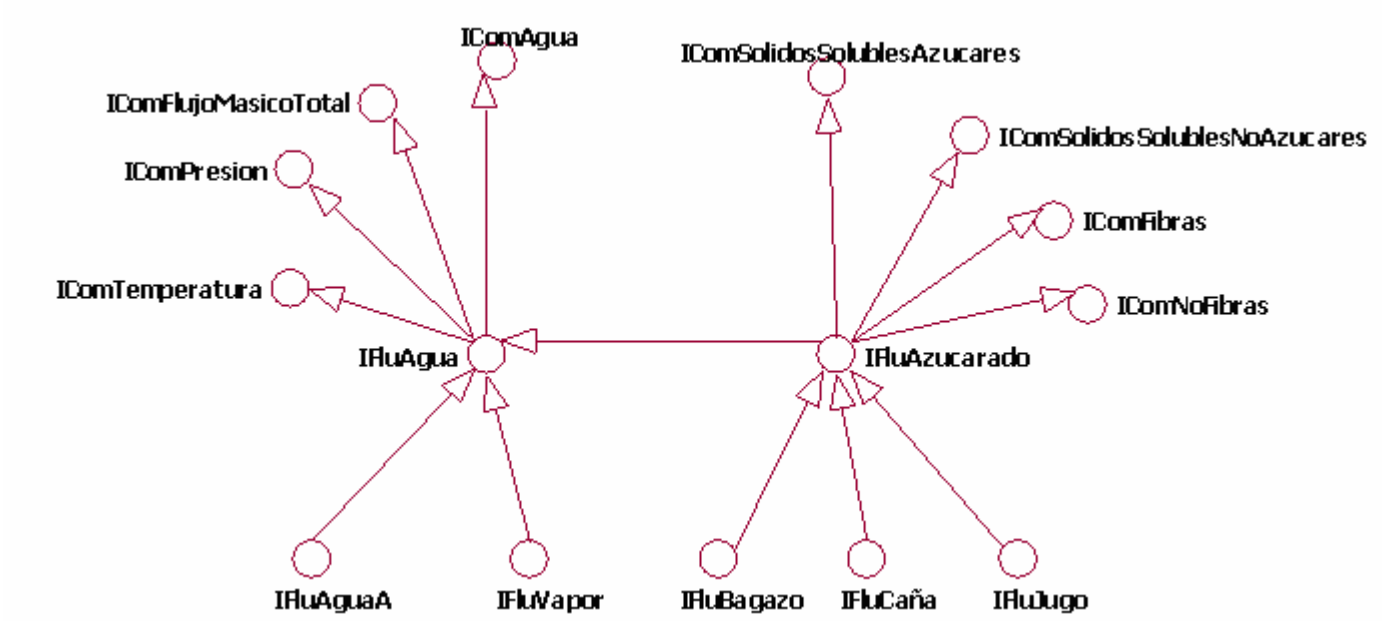

Figura 18: Interfaces de flujo.

En el diagrama aparecen otras interfaces, las cuales representan los flujos concretos que existen en el proceso y el hecho de que implementen IFluAgua o IFluAzucarado indica a que familia de corrientes pertenece. Cuando se hayan definido todas las interfaces relacionadas con los flujos del proceso, se puede decir que la arquitectura esta lista para soporta módulos de ese proceso.

Impacto de la arquitectura sobre el proyecto.

En el desarrollo de un proyecto de software intervienen muchos factores y si bien la arquitectura tiene un papel protagónico no es la única que decide el éxito del proyecto, pero por esto no deja de ser uno de los pilares que sustenta y lleva al traste un buen producto final. Hace 15 años la arquitectura de software era una ciencia que estaba naciendo, pero por eso no se dejaron de hacer buenos productos que hicieron impacto en su época. El simulador, que sirvió como punto de partida para el sistema al que la arquitectura propuesta da solución, data de los años 90, es un software desarrollado en Borland Pascal 6.0 que es un lenguaje estructurado y secuencial. El sistema carece de una arquitectura propiamente dicha ya que lo más que se asemeja a la programación actual es la división del código en subrutinas que son llamadas cuando se necesitan. Por otro lado carece de una interfaz de ventana y el DFI debe ser representado en papel para poder entender lo que se entra al sistema. El tratamiento de errores es ínfimo y precario, donde

incluso los mensajes que se muestran son los que se capturan del propio lenguaje y no presentan ninguna vinculación con la ayuda del sistema, la cual es completamente técnica y acerca de los aspectos químicos del simulador. En este momento, al terminar la arquitectura propuesta se cuenta con una plataforma para el desarrollo de simuladores del tipo modular secuencial, que permite la creación de un nuevo simulador con estas características en poco tiempo, siempre que los demás factores del proyecto como son la organización y la composición del equipo de desarrollo sean adecuados. La arquitectura desarrollada agiliza la incorporación de nuevos módulos puesto que la jerarquía de clases generaliza el comportamiento de los objetos, dejando en manos del los desarrolladores solo los elementos específicos de cada módulo. Por otro lado la presencia de una interfaz gráfica que interactúa con el usuario permite al usuario final que el desarrollo de DFI sea mucho más rápido, eficiente y menos costoso en errores. La manipulación de más de una unidad de medida es otro elemento que da la superioridad de la arquitectura desarrollada.

Conclusiones

1. Se realizó un estudio del estado del arte sobre arquitectura para simuladores en el mundo encontrando así, cual de las características presentes en ellos era la más apropiada para aplicar a la descrita en este trabajo.
2. La arquitectura diseñada es reusable porque permite la creación de nuevos simuladores modulares secuenciales con características semejantes, con tiempos de desarrollado relativamente cortos y sin cambios significativos a la arquitectura.
3. Al ser diseñada y desarrollada en componentes separados, la arquitectura permite la agregación, eliminación e implementación de componentes sin que la arquitectura tenga que cambiar mucho su estructura, lo que denota su flexibilidad.
4. Con el trabajo desarrollado se generaron los artefactos necesarios que contemplan y permiten la implementación del sistema a desarrollar.
5. Todos los resultados obtenidos y analizados son positivos a pesar que las métricas de arquitectura no son medibles numéricamente, evaluando así a la arquitectura como buena.
6. Se estableció una arquitectura base para simuladores, sobre la cual se pueden ir agregando nuevos componentes y funcionalidades.
7. Se logró con el desarrollo del presente trabajo, diseñar y modelar una arquitectura que cumple con los requisitos funcionales y no funcionales del sistema al cual da solución informática, por tanto se cumplió con el objetivo de diseñar una arquitectura robusta, flexible y reusable.

Recomendaciones

1. Se recomienda siempre que se vaya a realizar una arquitectura de este tipo hacer un estudio previo sobre los diseños arquitectónicos relacionados, con el objetivo de ganar en tiempo y costo de producción.
2. Se recomienda emigrar la arquitectura para otras plataformas de desarrollo como Mono en aras de poder implementar su uso en otros Sistemas Operativos como Linux.
3. Se recomienda hacer un estudio para el desarrollo de posteriores versiones sin el uso de los controles de usuario, lo cual es un recurso de la plataforma seleccionada, esto permitirá que sea aun más portable y extensible la arquitectura.
4. Se recomienda el uso de herramientas Cases que generen código para incrementar la productividad en el orden del tiempo de desarrollo.
5. Se recomienda eliminar el uso de herramientas específicas de una plataforma para hacer el sistema más portable hacia otras plataformas de desarrollo.

Bibliografía

1. Booch, G., Rumbaugh, J. y Jacobson, I. (2000). El Proceso Unificado de Desarrollo del Software. Madrid: Pearson Educación S.A.

2. Garlan, D. y Shaw, M. (1993). An Introduction to Software Architecture. New Jersey, World Scientific Publishing Company.

3. Monroe, R. T., Kompanek, D., Melton, R. y Garlan, D. (1996). Stylized Architecture, Design Patterns, and Objects. Consultado en Enero 23, 2007 en http://www.cs.cmu.edu/afs/cs.cmu.edu/project/able-10/OldFiles/ftp/ieee_arch_pattern_obj_submitted.pdf.

4. Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., Stal, M (1996). Pattern-Oriented Software Architecture: A System of Patterns. John Wiley,

5. Reynoso, C. B. (2004). Introducción a la Arquitectura de Software. Consultado en Enero 21, 2007 en http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.

6. Zachman, J. A. (1987). A Framework for Information Systems Architecture. : .

7. Kazman, R., Bass, L., Abowd, G. y Webb, M. (1994). SAAM: A Method for Analyzing the Properties of Software Architectures. Consultado en Febrero 21, 2007 en http://sunset.usc.edu/~neno/teaching/s99/SAAM.pdf.

8. Bass, L., Clements, P. y Kazman, R. (2003). Software Architecture in Practice. : Addison Wesley.

9. Larman, C. (2004). UML y Patrones. La Habana: Félix Varela.

10. Casanovas, J. (2004). Usabilidad y arquitectura del software. Consultado en Enero 10, 2007 en http://www.alzado.org/articulo.php?id_art=355.

11. Fowler, M. (2004). ¿Ha muerto el diseño? Consultado en Enero 15, 2007 en http://www.programacionextrema.org/articulos/designdead.es.html.

Glosario de términos

A lo largo del documento el lector puede encontrar palabras o siglas que le pueda afectar en su comprensión del mismo. Aquí se muestran algunas de estos elementos.

- **Corriente** – Se refiere a los flujos de distintas sustancias que existen en la industria, por ejemplo, en el caso de la industria azucarera existen corrientes de jugo, meladura, vapor, agua, etc.
- **DFI** – Diagrama de Flujo de Información, se refiere a los diagramas que dibujan los ingenieros químicos u otros especialistas que modelan la estructura de la fábrica que se desea simular.
- **Módulo** – Cuando se utiliza la palabra módulo se refiere a una abstracción de los equipos reales de la industria, cada módulo en el sistema tiene correspondencia con un equipo de la vida real, a excepción de los módulos lógicos que fueron incluidos para simular otros aspectos de la industria.
- **Puerto** – Se refiere a los puntos que permiten la conexión entre módulos, los puertos pueden ser de entrada o salida.
- **Diagrama** – Se refiere a la representación esquemática del DFI sobre la aplicación.

OPINIÓN DEL TUTOR DEL TRABAJO DE DIPLOMA

Título: **Desarrollo de la arquitectura del sistema para la simulación de procesos en la Industria Química Cubana.**
Autores: **Yurisnel Corrales Valdés, Juan Carlos Suárez López.**
Tutor: **Ing. Arnaldo Gandol Álvarez.**

Los diplomantes mostraron gran independencia en el desarrollo del trabajo de diploma si se toma en cuenta además que su tutor no se encontró presencialmente en todo el curso del trabajo, además de una gran originalidad y creatividad al abarcar objetivamente toda la arquitectura del proyecto, adoptando posiciones críticas. Un trabajo arduo e incansable incluso en días no laborables, y contribuyendo con la formación de sus compañeros de años inferiores, demostrando asimismo una gran responsabilidad en el cumplimiento de las tareas asignadas y en el trato directo con los clientes del producto.

El trabajo presenta buena ortografía, redacción y concordancia entre las ideas, además de una buena estructuración de los contenidos y un alto rigor científico.

Cabe destacar que este trabajo de diploma tiene un precedente pues se ha presentado en otro formato y redacción en forma de ponencia en jornadas científico-estudiantil y forum de ciencia y técnica obteniendo premios en todas las presentaciones.

Los beneficios de la solución de software obtenidos son tangibles en la formación docente en la especialidad de ingeniería química, y en la simulación de procesos en las industrias químicas cubanas, actualmente solo el MINAZ pero con la proyección futura de extenderlo a otras industrias debido a la arquitectura flexible y extensible que presenta.

Por todo lo anteriormente expresado considero que los estudiantes están aptos para ejercer como Ingenieros Informáticos; y propongo que se le otorgue al Trabajo de Diploma la calificación de 5 puntos. Debido a la calidad, rigor científico y resultados obtenidos, este trabajo se recomienda que se publique y presente en eventos de alta categoría.

Ing. Arnaldo Gandol Álvarez.

_____ Firma

_____ Fecha