Universidad de las Ciencias Informáticas

Facultad 9

Título: Arquitectura para integrar los módulos de la plataforma para el desarrollo de simuladores de procesos químicos.

Trabajo de diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor: Katia González Cabrera

Tutor: Ing. Yurisnel Corrales Valdés

Ciudad de la Habana, mayo 2010

AÑO 52 DE LA REVOLUCIÓN

Declaración de Autoría

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

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

_____ _____

Katia González Cabrera Yurisnel Corrales Valdés

Resumen

Con el surgimiento y evolución del uso de las tecnologías de la informática y las comunicaciones ha ocurrido un aumento en la aplicación de programas computarizados en todas las esferas de la sociedad en las que se desarrolla el ser humano. Para esto se llevan a cabo la realización de software que contribuyen al mejoramiento de las acciones que se realizan en las empresas, tal es así que uno de los procesos que en los últimos años ha tenido gran desempeño en el mundo informático es el desarrollo de simuladores.

La simulación se ha convertido en una poderosa herramienta a la hora de tomar decisiones apoyadas por los ordenadores, de ahí a que muchos procesos industriales se puedan realizar a través de la misma.

En Cuba se han desarrollado simuladores para la industria química y a su vez se han utilizado en ramas como la industria azucarera, en la medicina y otras.

Los simuladores han tenido gran importancia en el desarrollo computacional por lo que surge el propósito de este proyecto: desarrollar una plataforma mediante la cual se puedan informatizar los procesos químicos a través del uso de simuladores y de este trabajo en específico, el cual es: plantear por medio de este documento una arquitectura capaz de dar soporte a dicha plataforma.

La solución propuesta es diseñar una arquitectura de software (en adelante AS) que constituya una línea base para el desarrollo del proyecto. Para esto se ha realizado un análisis de los principales elementos que constituyen la arquitectura del software analizando sus conceptos, patrones y estilos arquitectónicos, para centrar su atención hacia el desarrollo de una propuesta arquitectónica que cumpla con los requisitos y con las expectativas del cliente.

Palabras claves:

- Arquitectura del software.
- Simulación.

A medida que transcurren los años el desarrollo de las tecnologías aumenta de manera considerable. Es un hecho que en los últimos tiempos ha surgido una proliferación en el uso de dichas tecnologías en todas las esferas de la vida social. Esto ha quedado demostrado en el uso de los ordenadores y de las funcionalidades que brindan para realizar cualquier actividad que el hombre se proponga. Los programas de software han contribuido de forma significativa a lograr lo antes mencionado y hoy día todas las empresas hacen uso de los mismos para lograr un mejor desempeño en su trabajo.

La industria química no es la excepción. Muchos de los procesos que se realizan en la misma se pueden llevar a cabo mediante la simulación y esta a su vez puede ser realizada a través de software llamados simuladores que no son más que herramientas de la ingeniería de procesos, capaces de realizarlos de forma computarizada.

Esta industria ha estado sumergida en un proceso de constante informatización. Muchos de los procesos que se desarrollan en la misma han estado en un perfeccionamiento continuo para lograr aumentar la calidad de las producciones y una disminución en los costos. La optimización de la industria y la modernización han sido factores claves para la puesta en práctica de simuladores que garanticen un incremento en la eficiencia de los procesos.

Desde la década de los años 60 se desarrollaron y aplicaron simuladores extranjeros 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. La mayoría de estos simuladores son del tipo deterministas y asumen condiciones de estado estacionario. Otros como el GASP II y el GPSS están orientados para la simulación probabilística de procesos y el SPEEDUP para la simulación dinámica. Algunos de los mencionados dieron lugar a nuevas versiones o nuevos desarrollos, como el GEMCS, GASP II, CHEMCAD, HYSYS, SUGARS, ASPEN PLUS y otros. (Vitoria, 2005).

En Cuba vinculadas a la industria azucarera y alcoholera se han desarrollado diversas bases de cálculos para la simulación del proceso, entre las que se tienen a TERMOAZUCAR (Pérez de Alejo, 1989), ACOPLA (Suárez, 1973), SIMFAD (Sabadi 1991) y SIDEL (Garrido, 2002). De estos programas desarrollados en nuestro país TERMOAZUCAR es el único de naturaleza modular-secuencial y que utiliza un sistema experto para el análisis de los resultados (Gozá, 2002). Los programas de simulación existentes para la Industria Alcoholera en nuestro país, ACOPLA, SIMFAD y SIDEL, se caracterizan por desarrollar una simulación poco flexible. Por otra parte TERMOAZUCAR cuenta con un solo

módulo de cálculo para la simulación de destilerías de alcohol con posibilidades muy limitadas para la evaluación de distintas tecnologías. Sin embargo, su naturaleza modular-secuencial hace que esté abierto a un continuo perfeccionamiento y que tenga una gran flexibilidad para la evaluación integral y sistémica de distintos esquemas. Estas características, junto con la posibilidad de implementar un análisis automatizado de la simulación con un sistema experto, hacen de TERMOAZUCAR una base de cálculo muy propicia para poder analizar con rapidez y rigor estudios energéticos que integren las industrias azucarera y alcoholera.

El desarrollo de los simuladores en el país se ha visto dificultado debido a que los software cubanos carecen de varias facilidades, sobre todo de las necesarias para el análisis y la síntesis de los resultados de las simulaciones de procesos completos y además los software extranjeros solo pueden utilizarse con fines académicos. Todo esto ligado a que actualmente los sistemas de información son cada vez más complejos, debido sobre todo a los cambios tecnológicos que actúan sobre ellos y que desafortunadamente las arquitecturas propuestas en su desarrollo no soportan en gran medida, la evolución constante a la que se ven sometidos surge la **situación problemática** del presente trabajo: En Cuba en la mayoría de los casos se emplean copias no autorizadas de simuladores extranjeros para la simulación de procesos, debido principalmente a que los software cubanos carecen de muchas de las funcionalidades que ofrecen estos simuladores, además la mayoría de estos software son creados para procesos en específico por lo que no se cuenta con una plataforma que permita desarrollar simuladores de procesos químicos en diferentes ambientes industriales. Dichos simuladores presentan una arquitectura que no es modificable por estar sujetos a licencias de software propietario en varias ocasiones, aunque no siempre. Además dicha arquitectura no es flexible a cambios ni modificaciones que permitan agregarle funcionalidades al simulador sin tener que rehacer una parte del mismo o rehacerlo completamente.

A partir de esto se plantea como **problema a resolver:** ¿Cómo integrar los módulos de la plataforma para el desarrollo de simuladores químicos del proyecto del grupo de simulación del polo Petrosoft en la Universidad de las Ciencias Informáticas (en adelante UCI)? Para dar respuesta al problema planteado se plantea como **objetivo general:** Proponer la arquitectura base de una plataforma de simuladores de procesos químicos del grupo de simulación del polo Petrosoft en la UCI. El **objeto de estudio** está enmarcado en el proceso de desarrollo de simuladores de procesos químicos del grupo de simulación de la facultad 9 de la UCI y el **campo de acción** es la arquitectura para una plataforma que permita

desarrollar simuladores de procesos químicos, por parte del grupo de simulación del polo Petrosoft en la UCI.

Se plantean como **objetivos específicos**:

- Analizar las tendencias arquitectónicas actuales para dar una mejor propuesta de solución.
- Definir patrones de diseño útiles para el proyecto, según los problemas recurrentes que aparezcan durante el desarrollo del mismo.
- Elaborar y construir la propuesta arquitectónica para desarrollar el sistema.
- Evaluar resultados a partir del cumplimiento de los objetivos planteados.

Idea a defender:

La definición e implementación de una arquitectura base que constituya un marco de trabajo para el desarrollo de simuladores permitirá una correcta integración de los subsistemas que lo componen.

Se han definido como **tareas de la investigación:**

- Síntesis de los principales aspectos sobre la Arquitectura y la Simulación de procesos químicos.
- Identificación y comparación de algunas propuestas de los estilos arquitectónicos.
- Identificación de los patrones arquitectónicos a utilizar.
- Definición de las actividades, artefactos, herramientas y tecnologías para el desarrollo de la arquitectura.
- Análisis de propuestas de arquitectura en sistemas similares al que se quiere desarrollar.
- Identificación de posibles componentes de reutilización que permitan un mejor desarrollo de las funcionalidades del sistema.
- Descripción de la propuesta de arquitectura.
- Evaluación de la arquitectura propuesta con el fin de identificar riesgos en su estructura y comprobar que el subsistema a desarrollar cumplirá con los requerimientos del cliente y las normas esperadas en cuanto a atributos de calidad.

Como **resultado final** se espera alcanzar la descripción de la arquitectura base de la aplicación que permita integrar los módulos de la plataforma para el desarrollo de simuladores de procesos químicos.

Los **métodos teóricos** utilizados para la realización de la investigación son el analítico-sintético al realizar un análisis de los elementos que describen la arquitectura de un software por separado y profundizar en el estudio de cada uno de ellos, para luego sintetizarlos en la solución de la propuesta. El análisis histórico lógico a través del cual se realizó un estudio de la evolución que ha presentado la Arquitectura del Software en el mundo durante los últimos años, además de sus repercusiones, para determinar las tendencias actuales de desarrollo de los modelos y enfoques arquitectónicos y mediante lo lógico no repetir totalmente lo histórico si no, tomar las cosas más importantes y que sean de ayuda en la investigación.

Los **métodos empíricos** empleados son la entrevista con el fin de planificar una conversación con personas expertas en el tema de arquitectura en la universidad para obtener información detallada sobre el tema.

El siguiente trabajo está estructurado en tres capítulos.

Capítulo 1: Fundamentación teórica. En este capítulo se realiza un estudio detallado de los principales conceptos y definiciones sobre la arquitectura, identificando los estilos y patrones arquitectónicos. Además se presenta una descripción de las principales herramientas y tecnologías a utilizar en el desarrollo de la plataforma.

Capítulo 2: Descripción de la arquitectura. Se hace una descripción detallada de la arquitectura a utilizar según la propuesta realizada, teniendo en cuenta patrones, lenguajes de programación, requisitos del sistema, etc.

Capítulo 3: Evaluación de la Arquitectura Propuesta. En este capítulo se hace un análisis de la solución propuesta y se evalúa el impacto de la arquitectura en el proyecto.

1.1 Introducción

En este capítulo se hace un análisis de los principales temas relacionados con la arquitectura. Dentro de su contenido se pueden encontrar diferentes temáticas como son: un acercamiento teórico a la Arquitectura del Software así como sus principales conceptos y principios, aspectos relacionados con los patrones arquitectónicos, las vistas de la arquitectura y las herramientas que se utilizan en el desarrollo de la misma para realizar una selección de las más adecuadas al proyecto en cuestión.

1.2 Estado del arte de la Arquitectura del Software.

La AS es una práctica joven, que en la actualidad, con el desarrollo de programas computacionales se ha convertido en una de las prácticas más usadas y a su vez estudiada por muchos interesados en el tema. Con el surgimiento de la arquitectura se logra una organización del proceso de desarrollo de un software y juega un papel importante a la hora de tomar decisiones para así evitar errores, encontrar fallas y reducir costos en un determinado proyecto.

1.2.1 Antecedentes

Los antecedentes de la AS están dados a partir del año 1968, cuando Edsger Dijktra de la Universidad Tecnológica de Eindhoven en Holanda y Premio Turing 1972 propuso que se estableciera una estructuración correcta de los sistemas de software antes de lanzarse a programar, escribiendo código de cualquier manera (Dijkstra., Enero de 1983). Aunque Dijkstra no utiliza el término arquitectura para describir el diseño conceptual del software, sus conceptos sientan las bases para el posterior desarrollo de los mismos siendo aplicada la arquitectura al proceso de desarrollo.

En la conferencia de ingenieros del software de la OTAN de 1969, un año después de la sesión en que se fundara la ingeniería de software, P. I. Sharp formuló estas apreciaciones comentando las ideas de Dijkstra:

“Lo que sucede es que las especificaciones de software se consideran especificaciones funcionales. Sólo hablamos sobre lo que queremos que haga el programa. Es mi creencia que cualquiera que sea responsable de la implementación de una pieza de software debe especificar más que esto. Debe especificar el diseño, la forma; y dentro de ese marco de referencia, los programadores e ingenieros deben crear algo. Ningún ingeniero o programador, ninguna herramienta de programación, nos ayudará, o ayudará al negocio del software, a maquillar un diseño feo. El control, la administración, la educación y todas las

cosas buenas de las que hablamos son importantes; pero la gente que implementa debe entender lo que el arquitecto tiene en mente. (Sharp, Octubre de 1969).

En la década de 1970 se pone en práctica el diseño estructurado y los primeros modelos explícitos de desarrollo del software. Entre 1977 y 1979 surgen los patrones cuyo originador fue Christopher Alexander, quien era arquitecto de edificios y había realizado estudios sobre el tema. Se puede destacar que tanto en los patrones como en la arquitectura la idea fundamental es la reutilización.

En los 80 surge un nuevo paradigma: la programación orientada a objetos y no fue sino hasta 1992 que se realiza el primer estudio en que aparece la expresión "arquitectura de software" en el sentido en que hoy se conoce planteado por Perry y Wolf (Wolf, Octubre de 1992.). En él, los autores proponen concebir la AS por analogía con la arquitectura de edificios, una analogía que unos encontraron útil y para otros pocos ha devenido inaceptable (Reed., 2001).

En el período de los 90 se desarrollaron los temas relacionados con estilos arquitectónicos, se elaboraron los lenguajes de descripción de arquitectura (ADLs) y se consolidó la concepción de las vistas arquitectónicas temas que serán tratados más adelante con una mayor profundización.

No obstante y a pesar de los adelantos que han ocurrido desde el surgimiento de la AS queda mucho por explorar aún en el campo de la informática y todo lo que se ha hecho en la ingeniería puede ser cambiado aplicando ahora los nuevos conceptos de AS. Esto no quiere decir que lo que existe hoy día no pueda ser actualizado y mejorado en un futuro.

1.2.2 Definiciones

Se pueden encontrar varias definiciones para la arquitectura e inclusive todos los arquitectos, unánimemente, no respaldan una definición determinada. Sin embargo este término aparece frecuentemente en los temas de Ingeniería del Software. Esto se puede encontrar en el Proceso Unificado como una de las características de su ciclo de vida que se define como centrado en la arquitectura para así lograr una visión común del sistema completo en la que el equipo de proyecto y los usuarios deben estar de acuerdo, por lo que se describe los elementos del modelo que son más importantes para su construcción, los cimientos del sistema que son necesarios como base para comprenderlo, desarrollarlo y producirlo económicamente. (Rumbaugh, 1999)

Una definición reconocida es la de Clements, en la cual plantea: La AS es, a grandes rasgos, una vista del sistema que incluye los componentes principales del mismo, la conducta de esos componentes según se la percibe desde el resto del sistema y las formas en que los componentes interactúan y se coordinan para alcanzar la misión del sistema. La vista arquitectónica es una vista abstracta, aportando el más alto nivel de comprensión y la supresión o diferimiento del detalle inherente a la mayor parte de las abstracciones. (Clements, 1996)

David Garlan establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño. (Garlan, 2000)

La definición oficial de AS se ha acordado que sea la estipulada en el documento de la IEEE estándar 1471-2000 y que es adoptada también por Microsoft que plantea: "La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución."

A pesar de las diferencias que existen en cuanto a las definiciones de la AS, se puede decir que entre los autores se llega a un punto común y es que la misma posee un alto nivel de abstracción, tema que será tratado en la sección que se encuentra a continuación.

1.2.3 Principales conceptos

Estilos: Es un concepto descriptivo que define una forma de articulación u organización arquitectónica. Los estilos conjugan elementos, conectores, configuraciones y restricciones. La descripción de un estilo puede ser en lenguaje natural o en diagramas, pero es mejor hacerlo en un lenguaje de descripción arquitectónica. Algunos ejemplos son las arquitecturas basadas en flujos de datos, las peer to peer, las de llamada y retorno, las centradas en datos o las de intérprete máquina virtual.

Lenguajes de descripción arquitectónica (ADLs en adelante): Lenguaje descriptivo de modelado que se focaliza en la estructura de alto nivel de la aplicación antes que en los detalles de implementación de sus módulos concretos (Vestal, Febrero de 1993). Los ADL que existen actualmente en la industria son alrededor de veinticinco, algunos de ellos han experimentado dificultades en su desarrollo o no se han impuesto en el mercado de las herramientas de arquitectura. Todos ellos oscilan entre constituirse como ambientes altamente intuitivos con eventuales interfaces gráficas o presentarse como sistemas rigurosamente formales con profusión de notación simbólica, en la que cada entidad responde a algún teorema. Los ADLs permiten modelar una arquitectura mucho antes que

se lleve a cabo la programación de las aplicaciones que la componen, analizar su adecuación, determinar sus puntos críticos y eventualmente simular su comportamiento.

Vistas:

Es una representación de un conjunto de sistema y la relación asociadas entre ellos. En el contexto de tipos de vistas y estilos, una vista puede analizarse como un estilo que es limitado a un sistema particular. (Clements, 2002)

La arquitectura de un sistema cuenta con múltiples vistas, asociadas a diferentes dimensiones o estructuras del sistema, se encuentran dirigidas a usuarios particulares y asociadas a requisitos no funcionales concretos, además ninguna vista concreta constituye la arquitectura de un sistema.

Las 4+1 vistas de la arquitectura del software están conformadas por dígase 4 vistas que son guiadas por una rectora que es la vista de casos de uso. Las demás vistas son la lógica, la de procesos, de despliegue y la de implementación.

Vista de casos de uso: Esta vista representa un subconjunto del artefacto Modelo de casos de uso y lista los casos de usos o escenarios del modelo de casos de uso más significativos, con las funcionalidades centrales del sistema. Si el sistema se hace extenso entonces se debería organizar en paquetes, lo cual facilitaría la comprensión de la vista de casos de uso.

Vista lógica: Esta vista representa un subconjunto del artefacto Modelo de diseño, la cual representa los elementos de diseño más importantes para la arquitectura del sistema. Este describe las clases más importantes, su organización en paquetes y subsistemas. También describe las realizaciones de casos de uso más importantes como por ejemplo las que describen aspectos dinámicos del sistema.

Vista de procesos: Esta vista suministra una base para la comprensión de la organización de los procesos de un sistema, ilustrados en el mapeo de las clases y subsistemas en procesos e hilos. Solo suele usarse cuando el sistema presenta procesos concurrentes o hilos.

Vista de despliegue: Esta vista suministra una base para la comprensión de la distribución física de un sistema a través de nodos. Suele utilizarse cuando el sistema está distribuido. Y hay una traza directa del modelo de implementación, puesto que cada componente físico debe estar almacenado en un nodo, esto incluye también la asignación de tareas provenientes de la vista de procesos en los nodos.

Vista de implementación: Esta vista describe la descomposición del software en capas y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica ahora para la implementación.

Procesos y Metodologías:

En los diferentes marcos, las vistas estáticas se corresponden con las perspectivas particulares de los diferentes participantes (stakeholders), mientras que las vistas dinámicas tienen que ver con etapas del proceso, ciclo de vida o metodología, caracterizadas como requerimiento, análisis, diseño (o construcción, o modelado), implementación, integración (prueba de conformidad, evaluación). La terminología, lo mismo que la articulación temporal del proceso o el ciclo, depende de cada marco. Durante varios años en la Arquitectura de Software la metodología discurrió sin elaborarla más que circunstancialmente, como si se estimara compatible con las prácticas establecidas en Ingeniería de Software, cualesquiera fuesen: RUP*, RAD*, RDS*, ARIS*, CMM*. Hoy en día la metodología dominante en la industria es tal vez el Modelo de Madurez de la Capacidad (CMM), aunque el Instituto de Ingenieros del Software (SEI) no la considera formalmente como tal.

Abstracción: Una abstracción denota las características esenciales de un objeto que lo distinguen de otras clases de objeto y provee de este modo delimitaciones conceptuales bien definidas, relativas a la perspectiva del observador (Booch, 1991). Para la IEEE y otros autores como Shaw la abstracción consiste en extraer las propiedades esenciales, o identificar los aspectos importantes, o examinar ciertos aspectos del problema posponiendo o ignorando los detalles menos sustanciales.

1.2.4 Lenguajes de Descripción

Aunque no existe hasta hoy una definición consensuada y unívoca de ADL, comúnmente se acepta que un ADL debe proporcionar un modelo explícito de componentes, conectores y sus respectivas configuraciones. Además se desea que un ADL suministre soporte de herramientas para el desarrollo de soluciones basadas en arquitectura y su posterior evolución. Los ADLs se utilizan para satisfacer requerimientos descriptivos de alto nivel de abstracción que las herramientas basadas en objeto en general y UML no cumplen satisfactoriamente. Por esta razón UML no es un ADL en sentido estricto ya que no se presta a las mismas tareas que los lenguajes descriptivos de la arquitectura diseñados para esa finalidad. A pesar de esto puede utilizarse, no tanto como un ADL sino como multilenguaje para simular otros ADLs (Jason E. Robbins, Abril, 1998).

1.2.5 Ventajas que aporta la AS

Comunicación mutua: La AS representa un alto nivel de abstracción común que la mayoría de los participantes, si no todos, pueden usar como base para crear un entendimiento mutuo, formar un consenso y comunicarse entre sí. En sus mejores expresiones, la descripción arquitectónica expone las restricciones de alto nivel sobre el diseño del sistema, así como la justificación de decisiones arquitectónicas fundamentales.

Decisiones tempranas de diseño: La AS representa la encarnación de las decisiones de diseño más tempranas sobre un sistema. Es una especie de puerta de peaje: el desarrollo no puede proseguir hasta que los participantes involucrados aprueben su diseño.

Restricciones constructivas: Una descripción arquitectónica proporciona modelos parciales para el desarrollo e indica los componentes y las dependencias entre ellos. Por ejemplo, una vista en capas de una arquitectura documenta típicamente los límites de abstracción entre las partes. Esto a su vez identifica las principales interfaces y establece las formas en que unas partes pueden interactuar con otras.

Reutilización, o abstracción transferible de un sistema: La AS personifica un modelo relativamente pequeño e intelectualmente tratable, de la forma en que un sistema se estructura y sus componentes se entienden entre sí. Este modelo es transferible a través de sistemas; en particular, se puede aplicar a otros sistemas que exhiben requerimientos parecidos y puede promover reutilización en gran escala. El diseño arquitectónico soporta reutilización de grandes componentes o incluso de Frameworks en el que se pueden integrar componentes.

Evolución: La AS puede exponer las dimensiones a lo largo de las cuales puede esperarse que evolucione un sistema. Haciendo explícitas estas paredes perdurables, quienes mantienen un sistema pueden comprender mejor las ramificaciones de los cambios y estimar con mayor precisión los costos de las modificaciones. Esas delimitaciones ayudan también a establecer mecanismos de conexión que permiten manejar requerimientos cambiantes de interoperabilidad, prototipado y reutilización.

Análisis: Las descripciones arquitectónicas aportan nuevas oportunidades para el análisis, lo cual incluye verificaciones de consistencia del sistema, conformidad con las restricciones impuestas por un estilo, conformidad con atributos de calidad, análisis de dependencias y análisis específicos de dominio y negocios.

Administración: La experiencia demuestra que los proyectos exitosos consideran una arquitectura viable como un logro clave del proceso de desarrollo industrial. La evaluación crítica de una arquitectura conduce típicamente a una comprensión más clara de los requerimientos, las estrategias de implementación y los riegos potenciales. (Billy, 2004)

La AS permite a los ingenieros mayor control en el sistema en el proceso de desarrollo desde sus inicios y promueve la temprana identificación de errores. Como resultado la arquitectura puede guiar el proyecto al éxito en vez de librarlo del fracaso.

1.3 Estilos Arquitectónicos

En este epígrafe se analizará lo relacionado con los estilos arquitectónicos presentes en la AS, se verán temas como sus definiciones y algunos ejemplos.

1.3.1 Concepto y definiciones

Estilos Arquitectónicos: Un estilo describe una clase de arquitectura, o piezas identificables de las arquitecturas empíricamente dadas. Esas piezas se encuentran repetidamente en la práctica, trasuntando la existencia de decisiones estructurales coherentes. Una vez que se han identificado los estilos, es lógico y natural pensar en re-utilizarlos en situaciones semejantes que se presenten en el futuro (Kazman, 2001). Igual que los patrones de arquitectura y diseño, todos los estilos tienen un nombre: cliente-servidor, modelo-vista-controlador, tubería-filtros, arquitectura en capas, etc.

Mary Shaw y Paul Clements (Clements, Abril de 1996) identifican los estilos arquitectónicos como un conjunto de reglas de diseño que identifica las clases de componentes y conectores que se pueden utilizar para componer en sistema o subsistema, junto con las restricciones locales o globales de la forma en que la composición se lleva a cabo.

En un ensayo de 1996 en el que aportan fundamentos para una caracterización formal de las conexiones arquitectónicas, Robert Allen y David Garlan (Garlan, 1996) asimilan los estilos arquitectónicos a descripciones informales de arquitectura basadas en una colección de componentes computacionales, junto a una colección de conectores que describen las interacciones entre los componentes. Consideran que esta es una descripción deliberadamente abstracta, que ignora aspectos importantes de una estructura arquitectónica, tales como una descomposición jerárquica, la asignación de computación a los procesadores, la coordinación global y el plan de tareas. En esta concepción, los estilos califican como una macro-arquitectura, en tanto que los patrones de diseño (como por ejemplo el MVC) serían más bien micro-arquitecturas.

En 1999 Mark Klein y Rick Kazman proponen una definición según la cual un estilo arquitectónico es una descripción del patrón de los datos y la interacción de control entre los componentes, ligada a una descripción informal de los beneficios e inconvenientes aparejados por el uso del estilo. Los estilos arquitectónicos, afirman, son artefactos de ingeniería importantes porque definen clases de diseño junto con las propiedades conocidas asociadas a ellos. Ofrecen evidencia basada en la experiencia sobre la forma en que se ha utilizado históricamente cada clase, junto con razonamiento cualitativo para explicar por qué cada clase tiene esas propiedades específicas (Kazman, Octubre de 1999).

1.3.2 Descripción de algunos estilos arquitectónicos.

Después de haber visto algunas de las definiciones de estilos arquitectónicos se muestran a continuación algunos ejemplos y sus descripciones.

Estilos de Flujo de Datos: Enfatiza la reutilización y la modificabilidad. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos.

- Tubería y filtros: Es el estilo que más temprano se definió y que puede identificarse con menor ambigüedad. El sistema tubería-filtros se percibe como una serie de transformaciones sobre sucesivas piezas de los datos de entrada. Los datos entran al sistema y fluyen a través de los componentes.

Estilos Centrados en Datos: Enfatiza la integrabilidad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento.

- Arquitecturas de Pizarra o Repositorio: En este estilo están presentes dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. Este tipo de arquitectura se ha usado en aplicaciones que requieren complejas interpretaciones de procesos de señales, o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados. También se han usado en procesos de lotes de bases de datos.

Estilos de Código Móvil: Enfatiza la portabilidad

- Arquitectura de Máquinas Virtuales: El estilo comprende básicamente dos formas o sub-estilos, que se han llamado intérpretes y sistemas basados en reglas. El estilo en su conjunto, se utiliza habitualmente para construir máquinas virtuales que reducen el vacío que media entre el engine de computación esperado por la semántica del programa y el

engine físicamente disponible. Las aplicaciones inscriptas en este estilo simulan funcionalidades no nativas al hardware y software en que se implementan, o capacidades que exceden a (o que no coinciden con) las capacidades del paradigma de programación que se está implementando. Dado que hasta cierto punto las máquinas virtuales no son una opción sino que no son viables en ciertos contextos, nadie se ha entretenido identificando sus ventajas y desventajas.

Estilos Peer-to-Peer: También llamada de componentes independientes, enfatiza la modificabilidad por medio de la separación de las diversas partes que intervienen en la computación. Consiste por lo general en procesos independientes o entidades que se comunican a través de mensajes. Cada entidad puede enviar mensajes a otras entidades, pero no controlarlas directamente. Los mensajes pueden ser enviados a componentes nominados o propalados[i] mediante broadcast[ii].

- Arquitecturas Basadas en Eventos: Las arquitecturas basadas en eventos se vinculan históricamente con sistemas basados en actores, demonios y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia (bajo la forma de disparadores, por ejemplo), en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos y en editores sintácticamente orientados para proporcionar verificación semántica incremental.

- Arquitecturas Orientadas a Servicios (SOA).

Este estilo de arquitectura, es una metodología o una estrategia en la cual las aplicaciones hacen uso de los servicios disponibles en la red, estos servicios representan procesos de negocio y que se combinan entre sí para ofrecer soluciones adecuadas a las diferentes necesidades de negocio, ofreciendo un marco de trabajo para alinear los procesos de negocio con los sistemas de las Tecnologías de la Información. SOA no es una tecnología, una herramienta, servicios Web, arquitectura para todo tipo de aplicaciones, ni tampoco una arquitectura de rápida implementación. El éxito de SOA depende fuertemente del grado de interoperabilidad dado a los servicios y de la reutilización los mismos.

Estilos de Llamada y Retorno: Enfatiza la modificabilidad y la escalabilidad. Son los estilos más generalizados en sistemas en gran escala.

A continuación en la figura 1 se puede identificar el funcionamiento del estilo arquitectónico.

- Model-View-Controller (MVC).

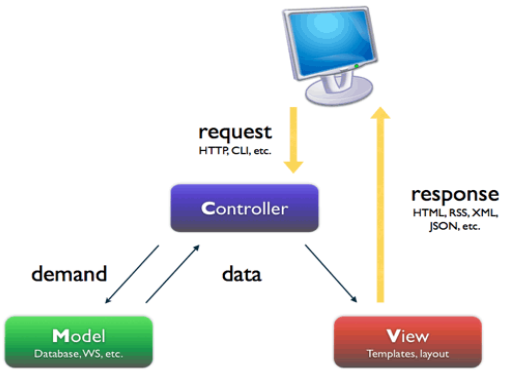

Figura 1 Patrón MVC

Separa el modelado del dominio, la presentación y las acciones basadas en datos ingresados por el usuario en tres partes diferentes:

Modelo. El modelo administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador).

Vista. Maneja la visualización de la información.

Controlador. Interpreta las acciones del ratón y el teclado, informando al modelo y/o a la vista para que cambien según resulte apropiado.

En la figura 2 se puede observar la estructura del estilo en capas el cual forma parte de los estilos de llamada y retorno.

- Arquitecturas en Capas.

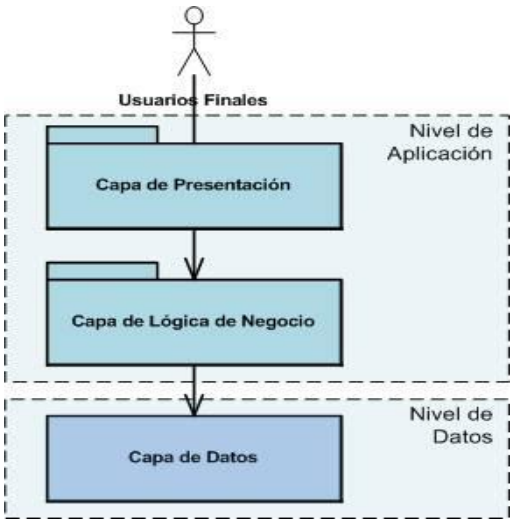

Figura 2 Arquitectura en Capas

Los sistemas o arquitecturas en capas constituyen uno de los patrones que aparecen con mayor frecuencia mencionados, o, por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente. Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

Casos representativos de este estilo son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware. El ejemplo más característico es el modelo OSI (Open System Interconnection, sus siglas en inglés), Interconexión de Sistemas Abiertos con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación. El estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos.

El estilo en capas posee varias ventajas como son: el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales, admite muy naturalmente optimizaciones y refinamientos y proporciona amplia reutilización.

- Arquitecturas Orientadas a Objetos.

Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. Se basan en principios orientados a objetos: encapsulamiento, herencia y polimorfismo. Las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación. En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

Estilo propuesto

Después de un análisis detallado de los estilos anteriormente descritos y debido a las características del proyecto, se propone utilizar el estilo arquitectónico MVC, ya que este es el estilo más recomendado en aplicaciones que utilicen JAVA como lenguaje de programación, separa los conceptos de diseño, y por lo tanto ocurre un decremento en la duplicación de código, la centralización del control y hace que la aplicación sea más extensible. MVC también ayuda a los desarrolladores con diferentes habilidades a enfocarse en sus habilidades principales y a colaborar a través de interfaces claramente definidos.

El estilo arquitectónico MVC tiene una característica fundamental y es que cualquier cambio en el modelo se refleja en cada una de las vistas. El modelo es el responsable de acceder a la capa de almacenamiento de datos, de definir las reglas del negocio y de llevar un registro de las vistas y los controladores del sistema. El controlador recibe los eventos de entrada y contiene las reglas de gestión de eventos. Mientras que las vistas, las cuales tienen cada una asociado un componente controlador, reciben los datos del modelo y los muestran al usuario y tienen registros de su controlador asociado.

Cada una de las características que presenta el estilo provee al desarrollador de más facilidades a la hora de programar. El correcto uso del estilo planteado en una arquitectura en el ciclo de vida de un software, proveerá de una mayor organización en todo el proceso de desarrollo del producto.

1.4 Patrones Arquitectónicos

En este epígrafe se podrán encontrar temas relacionados con los patrones arquitectónicos y se hará una explicación de los principales patrones de asignación de responsabilidades y de diseño.

1.4.1 Definición

Un patrón es una solución que se repite, tiene básicamente 4 secciones: nombre, descripción del problema, solución al problema y consecuencias de usar la solución. Los patrones han surgido para darle respuesta a problemas que se presentan continuamente en el desarrollo de un producto software, no buscan descubrir nuevos principios o expresiones en la ingeniería del software sino que intentan reutilizar los ya existentes de una forma más eficaz y flexible. Los patrones se han convertido en un valioso instrumento para la descripción y reutilización del conocimiento empleado durante las distintas fases del ciclo de vida del software.

Christopher Alexander es el arquitecto que primero estudió el concepto de patrones, en el contexto de construcción de edificios y comunidades. Escribió, ya en 1977: "Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, y además, describe el núcleo de la solución a ese problema, de tal manera, que podemos usar esa solución un millón de veces más en el tiempo, sin que tenga que ser la misma cada vez". El escribía acerca de patrones en la arquitectura, pero lo que describe, se puede aplicar a la ingeniería del software.

Pero es con el trabajo de Gamma, Helm, Johnson, Vlissides, "Design Patterns", en 1995, subtitulado "Elementos de software orientado a objetos reusables", cuando el tema se pone maduro. Estos autores, conocidos afectuosamente como "GoF" (la "Gang of Four", la banda de los cuatro), son los que toman el trabajo algo monumental, de hacer un catálogo de los patrones de diseño que hasta ese momento habían aparecido, y en una lista de 23 patrones, tratan de enumerar el conocimiento acumulado sobre el tema.

En la tabla 1 se puede encontrar la clasificación de los patrones de arquitectura, así como algunas de sus características.

1.4.2 Categorías

Patrones de Arquitectura	Relacionados a la interacción de objetos dentro o entre niveles arquitectónicos	Problemas arquitectónicos, adaptabilidad a requerimientos cambiantes, performance, modularidad, acoplamiento.	Patrones de llamadas entre objetos (similar a los patrones de diseño), decisiones y criterios arquitectónicos, empaquetado de	Diseño inicial

			funcionalidad.	
Patrones de Diseño	Conceptos de ciencia de computación en general, independiente de aplicación.	Claridad de diseño, multiplicación de clases, adaptabilidad a requerimientos cambiantes.	Comportamiento de factoría, Clase-Responsabilidad-Contrato (CRC)	Diseño detallado
Patrones de Análisis	Usualmente específicos de aplicación o industria.	Modelado del dominio, completitud, integración y equilibrio de objetivos múltiples, planeamiento para capacidades adicionales comunes.	Modelos de dominio, conocimiento sobre lo que habrá de incluirse (p. ej. logging & reinicio)	Análisis
Patrones de Proceso u Organización	Desarrollo o procesos de administración de proyectos, o técnicas, o estructuras de organización.	Productividad, comunicación efectiva y eficiente.	Armado de equipo, ciclo de vida del software, asignación de roles, prescripciones de comunicación.	Planea-miento
Idiomas	Estándares de codificación y proyecto.	Operaciones comunes bien conocidas en un nuevo ambiente, o a través de un grupo. Legibilidad, predictibilidad.	Sumamente específicos de un lenguaje, plataforma o ambiente.	Implementación, Mantenimiento, Despliegue.

Tabla 1 Clasificación (parcial) de patrones (Sarver, 2000)

1.4.3 Patrones de asignación de responsabilidades (GRASP)

Los patrones de asignación de responsabilidades se aplican durante la fase de elaboración, de los diagramas de iteración, al asignar responsabilidades a los objetos y diseñar la colaboración entre ellos.

Principales Patrones GRASP

Experto en Información: Asigna responsabilidades a las clases que tienen la información necesaria para cumplir con la responsabilidad.

Creador: Guía la asignación de responsabilidades relacionadas con la creación de objetos. El propósito fundamental es encontrar un creador que se conecte con el objeto producido en cualquier evento.

Alta Cohesión; Cada elemento del diseño debe realizar una labor única dentro del sistema.

Bajo Acoplamiento: Plantea que debe haber pocas dependencias entre las clases.

Controlador: Se encarga de asignar la responsabilidad de controlar el flujo de eventos del sistema, a clases específicas.

Además existen cuatro patrones GRASP adicionales:

- Fabricación Pura.
- Polimorfismo.
- Indirección.
- No hables con extraños.

1.4.4 Patrones de Diseño (GoF)

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces. Son un esqueleto básico que cada desarrollador puede adaptar a las peculiaridades y requerimientos de su aplicación.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características, una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores, otra es

que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Objetivos de los patrones de diseño:

- Proporcionar catálogos de elementos reusables en el diseño de sistemas software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre diseñadores.
- Estandarizar el modo en que se realiza el diseño.
- Facilitar el aprendizaje de las nuevas generaciones de diseñadores condensando conocimiento ya existente.

Por otra parte estos patrones no pretenden:

- Imponer ciertas alternativas de diseño frente a otras.
- Eliminar la creatividad inherente al proceso de diseño.
- No es obligatorio utilizar los patrones siempre, sólo en el caso de tener el mismo problema o similar que soluciona el patrón, siempre teniendo en cuenta que en un caso particular puede no ser aplicable.
- Abusar o forzar el uso de los patrones puede ser un error.

Principales Patrones GoF (Gang of Four)

Estos patrones presentan ciertas características que los hacen de gran utilidad en el diseño de una aplicación y otras que no pero igual son de gran ayuda para el desarrollador. Entre ellas se pueden mencionar que cada uno es independiente del resto de los patrones, se aplican en situaciones muy comunes ya que son soluciones simples que no indican cómo diseñar un sistema sino solo aspectos puntuales del mismo, el uso de un patrón no se refleja en el código ya que cuando se realiza la implementación es difícil determinar qué patrón de diseño se ha usado. Otra de las características es que es difícil reutilizar la implementación de un patrón ya que en la misma aparecen clases concretas, además suponen una sobrecarga de trabajo pues se usan más clases y es necesario delegar más mensajes.

Patrones creacionales

Abstract Factory (Fábrica abstracta): Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.

Builder (Constructor virtual): Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.

Factory Method (Método de fabricación): Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que crear.

Prototype (Prototipo): Crea nuevos objetos clonándolos de una instancia ya existente.

Singleton (Instancia única): Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

Patrones Estructurales

Adapter (Adaptador): Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.

Bridge (Puente): Desacopla una abstracción de su implementación.

Composite (Objeto compuesto): Permite tratar objetos compuestos como si se tratase de uno simple.

Decorator (Envoltorio): Añade funcionalidad a una clase dinámicamente.

Facade (Fachada): Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.

Flyweight (Peso ligero): Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.

Proxy: (Apoderado) Mantiene un representante de un objeto.

Patrones de Comportamiento

Chain of Responsibility (Cadena de responsabilidad): Permite establecer la línea que deben llevar los mensajes para que los objetos realicen la tarea indicada.

Command (Orden): Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.

Interpreter (Intérprete): Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.

Iterator (Iterador): Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.

Mediator (Mediador): Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.

Memento (Recuerdo): Permite volver a estados anteriores del sistema.

Observer (Observador): Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.

State (Estado): Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.

Strategy (Estrategia): Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.

Template Method (Método plantilla): Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.

Visitor (Visitante): Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera. (Hophe-Woolf, 2003)

1.5 Metodologías de desarrollo del Software.

Cuando se va a realizar un determinado proyecto, definir una metodología de desarrollo es sin duda una de las actividades que posee un elevado grado de importancia. Esto se debe a que dicha metodología se convierte en un plano en el cual se pueden apoyar los desarrolladores del software, para así lograr clientes satisfechos con el resultado final y a la vez lograr quién debe hacer qué, cuándo y cómo dentro del equipo de trabajo.

A continuación una panorámica de las principales metodologías que se pueden definir para un proyecto con sus principales características y descripción.

1.5.1 Metodologías robustas. Proceso Unificado de Desarrollo (RUP)

La metodología RUP, llamada así por sus siglas en inglés Rational Unified Process, divide en 4 fases el desarrollo del software:

- Inicio: El objetivo en esta fase es determinar la visión del proyecto.
- Elaboración: El objetivo es determinar la arquitectura base.
- Construcción: El objetivo es obtener la capacidad operacional inicial.
- Transición: El objetivo es obtener el release del proyecto.

RUP es un proceso de desarrollo del software que se caracteriza por ser preparado para desarrollar grandes y complejos proyectos, orientado a objetos, utiliza el UML como lenguaje de representación visual y su ciclo de vida se caracteriza por ser dirigido por casos de uso, centrado en la arquitectura e Iterativo e incremental. Presenta 9 flujos de trabajo los cuales son modelamiento del negocio, requisitos, análisis y diseño, implementación, pruebas, instalación, administración del proyecto, administración de configuración y cambios y ambiente.

Los elementos del RUP son:

- Actividades: Son los procesos que se llegan a determinar en cada iteración.
- Trabajadores: Son las personas o entes involucrados en cada proceso.
- Artefactos: Un artefacto puede ser un documento, un modelo, o un elemento de modelo.

Objetivos del RUP:

- Proporcionar una guía del orden de las actividades de los equipos.
- Especificar cuáles artefactos deben ser desarrollados y cuándo estos deben ser desarrollados.
- Dirigir las tareas de desarrolladores individuales y equipos como una sola.
- Ofrecer criterios para monitorear y medir los productos y actividades del proyecto.

A continuación se muestra en la figura 3 una vista de las fases e iteraciones por las que está compuesta la metodología RUP.

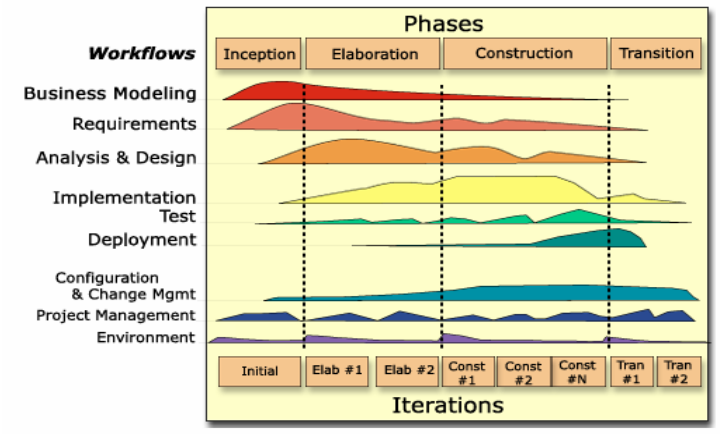

Figura 3 Fases e iteraciones de la metodología RUP

1.5.2 Enfoque ágil

En el 2001, motivados por la observación de que en muchas compañías los equipos de software estaban atascados cada vez más en el lodo de los procesos, un grupo de expertos de la industria se reunieron para establecer los valores y principios que permitan a los equipos de software trabajar rápidamente y responder al cambio. Ellos se denominaron a sí mismos la Alianza Ágil. Durante varios meses este grupo trabajó para crear una declaración de valores. El resultado es El Manifiesto de la Alianza Ágil. (Jacobson, 2000)

Manifiesto para el Desarrollo de Software Ágil

Se están descubriendo mejores maneras de desarrollar software haciéndolo y ayudando a otros a hacerlo. A través de este trabajo se valora:

- Los individuos y las interacciones sobre los procesos y las herramientas.
- Software operativo sobre documentos detallados.
- Colaboración del cliente sobre la negociación de contratos.
- Responder a los cambios sobre seguir un plan. (Martínez, 2005)

El enfoque Ágil considera que el problema básico del desarrollo de software es el riesgo, consecuente con la premisa de incertidumbre inevitable. Para enfrentar el riesgo, el enfoque ágil propone ciclos cortos de versiones y dentro de ellas, iteraciones de una a cuatro semanas; implementar las características de más alta prioridad; integrar al cliente al equipo de desarrollo; crear y mantener conjuntos detallados de pruebas, el enfoque ágil, renuncia a

la anticipación a escala de iteración y de sistema. Adopta una actitud reactiva, en el sentido de Roger S Pressman. También estima que el diseño no es dibujar un puñado de esquemas y entonces implementar el sistema conforme a esos esquemas. Se reconoce que los esquemas pueden aportar beneficios, pero que no ofrecen una retroalimentación concreta, es decir de funcionamiento. Con esta última idea, el enfoque ágil marca profundamente su vocación exploratoria y rompe el dogma que exige un buen diseño antes de la implementación. Una vez más se hace patente la premisa de incertidumbre inevitable que no se puede eliminar por mucho que se piense.

Selección de la metodología a utilizar

El análisis de los principios planteados tanto por las metodologías ágiles como por las robustas aportan una visión clara de que con un proyecto de tesis como el que se quiere desarrollar para obtener el producto: plataforma para el desarrollo de simuladores de procesos químicos, en una primera versión, no es necesario seguir los pasos de RUP como es típico en la UCI. En primer lugar, sólo se cuenta con cinco desarrolladores de modo que es un equipo pequeño. El cliente adopta el papel de experto en este caso convirtiéndose en un trabajador más del equipo. Las ideas innovadoras del producto que se planea obtener exigen un avance en todas las áreas que contribuyan a su satisfactoria terminación por lo que la metodología no queda exenta. En resumen, se decidió la utilización de RUP ágil para el proyecto en cuestión.

1.6 Lenguaje de modelado UML

UML acrónimo de Unified Modeling Languaje no es un lenguaje de programación, es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. Su objetivo es representar el conocimiento acerca de los sistemas que se pretenden construir y las decisiones tomadas durante su desarrollo, tanto los representados por diagramas estáticos (casos de uso, diagrama de clases, etc.) como los dinámicos (diagramas de actividades, interacción, etc.).

UML ofrece soporte para clases, clases abstractas, relaciones, comportamiento por interacción, empaquetamiento, entre otros. Permite soporte para algunos de los conceptos asociados a la AS como los componentes, los paquetes, librerías y la colaboración.

UML 2.0: Es un lenguaje de modelado más extensible, permite la validación y ejecución de modelos. Es interesante destacar que el UML 2.0 puede definirse a sí mismo, es decir, su estructura y organización es modelable utilizando el propio UML 2.0; de esta manera, se da un ejemplo de utilización del UML en un dominio distinto al del desarrollo de software. Entre

sus principales características se encuentran la introducción de nuevas notaciones para los diagramas de actividad y otros diagramas.

1.7 Herramientas de desarrollo de la Ingeniería del Software

Se considera a las HDS como herramientas basadas en computadoras que asisten el proceso de ciclo de vida de software (IEEE, 2004), consolidadas en la literatura en la forma de Ingeniería de software asistida por computadora (CASE, por sus siglas en inglés). Esto es, software que se utiliza para ayudar a las actividades del proceso de software o software que es utilizado para diseñar y para implementar otro software (Sommerville, 2005). Permiten automatizar acciones bien definidas, reduciendo también la carga cognitiva del ingeniero de software, quien requiere libertad para concentrarse en los aspectos creativos del proceso. Las HDS automatizan metodologías de software y desarrollo de sistemas y se vinculan con los diferentes conceptos involucrados en el desarrollo.

1.7.1 Visual Paradigm

Es una herramienta UML profesional que soporta el ciclo de vida completo del desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. El software de modelado UML ayuda a una más rápida construcción de aplicaciones de calidad, mejores y a un menor coste. Permite dibujar todos los tipos de diagramas de clases, código inverso, generar código desde diagramas y generar documentación. La herramienta UML CASE también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos UML (Medina Pasaje, 2006).

Soporta notación UML 2.x, capacidades de ingeniería inversa y directa, generación de código, importación desde Rational Rose, generación de código e ingeniería inversa a la vez de los lenguajes: Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python. Adicional, soporta la generación de código en: C#, VB .NET, Object Definition Language (ODL), Flash ActionScript, Delphi, Perl, Objective-C, y Ruby.

Entre sus principales características se pueden encontrar:

- Licencia: Gratuita y Comercial.
- Producto de calidad.
- Soporta aplicaciones Web.
- Varios idiomas.

- Generación de código para Java y exportación como HTML.
- Fácil de instalar y actualizar.

1.7.2 Rational Rose Interprise

Unifica todos sus equipos de desarrollo a través del modelamiento, el cual está basado en UML. Se encarga de la modelación de los diagramas correspondientes al desarrollo. Es una forma de ayuda para la comprensión del sistema y de sus distintos componentes. Presenta como desventaja que es un software propietario y que no soporta otras plataformas como GNU/Linux.

Diagramas: Clases, Componentes, Deployment, Secuencia, Statechart, Caso de Uso, Colaboración.

Características adicionales incluidas:

- Modelado UML para trabajar en diseños de base de datos, con capacidad de representar la integración de los datos y los requerimientos de aplicación a través de diseños lógicos y físicos.
- Capacidad de crear definiciones de tipo de documento XML para el uso en la aplicación.
- Integración con otras herramientas de desarrollo de Rational.
- Provee visualización, modelado y las herramientas para desarrollar aplicaciones Web.

Selección de la herramienta a utilizar

Ambas herramientas de desarrollo poseen características que son comunes, pero se puede llegar a la conclusión después de un detallado estudio que la mejor candidata a utilizar es el Visual Paradigm. Esta selección se basa en que posee una característica fundamental: soporte multiplataforma, esto permitiría que la aplicación se pueda desarrollar en cualquier plataforma de trabajo. Otra de las ventajas que posee el Visual Paradigm es que se adecúa al entorno de desarrollo permitiéndoles a los diseñadores, analistas y a todo aquel que trabaje con el mismo, una mayor organización y claridad en el trabajo. Es muy sencillo de usar, fácil de instalar y actualizar, además genera código para varios lenguajes. Permite modelado orientado a objeto, y a la vez, orientado al Lenguaje Unificado de Modelado (UML), el cual fue escogido para el desarrollo del proyecto, además tiene algunas versiones gratuitas a diferencia de Rational Rose EE que solo funciona en el sistema operativo Windows y es un software propietario.

Visual Paradigm 6.4 for UML Enterprise Edition: Se determina usar esta versión debido a que ofrece Interoperabilidad con modelos UML2 (metamodelos UML 2.x para plataforma Eclipse) a través de XMI (XML Metadata Interchange), permitiendo que distintas herramientas de modelado puedan intercambiar modelos entre sí. Además presenta un editor de detalles de casos de uso que es muy utilizado para la especificación de los detalles de los casos de uso, incluyendo la especificación del modelo general y de las descripciones de los casos de uso. También presenta integración con la herramienta Visio, de gran ayuda a la hora de dibujar los diagramas UML con plantillas de MS Visio. Todas estas características permiten un mejor desarrollo a la hora de realizar los diagramas correspondientes, a la vez que proveen de más facilidades a los usuarios brindando una serie de opciones para el modelado.

1.8 Lenguajes de programación

Los lenguajes de programación permiten establecer una comunicación entre el hombre y la máquina. A medida que aumenta el desarrollo de la computación este tema es uno de los más controversiales a la hora de realizar una selección adecuada de un lenguaje que se adecúe a las características de un proyecto. De ahí a que se presente a continuación una descripción de algunos de los lenguajes candidatos a utilizar en el desarrollo de una aplicación desktop y la selección del mismo para ser utilizado en el desarrollo del proyecto.

1.8.1 Lenguaje de programación C++

Es un lenguaje orientado a objetos, el cual presenta como particularidad principal la posibilidad de redefinir los operadores (sobrecarga de operadores), y de poder crear nuevos tipos que se comporten como tipos fundamentales. El lenguaje C++ soporta varios estilos de programación, por ejemplo: procedural, orientado a objetos; intenta ser tan eficiente y portable como lo es el lenguaje C. Este lenguaje ha sido utilizado para el desarrollo de disímiles aplicaciones, que van desde sistemas de gestión y videojuegos, hasta sistemas operativos. Entre ellos se encuentran los Sistemas Operativos Unix, Linux y Windows.

1.8.2 Lenguaje de programación PHP

PHP es un acrónimo recursivo para "PHP Hypertext Pre-processor" (inicialmente PHP Tools o Personal Home Page Tools), su creador Rasmus Lerdorf ha recibido muchas contribuciones de otros desarrolladores debido a su política de código abierto. PHP es usado para la creación de aplicaciones para servidores, o creación de contenido dinámico para sitios web, ofreciendo soluciones simples y universales para las paginaciones dinámicas web de fácil programación. Como producto de código abierto, también es

multiplataforma y está ampliamente difundido en el mundo entre las comunidades de programadores.

1.8.3 Lenguaje de programación JAVA

Es un lenguaje orientado a objetos. Presenta un modelo de objetos más simple que C y C++ y elimina herramientas de bajo nivel que suelen introducir muchos errores. Fue pensado para servir como nueva manera de manejar complejidad del software. Se utiliza en una variedad de plataformas computacionales de los dispositivos.

Características de Java (Manual de JAVA.)

- **Es sencillo, orientado a objetos y familiar**: Sencillo, para que no requiera grandes esfuerzos de entrenamiento para los desarrolladores. Orientado a objetos, porque la tecnología de objetos se considera madura y es el enfoque más adecuado para las necesidades de los sistemas distribuidos y/o cliente/servidor. Familiar, porque aunque se rechazó C++, se mantuvo Java lo más parecido posible a C++, eliminando sus complejidades innecesarias, para facilitar la migración al nuevo lenguaje.

- **Robusto y seguro**: Robusto, simplificando la gestión de memoria y eliminando las complejidades de la gestión explicita de punteros y aritmética de punteros del C. Seguro para que pueda operar en un entorno de red.

- **Independiente de la arquitectura y portable**: Java está diseñado para soportar aplicaciones que serán instaladas en un entorno de red heterogéneo, con hardware y sistemas operativos diversos. Para hacer esto posible el compilador Java genera 'bytecodes', un formato de código independiente de la plataforma diseñado para transportar código eficientemente a través de múltiples plataformas de hardware y software. Es además portable en el sentido de que es rigurosamente el mismo lenguaje en todas las plataformas. El 'bytecode' es traducido a código máquina y ejecutado por la Máquina Virtual de Java, que es la implementación Java para cada plataforma hardware-software concreta.

- **Interpretado, multi-hilo y dinámico**: El intérprete Java puede ejecutar bytecodes en cualquier máquina que disponga de una Máquina Virtual Java (JVM). Además Java incorpora capacidades avanzadas de ejecución multi-hilo (ejecución simultánea de más de un flujo de programa) y proporciona mecanismos de carga dinámica de clases en tiempo de ejecución.

Algunas ventajas de JAVA

- Es una fuente abierta, así que los usuarios no tienen que luchar con los impuestos sobre patente pesados cada año.
- Independiente de la plataforma.
- El poder de Java API es alcanzado fácilmente por los reveladores.
- Java realiza la colección de basura de las ayudas, así que la gerencia de memoria es automática.
- Java asigna siempre objetos en el apilado.
- Usando JAVA se pueden desarrollar aplicaciones web dinámicas.
- Permite crear programas modulares y códigos reutilizables. (JAVA y sus ventajas, 2008)

Debido a que Java se ha desarrollado a la misma vez que Internet, es utilizado ampliamente en el desarrollo de sistemas distribuidos como aplicaciones web y además en el desarrollo de sistemas incrustados; dentro de los que sobresalen las aplicaciones para teléfonos celulares. Además permite desarrollar software en una plataforma y ejecutarlo en prácticamente cualquier otra plataforma, crear programas para que funcionen en un navegador web y en servicios web, desarrollar aplicaciones para servidores como foros en línea, tiendas, encuestas, procesamiento de formularios HTML, etc., combinar aplicaciones o servicios que usan el lenguaje Java para crear servicios o aplicaciones totalmente personalizados y desarrollar potentes y eficientes aplicaciones para procesadores remotos, productos de consumo de bajo coste y prácticamente cualquier tipo de dispositivo digital.

1.8.4 Lenguaje de programación C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Está aprobado como un estándar por la ECMA4 e ISO5. Su primera versión fue publicada en 2001; su desarrollo fue guiado por Anders Hejlsberg y está basado en los lenguajes Delphi, Visual Basic, C++ y Java, entre otros.

El lenguaje C# ha sido diseñado para ser robusto, moderno y sencillo. Promueve las prácticas sanas de programación y las reglas de globalización. Además, funciona bajo un entorno de recolección automática de la memoria, lo que aumenta la productividad del desarrollador. En este lenguaje no existe el nivel de visibilidad global ni la herencia múltiple. Por otro lado, los punteros solo pueden ser utilizados en contextos no manejados y la

memoria manejada no puede ser liberada explícitamente. Se destaca, además, la existencia de tipos genéricos, indexadores, delegados, eventos, clases parciales, estructuras e interfaces, entre otros rasgos. Con este lenguaje se han desarrollado sistemas de gran diversidad. Sobresalen entre ellos las aplicaciones web, las aplicaciones de escritorio y los componentes reutilizables.

Lenguaje de programación seleccionado

El lenguaje de programación a utilizar debido a sus características es JAVA. Esto se debe a que es un lenguaje multiplataforma: El mismo código java que funciona en un sistema operativo, funcionará en cualquier otro sistema operativo que tenga instalada la máquina virtual java, por lo que se pone de manifiesto la alta portabilidad. Es seguro y robusto ya que verifica el código al mismo tiempo que lo escribe y antes de ejecutarse por lo que se logra una mayor codificación sin errores. Presenta niveles de seguridad que aumentan las medidas de seguridad en la interfaz, así como varias rutinas de verificación para lograr que la construcción del lenguaje sea la apropiada, soporta la multitarea y es dinámico y distribuido. Presenta un mecanismo robusto de manejo eficiente de excepciones, ya que utiliza las del mismo lenguaje lo cual permite encapsular en clases los errores y separar el flujo de ejecución normal del tratamiento de errores.

El lenguaje JAVA, como se ha podido analizar, ofrece varias ventajas a la hora de programar una aplicación. Para el proyecto es de gran ayuda utilizarlo para así lograr de forma más eficiente desarrollar un producto que cumpla con las expectativas del cliente y que los desarrolladores tengan más facilidades para realizarlo.

1.9 Ambiente de desarrollo (IDE)

El ambiente de desarrollo (Development Environment) es algo imprescindible en la elaboración de un producto software. Es donde se definen el conjunto de herramientas y tecnologías (frameworks), versiones a usar y su integración, que intervienen en un proceso de desarrollo de software.

1.9.1 Eclipse

Eclipse es como una tienda donde no solamente se hacen productos, sino que además se hacen las herramientas para hacer los productos. Eclipse contiene un equipo de instrumentos para desarrollo en Java o Java Development Toolkit (JDT) para escribir y depurar programas en Java; además se obtiene un ambiente de desarrollo de plugin para

heredar de Eclipse. Si todo lo que se quiere es un IDE para Java, no se necesita nada además que el JDT. Esto es por lo que la mayoría las personas usan Eclipse.

Aunque Eclipse es escrito en Java y su principal uso es como IDE para Java, este es un lenguaje neutral. El soporte para desarrollo en Java es proveído por un componente enchufado o plug-in, pero además están disponibles plugins para otros lenguajes, como C/C++, Cobol, C#, PHP. En principio permite ejecutar un programa sobre cualquier plataforma. Es una extensible plataforma de código abierto para desarrollar herramientas.

Ventajas que ofrece el uso de Eclipse:

- Soporta la construcción de una variedad de herramientas para el desarrollo de las aplicaciones.
- Corre en una gran cantidad de sistemas operativos, incluyendo GNU/Linux y Windows.
- Mediante JDT (JAVA Development Tools) facilita la creación de aplicaciones programadas en JAVA. (Deepak Alur, 2003)

1.9.2 NetBeans

NetBeans IDE es una aplicación de código abierto diseñada para el desarrollo de aplicaciones fácilmente portables entre las distintas plataformas, haciendo uso de la tecnología Java. Dispone de soporte para crear interfaces gráficas de forma visual, desarrollo de aplicaciones web, control de versiones, colaboración entre varias personas, creación de aplicaciones compatibles con teléfonos, móviles y resaltado de sintaxis.

El IDE NetBeans es una herramienta para programadores, pensada para escribir, compilar, depurar y ejecutar programas; proporciona una arquitectura de aplicaciones fiables y flexibles. Está escrito en Java pero puede servir para cualquier otro lenguaje de programación. Existe además un número importante de módulos para extenderlo. Este IDE es un producto libre y gratuito sin restricciones de uso, posee código abierto, escrito completamente en Java usando la plataforma NetBeans, soporta el desarrollo de todos los tipos de aplicación Java (J2SE, web, EJB y aplicaciones móviles).

NetBeans contiene todos los módulos necesarios para el desarrollo de aplicaciones Java en una sola descarga, permitiéndole al usuario comenzar a trabajar inmediatamente. Desde Julio de 2006, es licenciado bajo la Common Development and Distribution License (CDDL), una licencia basada en la Mozilla Public License (MPL).

Selección del ambiente de desarrollo

Como ambiente de desarrollo a utilizar en el proyecto se propone Eclipse. Esto se debe a que cuenta con una amplia documentación y soporte en la comunidad JAVA. Es un potente editor con buen completamiento de código. Es una plataforma de desarrollo libre y de código abierto basada en JAVA. Es soportado por varios sistemas operativos como Windows y GNU/Linux, además de que presenta varias funcionalidades para el desarrollo de editores gráficos y que pueden ser utilizados en el proyecto.

La versión seleccionada para el desarrollo del proyecto es el Eclipse 3.3, esto se debe a que una de las principales características de esta versión es la sencillez con que se instalan los proyectos, ya que existe un repositorio único donde aparecen publicados todos los proyectos que la conforman.

1.10 Arquitectura en simuladores de procesos químicos.

La Arquitectura en simuladores se encuentra ligada a la funcionalidad, aunque está más relacionada con la concepción, prestaciones y desarrollo del software que con su funcionalidad final. No obstante existe una estrecha relación ya que la arquitectura se orienta dependiendo de la funcionalidad.

Uno de los factores a tener en cuenta a la hora de diseñar una arquitectura es el grado de concentración y el grado de distribución.

Una AS concentrada es aquella en la que hay una o pocas aplicaciones que concentran cada una un número elevado de funcionalidades y estas funcionan en uno o pocos ordenadores con poco flujo de comunicación exterior.

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.

Una arquitectura distribuida tiene varias o muchas aplicaciones funcionando en varios ordenadores con un flujo importante de comunicaciones. (Madrid)

La utilización de la arquitectura distribuida basada en una red de ordenadores personales tiene como objetivo global: obtener prestaciones razonables a un coste bajo.

Cuando se quiere implementar simuladores complejos la arquitectura distribuida, debido a sus características es la candidata perfecta.

Ventajas que ofrece la arquitectura distribuida:

- 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.
- 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. (Madrid)

Es conveniente proponer una AS 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 esté terminado el producto. Es importante tener en cuenta el hecho de que sea multiplataforma lo que amplíe sus posibilidades de uso, sobre todo en la actualidad donde se hace más extensivo el uso de diferentes sistema operativos aparte de Windows como es el sistema GNU/Linux. Entre los factores fundamentales al definir una arquitectura para un simulador, además de su grado de concentración o distribución, es que esta se organice de manera eficiente y se puedan aprovechar mejor los recursos con la adecuada selección de los estilos y patrones arquitectónicos.

Después de realizado un acercamiento teórico a la arquitectura en simuladores se puede enfatizar en el estándar CAPE-OPEN, como un simulador capaz de integrar simuladores ofreciendo interoperabilidad e integración de componentes para la simulación de procesos físicos y químicos. El estándar CAPE-OPEN también permite acceso a aplicaciones de propiedades físicas. Está diseñado pensando en una gran variedad de usuarios finales; que van desde expertos en especialidades, es decir, ingeniería de las reacciones químicas, con o ninguna o limitada experiencia en programación hasta personas con conocimientos y habilidades en las tecnologías de informatización usadas para crear las interfaces estandarizadas de CAPE–OPEN. (Team, 1997)

En Cuba se han desarrollado diversos simuladores especialmente vinculados a la industria azucarera y alcoholera, entre los que se encuentran TERMOAZUCAR, ACOPLA, SIMFAD y SIDEL. De estos programas desarrollados el TERMOAZUCAR es el único de naturaleza modular-secuencial, aunque fue creado para procesos específicos, utiliza un sistema experto para el análisis de los resultados. El simulador TERMOAZUCAR puede enlazarse coherentemente con un Programa Experto computarizado (el ANSTE), cuenta con un solo módulo de cálculo para la simulación de destilerías de alcohol, lo cual limita sus posibilidades de evaluación en distintas tecnologías. No obstante dicho simulador presenta una estructura y una programación flexible lo que permite nuevas adaptaciones en su desarrollo permitiendo así un perfeccionamiento del mismo. (AlejoLeon, abril 2005)

El Sistema Termoazúcar (STA) desarrollado en la plataforma .net presenta una arquitectura en capas y orientada a componentes. Las tres capas presentes en el simulador están definidas como: presentación, negocio y persistencia, en las cuales se definen los principales componentes que dan soporte al simulador.

Conclusiones Parciales

Como parte de darle cumplimiento al objetivo principal del presente trabajo, en este capítulo se ha hecho un acercamiento teórico al estado del arte de la AS, así como los aspectos fundamentales relacionados con los patrones arquitectónicos, estilos, tecnologías y herramientas para un mejor desempeño de los desarrolladores a la hora de complementar su trabajo.

Para la realización de la plataforma de desarrollo de simuladores químicos queda estructurada de la siguiente forma las tecnologías, herramientas y metodología, como una propuesta para su aplicación en el desarrollo del proyecto y como un apoyo para plantear una primera versión de la arquitectura del mismo. Como estilo arquitectónico a utilizar se propone Modelo Vista Controlador, metodología de desarrollo del Software: AUP, lenguaje de modelado: UML 2.0, como herramienta de desarrollo: Visual Paradigm 6.4, lenguaje de programación: JAVA y ambiente de desarrollo: Eclipse 3.3.

[i] Divulgados
[ii] Transmisión de un paquete que será recibido por todos los dispositivos en una red

Capítulo 2: Descripción de la arquitectura

2.1 Introducción

En este capítulo se puede encontrar la descripción de la Arquitectura propuesta para el proyecto. Se realiza la organización de los componentes que forman la Arquitectura detallando las 4+1 vistas de la arquitectura: la de casos de uso, lógica, procesos, despliegue e implementación. Se describen los requisitos funcionales que responden a los casos de uso arquitectónicamente significativos para la arquitectura, así como los requisitos no funcionales que el producto debe tener.

2.2 Descripción general de la plataforma de simuladores químicos

Un simulador de procesos químicos está compuesto por varios componentes: las operaciones unitarias, las corrientes, la interfaz gráfica y otros.

La plataforma para el desarrollo de simuladores de procesos químicos será una aplicación de escritorio, de esta forma se pueden obtener varias ventajas ya que al estar instalada en cada computadora aprovecha recursos del sistema operativo ofreciendo al usuario características propias del mismo, además la información contenida no estará expuesta a personas no autorizadas a través de Internet.

En una primera versión del producto se desarrollarán los módulos: Componentes de Simulación, el cual se encarga de manejar todo lo referente a las funcionalidades y el cálculo de las propiedades físico-químicas y Editor Gráfico en el cual se maneja todo lo relacionado a la interfaz de la aplicación con todas las opciones que esta pueda brindar para el diseño gráfico de Diagramas de Flujo de Información para el posterior proceso de simulación.

2.3 Metas y restricciones

Las metas y restricciones estarán condicionadas por los requisitos funcionales, que no son más que capacidades o condiciones que el producto debe cumplir y los requisitos no funcionales que son las propiedades o cualidades que el producto debe tener. (IEEE Std 610). A continuación se presentan dichos requisitos.

2.3.1 Requisitos Funcionales

RF1: Gestionar información.

RF1.1 Mostrar información de las operaciones unitarias.

RF1.2 Mostrar información de las corrientes.

RF1.3 Insertar información de las corrientes.

RF1.4 Insertar información de los operaciones unitarias.

RF2: Manipular unidades de medidas.

RF3: Crear Diagrama de Flujo de la Información (DFI).

RF3.1: Adicionar módulos al DFI.

RF3.2: Conectar módulos del DFI.

RF3.3: Copiar módulos

RF3.4: Cortar módulos

RF3.5: Pegar módulos

RF3.6: Mover los módulos en el DFI.

RF3.7: Eliminar módulos.

RF3.8: Rotar módulos.

RF4: Analizar sensibilidad.

RF5: Analizar indicadores.

RF6: Realizar análisis sobre corridas múltiples.

RF7: Reportar resultados.

RF7.1 Reportar todos los resultados.

RF7.2: Reportar resultados seleccionados.

RF8: Analizar economía.

RF9: Simular

RF9.1: Determinar orden de cálculo.

RF9.2: Calcular parámetros de las operaciones unitarias.

RF9.3: Calcular parámetros de las corrientes.

RF9.4: Calcular propiedades de componente.

RF10: Identificar operaciones unitarias.

RF11: Identificar corrientes.

RF12: Exportar datos a otros ficheros de texto.

RF13: Exportar DFI como imagen.

RF14: Calcular indicadores.

RF14.1: Calcular indicadores por área.

RF14.2: Calcular indicadores por equipos.

RF14.3: Calcular indicadores globales.

RF15: Salvar toda la información manejada en el simulador.

RF16: Reportar datos.

RF17: Cargar la información guardada.

RF18: Obtener propiedades de corriente.

RF19: Obtener propiedades de operaciones unitarias.

RF20: Calcular operaciones unitarias.

RF21 Gestionar Proceso.

RF21.1: Listar corrientes.

RF21.2: Crear corriente.

RF21.3: Eliminar corriente.

RF21.4: Listar operaciones unitarias.

RF21.5: Adicionar operaciones unitarias.

RF21.6: Eliminar operación unitaria.

2.3.2 Requisitos no Funcionales

1. Requisitos de Software:

- El subsistema será multiplataforma, no dependiendo su instalación de ningún sistema operativo en específico, solamente se requiere que esté instalada la Máquina Virtual de JAVA.

2. Requisitos de hardware:

- Computadoras personales con 516 MB de memoria RAM como mínimo.
- Al menos 1 GB de espacio libre en disco duro.

3. Requisitos de diseño e implementación:

- El sistema será una aplicación de escritorio.
- El sistema será implementado en el lenguaje JAVA.
- El sistema usará como IDE de desarrollo Eclipse.
- Se utilizó para realizar los modelos del sistema, UML y como herramienta de apoyo a este Lenguaje de Modelación el Visual Paradigm.
- Se utilizó para una mejor comprensión del código de programación los estándares de codificación establecidos por la Sun Microsystems para el lenguaje de programación JAVA.

4. Requisitos de usabilidad:

- Facilidad de aprendizaje: El software será diseñado para que sea utilizado por aquellos usuarios que tengan conocimientos de la simulación de procesos químicos, debido a que la información que se gestiona requiere de estos, aunque cualquier usuario, si tiene los permisos puede usarlo.
- Ayuda y documentación en línea: Se brindará un sistema de ayuda que explique las diferentes funcionalidades con que cuenta el sistema.

5. Requisitos de apariencia o interfaz externa:

- Interfaz amigable y fácil de usar.
- Debe 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.

6. Requerimientos de portabilidad:

- La herramienta podrá ser usada bajo cualquier sistema operativo.

2.4 Vistas Arquitectónicas

Una vez analizado en el capítulo anterior el tema referente a las vistas de la arquitectura se realizará un detallado análisis de las mismas pero teniendo en cuenta las características del proyecto en cuestión. El objetivo principal de las vistas arquitectónicas es describir los aspectos fundamentales del sistema, proporcionándole un lenguaje común a los desarrolladores.

2.4.1 Vista de Casos de Uso

Presenta un subconjunto del modelo de casos de uso. Lista los casos de uso o escenarios del modelo de casos de uso que representen funcionalidades centrales del sistema final, que requieran una gran cobertura arquitectónica o aquellos que impliquen algún punto especialmente delicado de la arquitectura.

En esta vista se podrán encontrar una breve descripción de los casos de uso que se clasificaron como significativos así como un diagrama de casos de uso donde queda explícita la relación entre el actor del sistema y los casos de uso que inicializa.

Los casos de uso clasificados como significativos para la arquitectura, debido a su impacto en el sistema son los siguientes:

Caso de uso Crear_DFI: En este caso de uso se encuentran agrupadas varias funcionalidades presentes en el desarrollo de los diagramas de flujo de la información, están incluidas aquellas que permiten agregar módulos, conectarlos, así como funciones básicas del editor como son copiar, cortar, mover, eliminar y rotar módulos. Para la arquitectura este caso de uso tiene una importancia significativa ya que constituye aquello que va a interactuar con el usuario, es lo que va a comunicar dichos usuarios con el simulador.

Caso de Uso Simular: El objetivo fundamental de la plataforma para realizar simuladores de procesos químicos es que sea capaz de realizar operaciones como el cálculo de propiedades físicas y químicas y de las corrientes que fluyen. Dicho cálculo realizado de forma correcta permitirá simular el proceso, por lo que el caso de uso es determinante para la propuesta de arquitectura inicial ya que de su resultado dependerá la comprobación de la validez de la arquitectura planteada.

Caso de Uso Reportar_Resultados: Este caso de uso será el encargado de proveer al usuario de un reporte de todos los resultados que se obtengan luego de realizado el proceso de simulación. Este proceso arroja una gran cantidad de información y que naturalmente constituye la base para el entendimiento y la organización de la misma.

Caso de Uso Gestionar_Información: Para realizar el proceso de simulación se hace necesario primeramente tener en cuenta toda la información de entrada que se le va a ofrecer al sistema. Esta puede ser tanto de operaciones unitarias como de las corrientes que están presentes en el flujo de información. El almacenamiento, para su posterior utilización constituye una parte importante a la hora de procesar los datos que se necesitan. De ahí a que este caso de uso se clasifique como crítico para el sistema ya que son importantes para la realización de los cálculos y para lograr la completitud del DFI.

Caso de Uso Salvar_Información: Toda la información que se obtiene después de realizado el proceso de simulación posee gran importancia ya que constituye el resultado que el usuario quiere alcanzar, así como que toda la información que se maneje durante el proceso, se guarde para su uso posterior. Este caso de uso va a permitir realizar estas operaciones y de ahí a que se clasifique como crítico para la arquitectura.

Caso de Uso Gestionar_Proceso: Para realizar el proceso de simulación se debe contar con una serie de funcionalidades que en su conjunto proveerán los aspectos que influyen en su realización. Para lograr esto se cuenta con una serie de operaciones, las cuales darán la oportunidad al usuario de interactuar con el sistema de forma correcta y van a permitir que el proceso se realice satisfactoriamente. Las operaciones que se podrán llevar a cabo sobre algunos de los componentes que condicionan la simulación de procesos químicos, como las corrientes y las operaciones unitarias, pueden ser listar, crear, eliminar y adicionar para luego proceder a su utilización y de ahí a que surja la importancia de este caso de uso para la arquitectura pues no será posible realizar el proceso sin antes haber realizado algunas de estas operaciones.

En el diagrama de la figura 4 se presenta al especialista como actor del sistema que inicializa cada caso de uso presente en el diagrama. O sea, cada caso de uso se inicializa cuando el especialista decide realizar alguna operación ya sea Crear_DFI, Simular, Reportar_Resultados, Mostrar_Informacion, Gestionar_Informacion.

Por otra parte el Editor Gráfico hace función de actor e inicializa el caso de uso Gestionar_Proceso, ya que este caso de uso va a inicializarse una vez que el usuario haya seleccionado una acción y esta requiera de las funciones que brinda el caso de uso. El usuario directamente no interactúa con este caso de uso por lo tanto no es quien lo inicializa como se muestra en el siguiente diagrama.

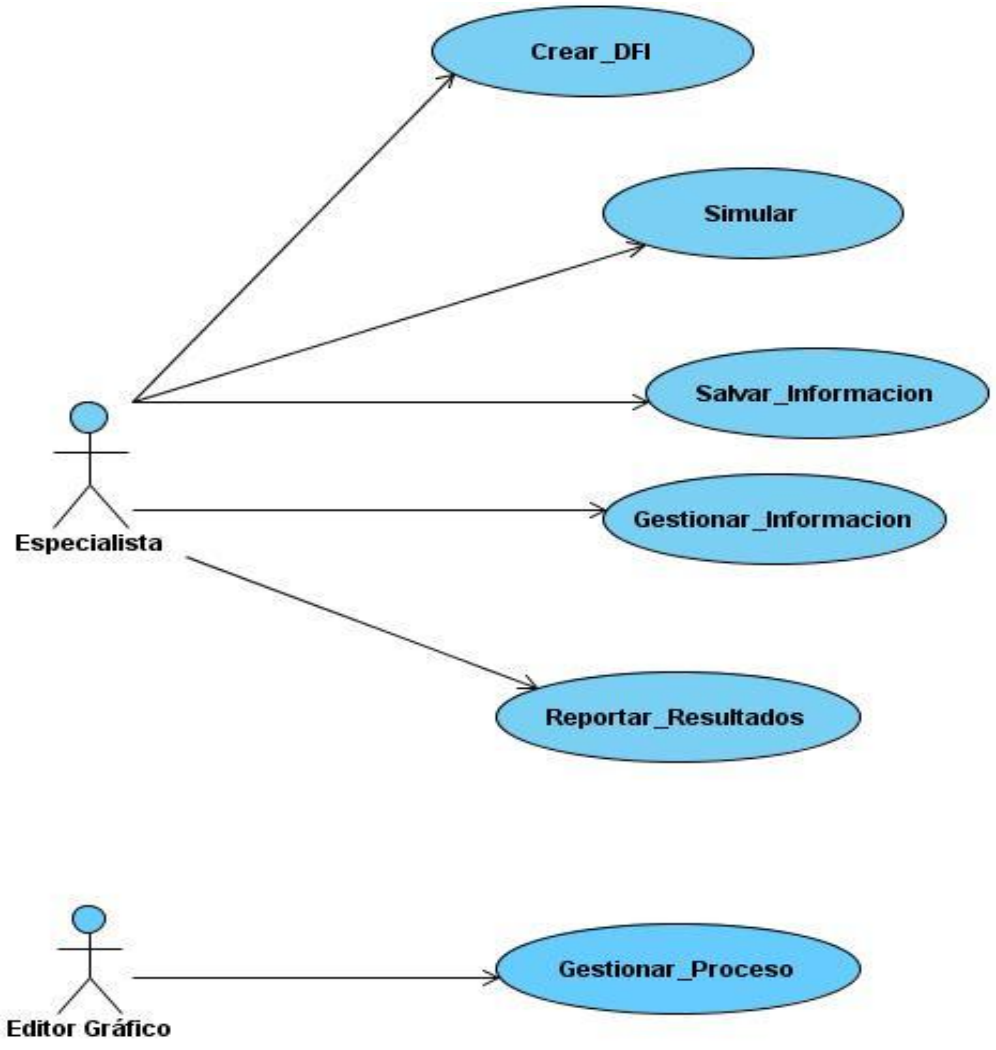

Figura 4 Diagrama de Casos de Uso Arquitectónicamente significativos

2.4.1.1 Breve descripción de los casos de uso arquitectónicamente significativos

Se verá a continuación una breve descripción de los casos de uso más significativos para la arquitectura. En la misma se hará un resumen de cada uno de ellos para brindarle una idea al lector de en qué consisten.

Caso de Uso:	Crear DFI
Actores:	Especialista
Resumen:	El Caso de Uso se inicia cuando el especialista selecciona la opción adicionar módulos al DFI, conectar, copiar, cortar, pegar, mover, eliminar o rotar módulos con el objetivo de conformar el diagrama del proceso que se desea simular.
Propósito:	Conformar el diagrama de proceso que se desea simular.
Referencias:	RF3.1, RF3.2, RF3.3, RF3.4, RF3.5, RF3.6, RF3.7,RF3.8
Prioridad:	Crítico

Tabla 2 Breve Descripción del Caso de Uso Crear DFI

Caso de Uso:	Gestionar_Informacion
Actores:	Especialista
Resumen:	El Caso de Uso se inicia cuando el especialista selecciona la opción mostrar información, ya sea de operaciones unitarias o corrientes que ha sido suministrada o la que se obtiene como resultado de la simulación e insertar información de entrada para el DFI.
Propósito:	Insertar o mostrar la información ya sea la suministrada por el usuario o la que se obtiene luego del proceso de simulación.

Referencias:	RF1.1, RF1.2, RF1.3, RF1.4
Prioridad:	Crítico

Tabla 3 Breve Descripción del Caso de Uso Gestionar_Información

Caso de Uso:	Simular
Actores:	Especialista
Resumen:	El caso de uso se inicia cuando el especialista desea simular un proceso químico y selecciona la opción simular.
Propósito:	Simular procesos químicos.
Referencias:	RF9.1, RF9.2, RF9.3, RF9.4
Prioridad:	Crítico

Tabla 4 Breve Descripción del Caso de Uso Simular

Caso de Uso:	Salvar_Informacion
Actores:	Especialista
Resumen:	El caso de uso se inicia cuando el especialista después de haber realizado el proceso de simulación desea guardar toda la información obtenida en los resultados alcanzados para su posterior análisis o en otro caso volver a utilizarla. El especialista selecciona la opción salvar información.
Propósito:	Guardar la información arrojada durante el proceso de simulación.
Referencias:	RF15

Prioridad:	Crítico

Tabla 5 Breve Descripción del Caso de UsoSalvar_Información

Caso de Uso:	Reportar_Resultados
Actores:	Especialista
Resumen:	El caso de uso se inicia cuando el especialista desea generar reportes de los datos manejados en la simulación, tanto los suministrados por el usuario como los que se obtienen después de simular para esto selecciona la opción Reportar todos los Resultados o si desea generar reportes de los datos específicos seleccionados por el usuario, para lo cual seleccionaría la opción Reportar Resultados seleccionados.
Propósito:	Obtener reporte de los resultados alcanzados en la simulación.
Referencias:	RF7.1, RF7.2
Prioridad:	Crítico

Tabla 6 Breve Descripción del Caso de Uso Reportar_Resultados

Caso de Uso:	Gestionar_Proceso
Actores:	Editor Gráfico
Resumen:	El Caso de Uso se inicia cuando el Editor Gráfico invoca algunas de las funciones para gestionar proceso ya sea listar corriente, crear corriente, eliminar corriente, listar operaciones unitarias, adicionar operaciones unitarias o eliminar operación unitaria. El subsistema

	ejecuta la función invocada y termina el CU.
Propósito	Gestionar las funciones realizadas sobre las corrientes u operaciones unitarias.
Referencias	RF21.1, RF21.2, RF21.3, RF21.4, RF21.5, RF21.6
Prioridad	Crítico

Tabla 7 Breve Descripción del Caso de Uso Gestionar_Proceso

2.4.2 Vista Lógica

Describe las partes arquitectónicamente significativas del modelo de diseño, cómo va a ser la descomposición en capas, subsistemas o paquetes. Una vez presentadas estas unidades lógicas principales, se profundiza en ellas hasta el nivel que se considere adecuado.

Paquetes y subsistemas de diseño

Para lograr una mejor comprensión se agrupan en paquetes las clases del diseño a partir de las funcionalidades que brindan. Esto con el fin de lograr un diseño que sea el más adecuado para la arquitectura de cualquier sistema que se quiera desarrollar. Para esto cada paquete agrupa las funcionalidades que brindan las clases que lo componen.

Para la realización del diagrama de paquetes y subsistemas de diseño se separó según el estilo arquitectónico a utilizar todos los paquetes que estarán presentes en cada uno de los paquetes principales que son: Modelo, Vista y Controlador. En cada uno de ellos se especifican los paquetes que contienen las clases que estarán presentes, así como las relaciones que existen entre ellas.

El paquete **MODELO** contiene todo lo relacionado con el módulo componentes de simulación, lo relacionado con las excepciones y el tratamiento de errores y la clase encargada de la manipulación de las operaciones matemáticas llamada Solver y además una clase para el análisis de los resultados.

En el paquete **VISTA** se pueden encontrar lo relacionado con el módulo de editor gráfico, el análisis de los resultados que arroje la simulación y la gestión de reportes sobre los resultados. Además se puede encontrar un paquete llamado Principal que contiene la

primera visión del simulador, o sea lo primero con que se va a encontrar el usuario una vez ejecute la aplicación.

En el **CONTROLADOR** se encontrarán las clases controladoras CC_Reportes, CC_Analisis_Resultados y CC_Simulación, que serán las encargadas de llevar el control sobre los eventos del sistema y su gestión y la lógica de la aplicación.

Cada una de las operaciones que se realizan dentro de estos paquetes y cuál es su función en el simulador se explicará con detalles más adelante en dicho documento.

Los paquetes antes mencionados que forman parte del sistema que se quiere desarrollar se presenta a continuación en la figura 5:

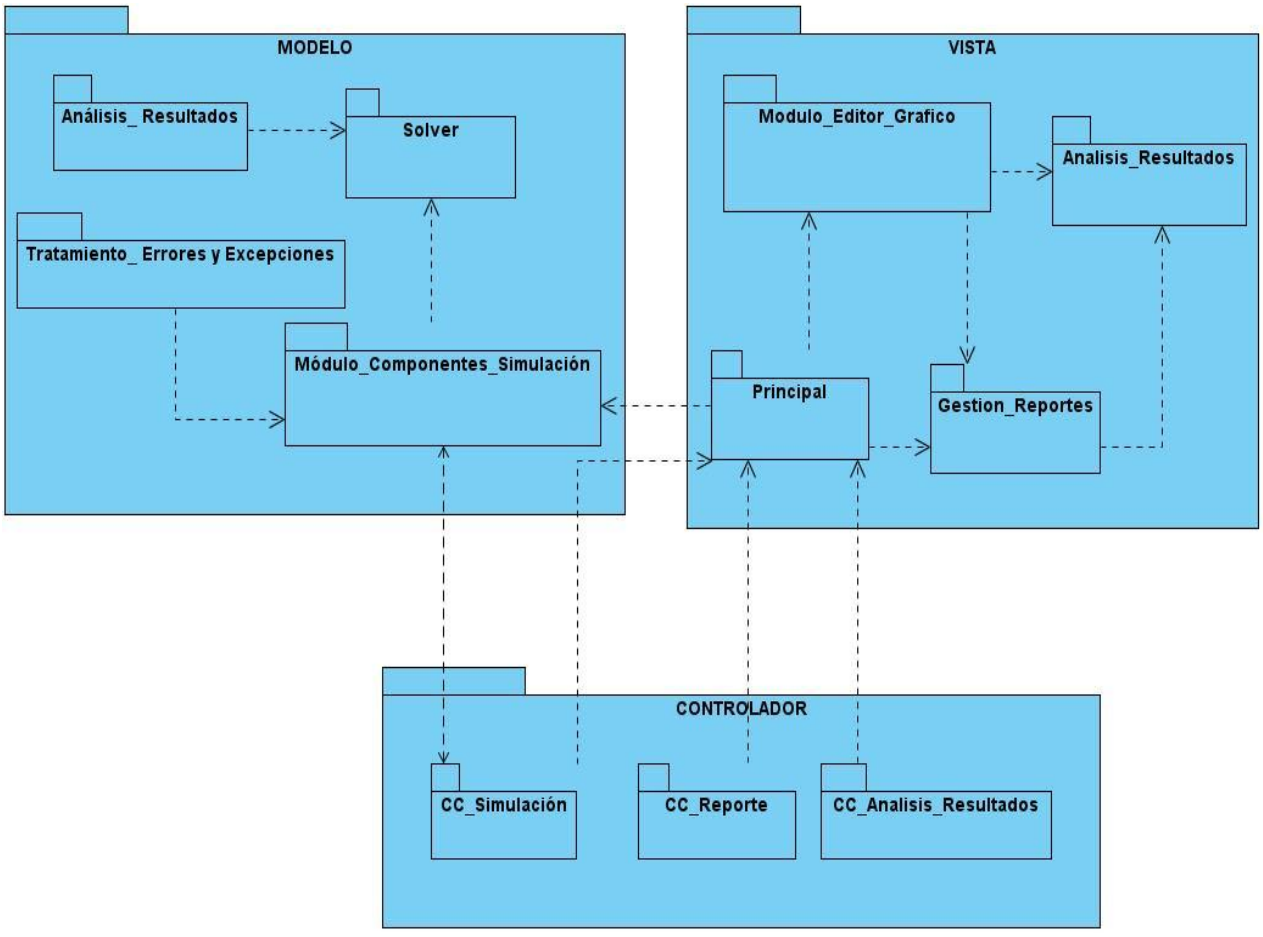

Figura 5 Diagrama de paquetes y subsistemas de diseño

Tratamiento de errores y excepciones: Este paquete contiene las clases que permiten manejar las excepciones que pueden lanzarse a medida que se realiza el proceso de simulación debido a los errores que puedan ocurrir.

Solver: En este paquete se van a almacenar las clases que contienen operaciones matemáticas complejas y que van a ser utilizadas en el proceso de simulación.

Módulo Componentes de Simulación: 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 esté simulando. Este componente provee la estructura general de un módulo y sus puertos. Dentro de este paquete se encuentran subsistemas como Operaciones Unitarias, Corrientes y Componentes que se encargan de las funcionalidades y el cálculo de las propiedades físico químicas asociadas.

Principal: 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.

Módulo Editor Gráfico: Contiene el conjunto de funcionalidades que permite proporcionar elementos claves como los módulos, líneas, puertos, el panel donde se dibujan los DFI, los formularios que van a permitir la entrada de la información por parte del usuario y los elementos que permiten la visualización de la información contenida en los equipos y las corrientes. Además se definen los mecanismos para el manejo de los módulos como son las operaciones de cortar, pegar, eliminar, insertar, deshacer/rehacer, rotar en diferentes direcciones, etc. Todas las funcionalidades que forman parte del editor gráfico están encapsuladas en este paquete, así se lograría que si se desea cambiar algo en la interfaz del simulador esto no afectaría lo que se encuentra encapsulado en el módulo de componentes de simulación que provee los mecanismos para el cálculo de propiedades de las corrientes y las operaciones unitarias. La ventaja que brinda es que cuando se desee cambiar cualquier función en la interfaz gráfica solo hay que sustituir la implementación de este paquete.

Análisis de Resultados: Como parte del proceso de simulación y como objetivo fundamental del mismo, el análisis de los resultados obtenidos es fundamental en el proceso. Esto con el objetivo de interpretar correctamente los datos que se obtienen en las simulaciones que se realizan pues estos son necesarios en momentos críticos como la toma de decisiones. Además se puede obtener información para complementar la investigación,

planeación y diseño de un sistema o proceso y así evitar errores que puedan ser costosos en el futuro.

Gestión de Reportes: El proceso de simulación genera gran cantidad de información, para lograr una mejor organización de dicha información se crea este paquete con el objetivo de brindar al usuario un reporte organizado y de mayor comprensión sobre los resultados obtenidos.

Realización de los principales casos de uso

Para la realización de los principales casos de uso se verán a continuación los diagramas correspondientes a las clases del diseño y los de interacción, específicamente los de secuencia.

Los diagramas de clases del diseño exponen las clases que intervienen en las realizaciones de los casos de uso del sistema. En este tipo de diagrama se representa un nivel de detalle más alto que los diagramas de clases del análisis, relacionándose con el lenguaje de programación del cual se hará uso en la implementación del sistema. El diagrama de clases de diseño describe gráficamente las especificaciones de las clases de software y de las interfaces en una aplicación. Normalmente contiene la siguiente información:

- Clases, asociaciones y atributos.
- Interfaces, con sus operaciones y constantes.
- Métodos.
- Información sobre los tipos de los atributos, navegabilidad y dependencias. (H.ASTUDILLO, 2003)

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

Se dividen en dos categorías: los diagramas de colaboración y los diagramas de secuencia. Un diagrama de secuencia es un diagrama de interacción que destaca el orden temporal de los mensajes. Muestran las interacciones expresadas en función de secuencias temporales. Un diagrama de colaboración es un diagrama de interacción que destaca la organización

estructural de los objetos que envían y reciben mensajes. Muestran las relaciones entre los objetos y los mensajes que intercambian. (Pressman, 2002)

Como ya se ha mencionado en una primera iteración del proyecto: Plataforma para el desarrollo de simuladores de procesos químicos se van a implementar dos módulos que conforman la misma: Módulo de Componentes de Simulación y Editor Gráfico.

Se presentará también la realización de algunos casos de uso, mostrando su diagrama de clases del diseño y de secuencia, para un mejor entendimiento de los mismos. Entre ellos se encuentran el caso de uso Simular y Reportar_Resultados.

A continuación se presenta en la figura 6 y 7 la estructuración de cómo quedarían los diagramas de clases del diseño para los módulos que serán implementados.

Para el módulo Componentes de simulación:

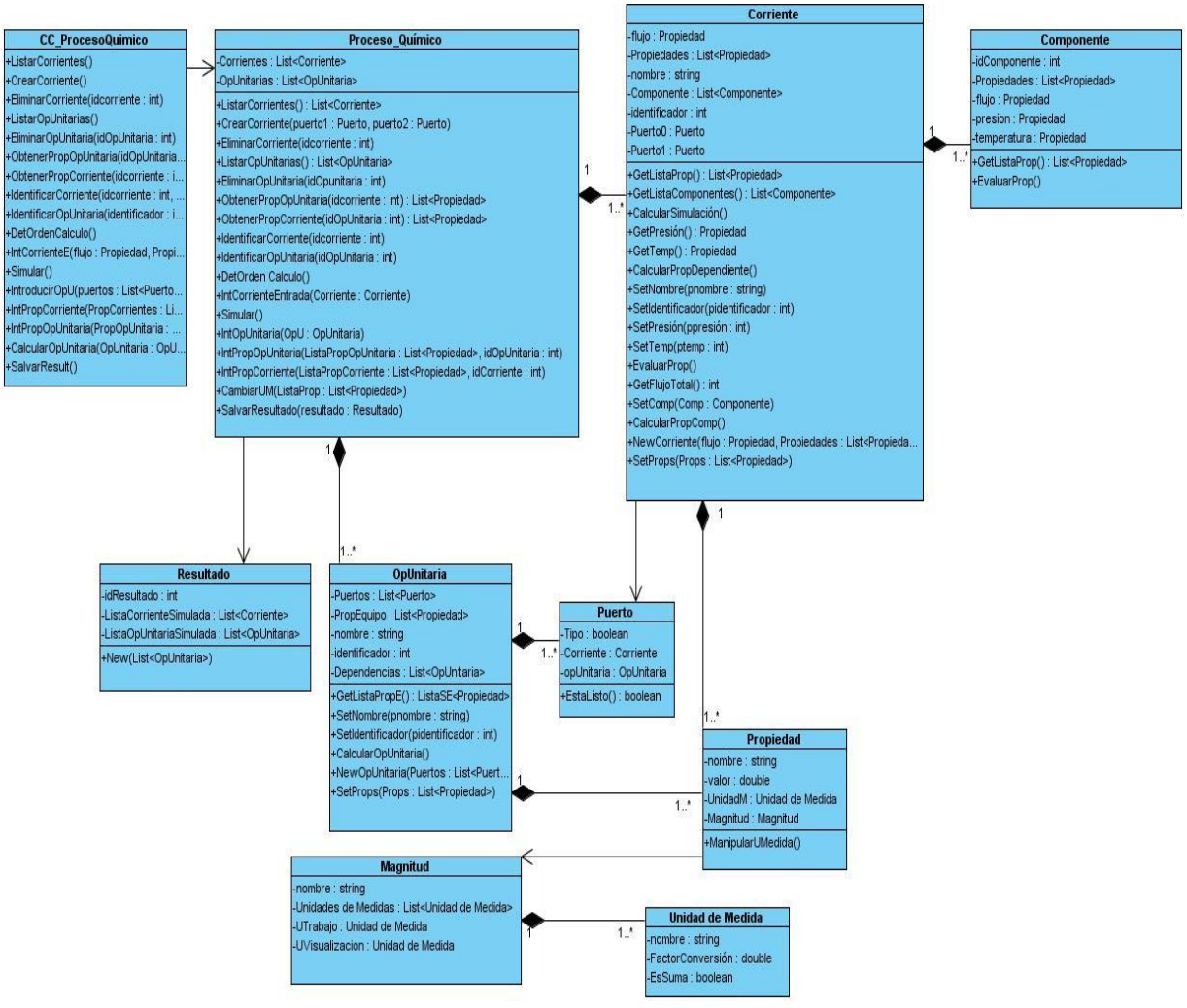

Figura 6 Diagrama de clases del diseño: Módulo Componentes de Simulación

Para el módulo Editor Gráfico:

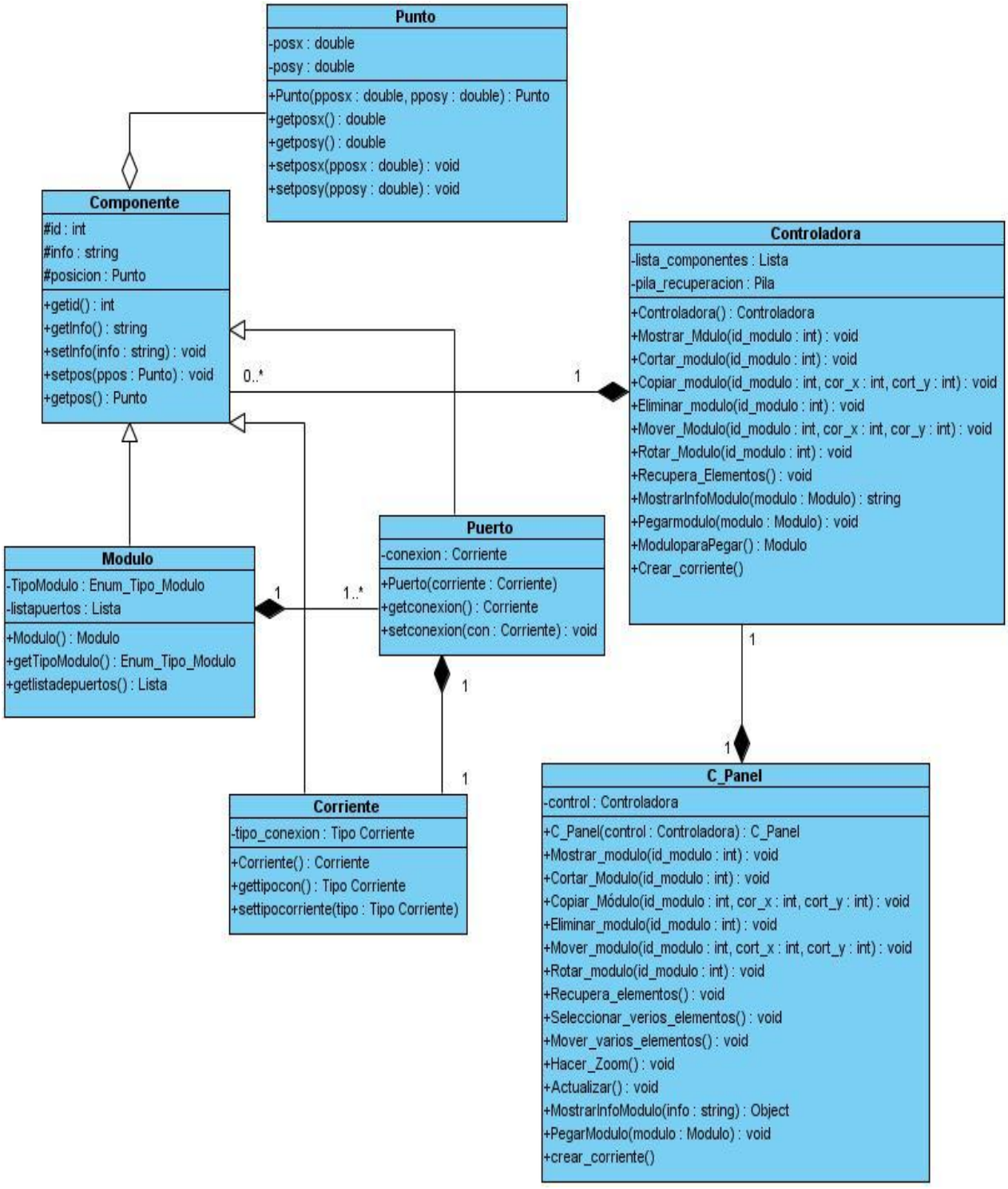

Figura 7 Diagrama de clases del diseño: módulo Editor Gráfico

Realización Caso de Uso Simular:

En la figura 8 se muestra la estructuración de las clases que conforman el caso de uso simular en el diagrama de clases de diseño.

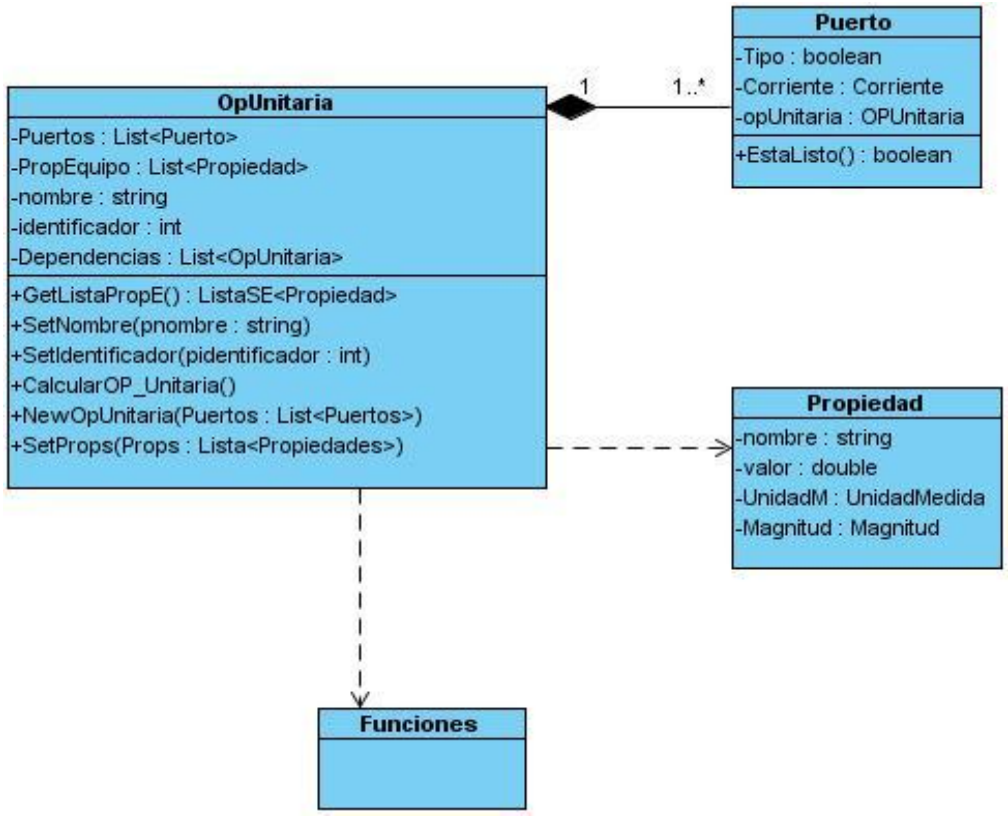

Figura 8 Diagrama de clases del diseño Caso de Uso Simular

La clase **OpUnitaria** contiene los modelos matemáticos de los equipos presentes para realizar la simulación. Esta clase presenta varios métodos y propiedades que contribuyen al buen desempeño de la simulación.

La clase **Puerto** es la encargada de conectar las corrientes con las operaciones unitarias.

La clase **Propiedad** tiene las propiedades valor, que se refiere al valor numérico de los cálculos, Magnitud que dice la magnitud asociada a ese valor y UnidadM la unidad de medida específica.

La clase **Funciones** contiene todas las funciones implementadas que son comunes a las operaciones unitarias, y que se utilizan en el cálculo de propiedades químicas.

Para mostrar el flujo de eventos que se desarrollan a partir de las clases del diseño para el caso de uso Simular se puede ver en la figura 9 el diagrama de secuencia correspondiente.

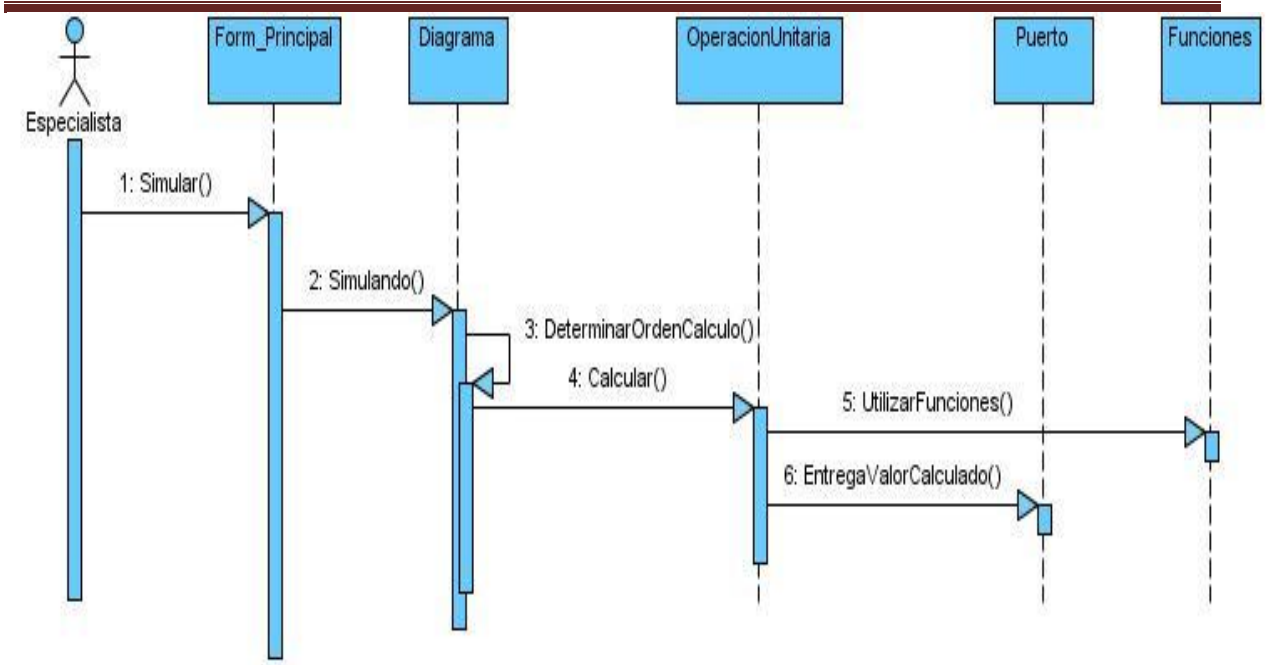

Figura 9 Diagrama de secuencia Caso de Uso Simular

El usuario solicita a la aplicación en el formulario principal que desea simular un diagrama determinado, la interfaz solicita al Diagrama que comience el proceso de simulación. El diagrama ordena los módulos en un orden de cálculo lógico y llama a cada uno de los métodos calcular de las operaciones unitarias, por último entrega el valor calculado.

Realización Caso de Uso Reportar_Resultados:

En la figura 10 se muestran las clases del diseño que conforman el caso de uso Reportar_Resultados.

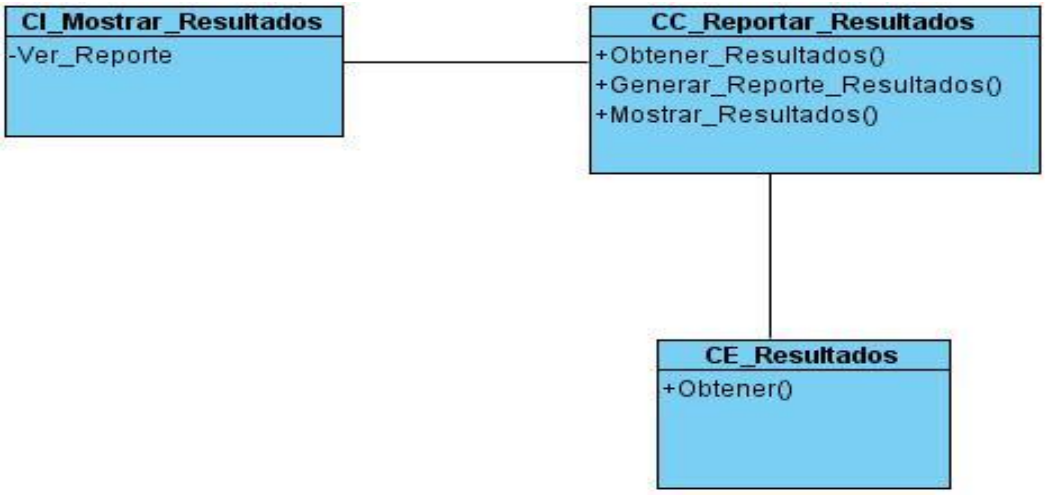

Figura 10 Diagramas de clases del diseño Caso de Uso Reportar Resultados

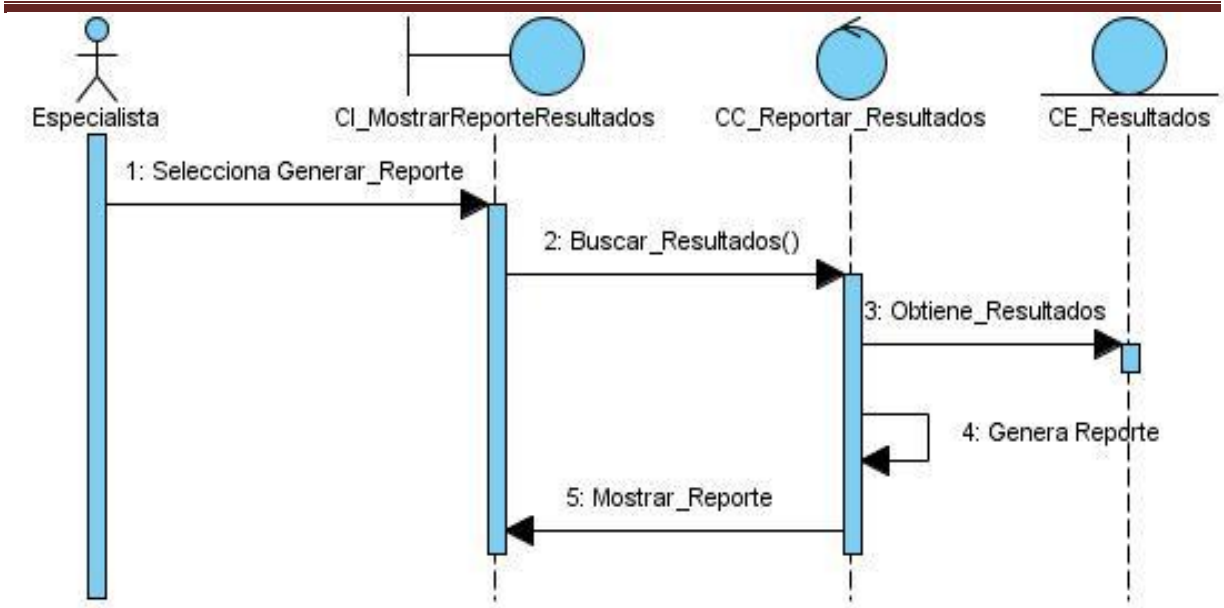

Figura 11 Diagrama de Secuencia Caso de Uso Reportar Resultados

El usuario para acceder a los resultados de la simulación realizada selecciona generar reporte de los resultados que se obtienen después de haber realizado un análisis de los mismos. La interfaz solicita los resultados y luego se obtienen los mismos, a partir de aquí se genera el reporte solicitado y luego se muestran al usuario.

2.4.3 Vista de implementación

Un componente de software es una parte física de un sistema como puede ser un módulo, una base de datos, un programa ejecutable, etc. Las clases son conceptos de abstracción que poseen atributos y métodos que se implementan o materializan en los componentes. A continuación se presenta en la figura 12 el diagrama de componentes correspondiente a la vista de implementación del sistema a desarrollar. Además ilustra la organización de los componentes y las dependencias que se establecen entre los mismos.

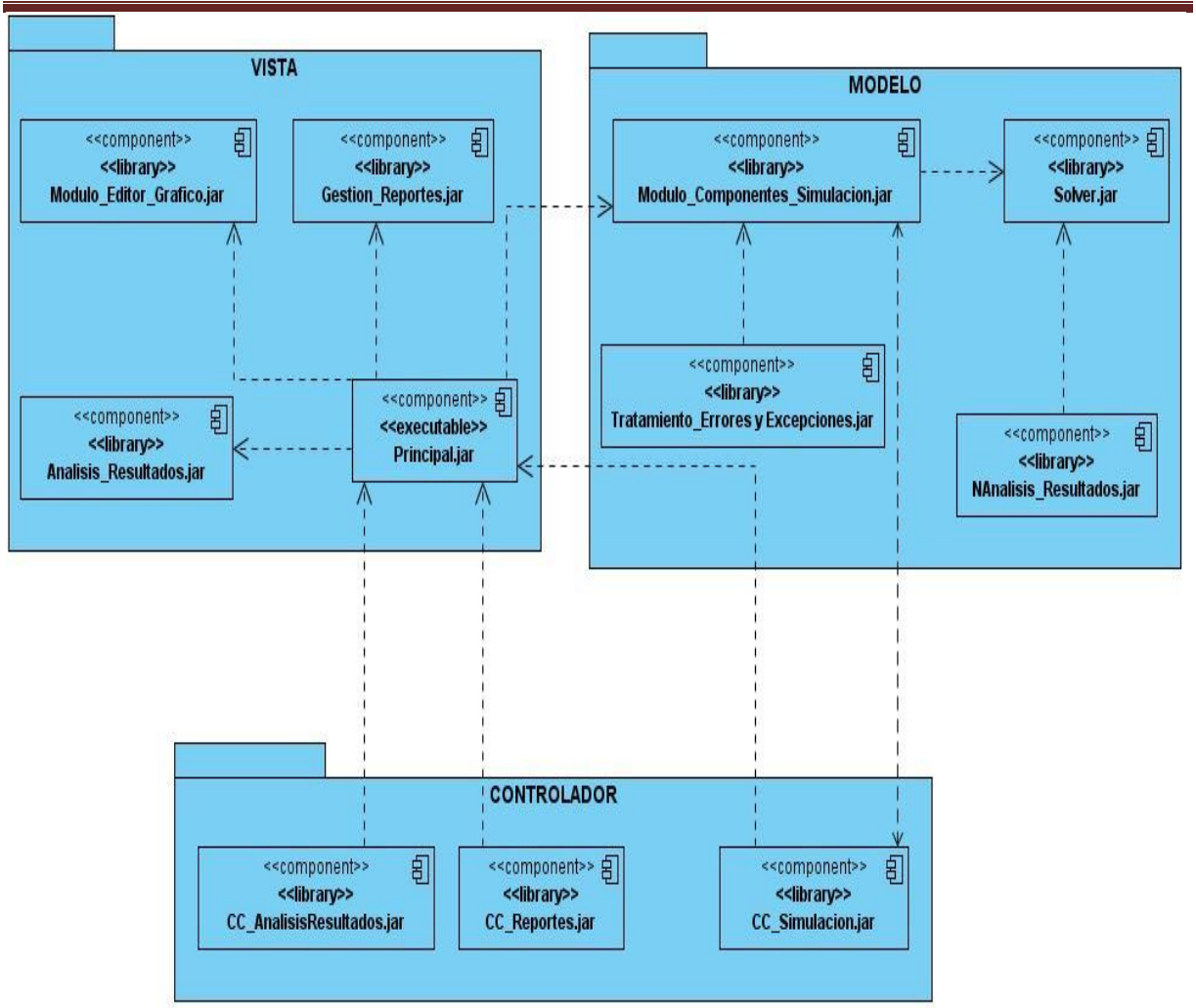

Figura 12 Diagrama de componentes

2.4.4 Vista de Despliegue

La vista de despliegue permite ver el sistema en términos de nodos de procesamiento, servidores o dispositivos. Muestra la comunicación entre los diferentes nodos que componen los escenarios de distribución física del sistema. Los diagramas de despliegue muestran la configuración en funcionamiento del sistema, incluyendo su hardware y su software.

En el caso de la plataforma que se quiere implementar se tendrá en cuenta que debido a que se desarrollará como una aplicación de escritorio, en la cual los componentes que la conforman corren sobre la misma computadora, estará presente en el diagrama de despliegue una computadora cliente donde estará instalado el software y una impresora como dispositivo la cual se utilizará para imprimir reportes o imágenes exportadas desde el sistema, lo cual se muestra en la figura 13.

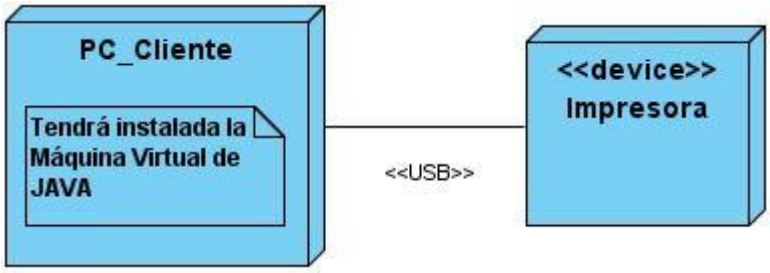

Figura 13 Diagrama de Despliegue

2.5 Uso de Patrones

2.5.1 Patrones de diseño utilizados

El diseño es un modelo del sistema que permite la descripción del mismo con el suficiente detalle para ser implementado. Pero esto muchas veces no es suficiente, por lo que se adopta la utilización de esquemas y estructuras de solución en función del contexto del sistema. Es aquí donde hacen su aparición los patrones de diseño como descripciones de clases cuyas instancias colaboran entre sí. Cada patrón se adecúa para ser adaptado a cierto tipo de problema.

Algunos de los patrones que se utilizaron por las características de la aplicación son los que se muestran a continuación, los cuales aparecen con frecuencia en alguno de los subsistemas:

Fachada: Para disminuir la dependencia entre subsistemas de diseño y garantizar puntos de acceso común a dichos componentes.

Instancia única (Singleton): Este patrón se utiliza para garantizar que exista una instancia de una clase en todo el sistema. Un ejemplo en el cual se pone de manifiesto es en la clase unidades de medida, la cual se crea para convertir las unidades de medida que se utilizan en el proceso las cuales son comunes para todo.

Observador: En la aplicación es utilizado para actualizar determinados elementos que dependen directamente de los datos de otros, específicamente para actualizar los datos de la interfaz gráfica ante un cambio en los mismos. O sea el formulario principal de la aplicación contiene varios componentes, a los cuales se les deberá notificar cualquier cambio significativo que ocurra en los diagramas que se realicen y actualice los datos correspondientes.

Comando: Este patrón se encarga de encapsular una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma. Esto se puede ver a la hora de realizar operaciones como deshacer o rehacer en el editor gráfico.

Memento: Dentro de una aplicación a desarrollar existe la necesidad de en algún momento salvar los datos que posee determinado objeto con el fin de poder restaurarlo a ese estado luego de una modificación no deseada. El patrón memento posee como propósito fundamental el cumplimiento de esta tarea sin violar además la encapsulación de los datos de dicho objeto.

2.5.2 Patrones GRASP utilizados

Experto: Se evidencia en los diagramas realizados, sobre todo en las clases controladoras de los distintos análisis, puesto que estas poseen la mayor información necesaria para resolver las funcionalidades requeridas.

Bajo acoplamiento: Se pone de manifiesto una vez que las relaciones entre las clases que participan en el desarrollo de los distintos casos de usos son mínimas asegurando así que exista entre ellas una alta cohesión.

Creador: Este patrón es utilizado con el objetivo de asignar la responsabilidad de crear instancias de otras clases a la clase adecuada para este fin.

2.6 Estándares de codificación.

Un estándar de codificación son reglas que se siguen para la escritura del código fuente de tal manera que otros programadores puedan entender con mayor facilidad el código escrito por otros, algunos ejemplos son: identificar las variables, las funciones o métodos.

En particular el lenguaje de programación Java tiene algunas reglas como: Las clases inician en mayúsculas, los atributos y métodos inician con minúsculas, las constantes son todas en mayúsculas. También menciona la forma de abrir y cerrar bloques de código y la forma de poner comentarios.

Establecido por la Sun Microsystems para este lenguaje se destacan algunos estándares que contribuyen a las buenas prácticas, los cuales han servido de base para su utilización en el proyecto en cuestión. En el documento Convenciones Código Java se establece que:

Las convenciones de código son importantes para los programadores por un gran número de razones:

- El 80% del coste del código de un programa va a su mantenimiento.
- Casi ningún software lo mantiene toda su vida el auto original.
- Las convenciones de código mejoran la lectura del software, permitiendo entender código nuevo mucho más rápidamente y más a fondo.
- Si distribuyes tu código fuente como un producto, necesitas asegurarte de que está bien hecho y presentado como cualquier otro producto.

Para que funcionen las convenciones, cada persona que escribe software debe seguir la convención. Todos. (Asociación JavaHispano 2002-2007)

Para el conocimiento de estos estándares dirigirse al documento mencionado anteriormente disponible en la bibliografía especificada.

Conclusiones Parciales

En este capítulo se ha abordado específicamente todo lo relacionado con la realización de la propuesta de arquitectura para la plataforma de simuladores de procesos químicos.

Se especificaron las vistas de la arquitectura de acuerdo a la situación del proyecto y las características que presenta el sistema (Vista de CU, Lógica, Despliegue e Implementación).

Han quedado definidos los patrones que contribuirán en el desarrollo del sistema así como la utilización del estilo MVC definido para establecer la línea base para la propuesta de la arquitectura.

Capítulo 3: Evaluación de la arquitectura

3.1 Introducción.

En este capítulo se abordará lo referente a la evaluación de la arquitectura propuesta como solución a la problemática existente en la investigación. Se podrán encontrar algunas de las técnicas utilizadas para evaluar la arquitectura. Se describirán brevemente algunos de los métodos de evaluación de arquitectura, fundamentalmente se abordarán características y pasos de aplicación de los mismos. Finalmente se estimará el comportamiento del subsistema a desarrollar según los atributos de calidad que el mismo debe poseer.

3.2 Objetivos de evaluar la arquitectura.

El objetivo de evaluar una arquitectura es saber si puede habilitar los requerimientos, atributos de calidad y restricciones para asegurar que el sistema a ser construido cumple con las necesidades de los stakeholders. (Gustavo Andres Brey, 2005).

La evaluación de una arquitectura de software es una tarea no trivial, puesto que se pretende medir propiedades del sistema en base a especificaciones abstractas, como por ejemplo los diseños arquitectónicos. Por ello, la intención es más bien la evaluación del potencial de la arquitectura diseñada para alcanzar los atributos de calidad requeridos. Las mediciones que se realizan sobre una arquitectura de software pueden tener distintos objetivos, dependiendo de la situación en la que se encuentre el arquitecto y la aplicabilidad de las técnicas que emplea. Algunos de estos objetivos son: cualitativos, cuantitativos y máximos y mínimos teóricos. (Erika Camacho, 2004).

Para dar cumplimiento a estos objetivos se pueden aplicar varias técnicas y métodos que de una forma u otra pretenden obtener un mismo final y es lograr que la arquitectura planteada para un determinado proyecto cumpla con todas las normas establecidas y que su aplicación pueda llevar al éxito. Además que cumpla con los atributos de calidad que se proponen para comprobar su correcto funcionamiento.

3.3 Necesidad de evaluar la Arquitectura.

¿Por qué evaluar una Arquitectura?

“El propósito de realizar evaluaciones a la arquitectura, es para analizar e identificar riesgos potenciales en su estructura y sus propiedades, que puedan afectar al sistema de software resultante, verificar que los requerimientos no funcionales estén presentes en la arquitectura, así como determinar en qué grado se satisfacen los atributos de calidad. Cabe

señalar que los requerimientos no funcionales también son llamados atributos de calidad". (Gómez, 2007).

Un atributo de calidad es una característica de calidad que afecta a un elemento. Donde el término "característica" se refiere a aspectos no funcionales y el término "elemento" a componente. (Gómez, 2007).

Las evaluaciones de la arquitectura tienden a aumentar la calidad, el control de gastos, presupuesto y además garantizan la detección de riesgos. La arquitectura es el marco para todas las decisiones técnicas y tiene un importante impacto en el costo de los productos y la calidad. Una evaluación de la arquitectura no garantiza la alta calidad o el bajo costo de desarrollar los productos, pero puede señalar áreas de riesgo que de ser tratadas garantizan una mayor calidad del producto final disminuyendo además el costo del desarrollo del mismo.

Evaluar la arquitectura de software de un sistema permite conocer el grado en que esta satisface lo que el cliente final espera del producto. La evaluación de una arquitectura no arroja en muchos casos una respuesta específica de si es buena la arquitectura o no en cuanto a los atributos comprobados, sino que identifica las deficiencias y fortalezas de la misma. Según los resultados de la evaluación se determinan qué decisiones aplicar al proyecto que se desarrolla y de esta manera obtener un producto final sin fallas.

Mediante la evaluación de la arquitectura se pueden mitigar los diferentes riesgos asociados con el desarrollo del software, cuanto antes se detecten los errores en las decisiones sobre el sistema, menos costosos serán los mismos en términos económicos y de tiempo requerido para solucionar dichos defectos, mejorar la visión de los procesos críticos y validar las decisiones de diseño que se tomaron, tomar acciones tempranas y valorar los atributos no funcionales sin esperar a que el software se construya. Todo esto llevaría a la satisfacción final del cliente y que el producto cumpla con todas las metas propuestas para un mejor desarrollo del mismo.

En la tabla 8 que se encuentra a continuación se pueden ver algunos de los atributos de calidad por los cuales pueden ser evaluadas las arquitecturas, así como una breve descripción de dichos atributos.

3.4 Atributos por los cuales puede ser evaluada una arquitectura

Atributo de Calidad	Descripción

Disponibilidad	Es la medida de disponibilidad del sistema para el uso (Barbacci et al, 1995).
Confidencialidad	Es la ausencia de acceso no autorizado a la información (Barbacci et al, 1995).
Funcionalidad	Habilidad del sistema para realizar el trabajo para el cual fue concebido (R Kazman, 2001)
Desempeño	Es el grado en el cual un sistema o componente cumple con sus funciones designadas, dentro de ciertas restricciones dadas, como velocidad, exactitud o uso de memoria. (IEEE 610.12).
Confiabilidad	Es la medida de la habilidad de un sistema a mantenerse operativo a lo largo del tiempo (Barbacci et al., 1995).
Seguridad externa	Ausencia de consecuencias catastróficas en el ambiente. Es la medida de ausencia de errores que generan pérdidas de información (Barbacci et al, 1995).
Seguridad interna	Es la medida de la habilidad del sistema para resistir a intentos de uso no autorizados y negación del servicio, mientras se sirve a usuarios legítimos (R Kazman, 2001).
Configurabilidad	Posibilidad que se otorga a un usuario experto a realizar ciertos cambios al sistema (Bosch et al., 1999).
Integrabilidad	Es la medida en que trabajan correctamente componentes del sistema que fueron desarrollados separadamente al ser integrados. (Bass et al. 1998)
Integridad	Es la ausencia de alteraciones inapropiadas de la información (Barbacci et al., 1995).

Interoperabilidad	Es la medida de la habilidad de que un grupo de partes del sistema trabajen con otro sistema. Es un tipo especial de *integrabilidad* (Bass et al. 1998)
Modificabilidad	Es la habilidad de realizar cambios futuros al sistema. (Bosch et al. 1999).
Mantenibilidad	Es la capacidad de someter a un sistema a reparaciones y evolución. (Barbacci et al., 1995). Capacidad de modificar el sistema de manera rápida y a bajo costo (Bosch et al. 1999).
Portabilidad	Es la habilidad del sistema para ser ejecutado en diferentes ambientes de computación. Estos ambientes pueden ser hardware, software o una combinación de los dos (R Kazman, 2001).
Reusabilidad	Es la capacidad de diseñar un sistema de forma tal que su estructura o parte de sus componentes puedan ser reutilizados en futuras aplicaciones (Bass et al. 1998).
Escalabilidad	Es el grado con el que se pueden ampliar el diseño arquitectónico, de datos o procedimental (Pressman, 2002).

Tabla 8 Atributos de calidad.

3.5 Técnicas de evaluación de Arquitectura.

Existen diferentes técnicas de evaluar la arquitectura, clasificadas en cualitativas o cuantitativas, por ejemplo: En las técnicas de evaluación cualitativas se pueden utilizar: escenarios, cuestionarios o listas de verificación. Por otro lado, en las técnicas de evaluación cuantitativas se pueden emplear: métricas, simulaciones, prototipos, experimentos o modelos matemáticos. La mayoría de los métodos de evaluación utilizan escenarios, que son secuencias específicas de pasos que involucran el uso o la modificación del sistema. Por lo regular, las técnicas de evaluación cualitativas son usadas cuando la arquitectura se encuentra en construcción, mientras que las técnicas de evaluación cuantitativas, se usan cuando la arquitectura ya ha sido implantada.

Las técnicas cuantitativas se aplican ya implementada la arquitectura. Estas están encaminadas a obtener valores para la toma de decisiones en cuanto a los atributos de calidad. En vista de que el interés es tomar decisiones de tipo arquitectónico en las fases tempranas del desarrollo, son necesarias técnicas que requieran poca información detallada y puedan conducir a resultados relativamente precisos. Las técnicas que posibilitan lo antes planteado son las cualitativas. Las técnicas de evaluación más empleadas por los arquitectos son estas debido al costo que implica realizar una evaluación cuantitativa, el cual es muy elevado en comparación con el relativo bajo costo de evaluar la arquitectura utilizando técnicas cualitativas. (E.M.d Armas, 2008)

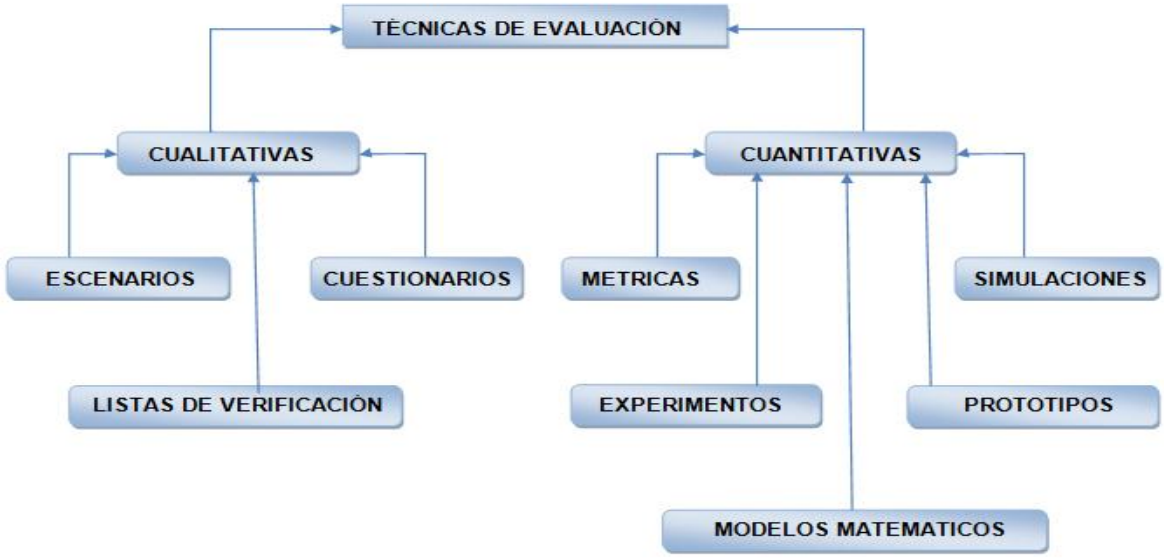

Figura 14 Clasificación de las técnicas de evaluación. (Gómez, 2007)

Para la correcta aplicación de una de las técnicas de evaluación de la arquitectura, se verá a continuación una descripción de algunas de las que pudieran utilizarse en la arquitectura propuesta. Para ello se verán sus características y en algunos casos los pasos para llevarla a cabo.

3.5.1 Evaluación basada en escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema.

Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. El contexto describe qué sucede en el sistema al momento del

estímulo. La respuesta describe a través de la arquitectura, cómo debería responder el sistema ante el estímulo. Este último elemento es el que va a permitir establecer cuál es el atributo de calidad asociado. Los escenarios proveen una forma para concretar y entender los atributos de calidad.

Entre las ventajas de su uso se pueden encontrar:

- Son simples de crear y entender.
- Son poco costosos y no requieren de mucho esfuerzo.
- Son efectivos.

Las técnicas basadas en escenarios cuentan con dos instrumentos de evaluación relevantes: el Utility Tree y Profile.

Utility Tree: Es un esquema en forma de árbol que presenta los atributos de calidad de un sistema de software, refinados hasta el establecimiento de escenarios que especifican con suficiente detalle el nivel de prioridad de cada uno. (R Kazman, 2001)

Profile: Un perfil es un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de estos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. (R Kazman, 2001)

3.5.2 Evaluación basada en simulación

La evaluación basada en simulación utiliza una implementación de alto nivel de la arquitectura de software. El enfoque básico consiste en la implementación de componentes de la arquitectura y la implementación a cierto nivel de abstracción del contexto del sistema donde se supone va a ejecutarse. La finalidad es evaluar el comportamiento de la arquitectura bajo diversas circunstancias. Una vez disponibles estas implementaciones, pueden usarse los perfiles respectivos para evaluar los atributos de calidad.

El proceso de evaluación basada en simulación sigue los siguientes pasos:

- **Definición e implementación del contexto:** Consiste en identificar las interfaces de la arquitectura de software con su contexto, y decidir cómo será simulado el comportamiento del contexto en tales interfaces. (E. F Camacho Cordeso, 2004)

Implementación de los componentes arquitectónicos: La descripción del diseño arquitectónico debe definir, por lo menos, las interfaces y las conexiones de los

componentes, por lo que estas partes pueden ser tomadas directamente de la descripción de diseño. El comportamiento de los componentes en respuesta a eventos sobre sus interfaces puede no ser especificado claramente, aunque generalmente existe un conocimiento común y es necesario que el arquitecto lo interprete, por lo que éste decide el nivel de detalle de la implementación. (E. F Camacho Cordeso, 2004)

- **Implementación del perfil:** Dependiendo del atributo de calidad que el arquitecto de software intenta evaluar usando simulación, el perfil asociado necesitará ser implementado en el sistema. El arquitecto de software debe ser capaz de activar escenarios individuales, así como también ejecutar un perfil completo usando selección aleatoria, basado en los pesos normalizados de los mismos. (E. F Camacho Cordeso, 2004)

- **Simulación del sistema e inicio del perfil:** El arquitecto de software ejecutará la simulación y activará escenarios de forma manual o automática, y obtendrá resultados de acuerdo al atributo de calidad que está siendo evaluado. (E. F Camacho Cordeso, 2004)

- **Predicción de atributos de calidad:** Dependiendo del tipo de simulación y del atributo de calidad evaluado, se puede disponer de cantidades excesivas de datos, que requieren ser condensados. Esto permite hacer conclusiones acerca del comportamiento del sistema. (E. F Camacho Cordeso, 2004)

La evaluación basada en simulación está ligada al contexto de evaluación a partir de un prototipo funcional de la aplicación o el sistema a desarrollar donde se activarán los escenarios y se evaluarán en dependencia de los resultados los atributos de calidad esperados bajo ciertas condiciones impuestas básicamente para probar el desempeño de la aplicación.

3.5.3 Evaluación basada en modelos matemáticos

Establece que la evaluación basada en modelos matemáticos se utiliza para evaluar atributos de calidad operacionales. Permite una evaluación estática de los modelos de diseño arquitectónico, y se presentan como alternativa a la simulación, dado que evalúan el mismo tipo de atributos. Ambos enfoques pueden ser combinados, utilizando los resultados de uno como entrada para el otro.

Pasos para la utilización de la evaluación de la arquitectura a partir de modelos matemáticos:

- **Selección y adaptación del modelo matemático.** La mayoría de los centros de investigación orientados a atributos de calidad han desarrollado modelos matemáticos para

medir sus atributos de calidad, los cuales tienden a ser muy elaborados y detallados, así como también requieren de cierto tipo de datos y análisis. Parte de estos datos requeridos no están disponibles a nivel de arquitectura, y la técnica requiere mucho esfuerzo para la evaluación, por lo que el arquitecto de software se ve obligado a adaptar el modelo.

- **Representación de la arquitectura en términos del modelo:** El modelo matemático seleccionado y adaptado no asume necesariamente que el sistema que intenta modelar consiste en componentes y conexiones. Por lo tanto, la arquitectura necesita ser representada en términos del modelo.

- **Estimación de los datos de entrada requeridos:** El modelo matemático aún cuando ha sido adaptado, requiere datos de entrada que no están incluidos en la definición básica de la arquitectura. Es necesario estimar y deducir estos datos de la especificación de requerimientos y de la arquitectura diseñada.

- **Predicción de atributos de calidad:** Una vez que la arquitectura es expresada en términos del modelo y se encuentran disponibles todos los datos de entrada requeridos, el arquitecto está en capacidad de calcular la predicción resultante del atributo de calidad evaluado.

Entre las desventajas que presenta esta técnica se encuentra la inexistencia de modelos matemáticos apropiados para los atributos de calidad relevantes y el hecho de que el desarrollo de un modelo de simulación completo puede requerir esfuerzos sustanciales.

Selección de la técnica de evaluación a utilizar.

Para la evaluación de la arquitectura propuesta se plantea utilizar la técnica basada en escenarios, pues los escenarios permiten de una manera más fácil concretar y entender los atributos de calidad. Además la técnica es poco costosa y no requiere de mucho entendimiento pues es muy fácil de aplicar y efectiva. Para la evaluación por esta técnica se utilizará como instrumento; por perfiles, pues permite detallar con más precisión el requerimiento para un determinado atributo de calidad.

3.6 Métodos de evaluación de Arquitectura.

Un método de evaluación sirve de guía a los involucrados en el desarrollo del sistema para la búsqueda de conflictos que puede presentar una arquitectura y sus soluciones. (R Kazman, 2001).

Los métodos de evaluación de arquitectura están dados por una serie de actividades que se deben desarrollar por varios roles o implicados para evaluar el desempeño en cuanto a atributos de calidad de la arquitectura desarrollada. Algunos de los métodos empleados para este tipo de evaluación son los siguientes:

- Software Architecture Analysis Method (SAAM) con tres perspectivas para el análisis de la arquitectura y cinco pasos para el análisis; este método está basado en escenarios y permite evaluar una arquitectura o comparar varias. Es más usado cuando el requerimiento de calidad: modificabilidad es el de mayor interés.

- Architecture Trade off Analysis Method (ATAM) el cual está 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 cumplimiento de dichas metas. No solo dice cuán bien satisface las metas de calidad, sino que provee ideas de cómo esas metas de calidad interactúan entre ellas. Es más profundo para evaluar aspectos como rendimiento y confiabilidad.

- An Evaluation Method for Partial Architectures (ARID): Este método es un híbrido de ADR (Revisiones de diseño activas) y ATAM que tienen características útiles para la evaluación de diseños preliminares, por lo que es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo y usarlo para evaluar la factibilidad de la arquitectura. (R Kazman, 2001).

Para realizar una correcta selección del método a utilizar para evaluar la arquitectura propuesta se puede encontrar a continuación en la tabla 9 una comparación de dichos métodos, lo que ayudará a llegar a una conclusión viable para la misma.

3.6.1 Comparación entre los métodos de evaluación de la arquitectura.

	ATAM	SAAM	ARID

Atributos de Calidad contemplados	No está orientado a ningún atributo en específico, pero se hace énfasis en: Modificabilidad Seguridad Confiabilidad Desempeño	Principalmente: Modificabilidad Funcionalidad	Conveniencia del diseño evaluado
Objetos analizados	Estilos arquitectónicos Documentación Flujo de datos Vistas Arquitectónicas	Documentación Vistas Arquitectónicas	Especificación de los componentes
Etapas del proyecto en las que se aplica.	Luego de que el diseño de la arquitectura ha sido establecido.	Luego de que la arquitectura cuenta con funcionalidades implementadas.	A lo largo del diseño de la arquitectura.
Enfoques utilizados	Árbol de utilidades y lluvia de ideas para articular los requisitos de calidad. Análisis arquitectónico que detecta puntos sensibles, puntos de balance y riesgos.	Lluvia de ideas para escenarios y articular los requerimientos de calidad. Análisis de los escenarios para Verificar funcionalidad o estimar el costo de los cambios.	Revisiones de diseños, lluvia de ideas para obtener escenarios.

Tabla 9 Comparación entre los diferentes métodos de evaluación de la arquitectura.

Como se ha podido analizar en la comparación se puede observar que un método de evaluación no es mejor que otro, sino que evalúa mejor, en ciertas condiciones, un atributo de calidad dado. Por lo que se puede concluir que en dependencia de las condiciones y lo que se desea evaluar será la selección del método de evaluación empleado.

El desarrollo de la plataforma de simuladores químicos se encuentra en una etapa prematura, por lo que el método que se utilizará es el ARID, el cual es más conveniente para realizar la evaluación de diseños parciales en etapas tempranas del desarrollo de un producto. Este método se basa en escenarios, es fácil de usar, aplicarlo es poco costoso, de gran beneficio y como se plantea evalúa mejor la factibilidad de la arquitectura.

3.7 Evaluación de la arquitectura propuesta.

Dado el estado actual de la arquitectura y aplicando la técnica cualitativa de evaluación por escenarios, específicamente el método ARID, se definen los escenarios a través de los cuales se evaluará la arquitectura propuesta. Se realizará una breve evaluación de la arquitectura con vista a evaluar los atributos de calidad considerados de mayor importancia según las características del tipo de sistema a desarrollar. Para ello se aplicarán algunos de los pasos que propone el método debido principalmente a que no se cuentan con muchos de los recursos que se necesitan, por lo tanto se define que el arquitecto está calificado para realizar dicha evaluación.

El método ARID cuenta con 9 pasos separados en 2 fases, los cuales se pueden encontrar en la tabla 10, la cual especifica dichos pasos en la fase en que se realiza.

Fases	Pasos
Actividades previas	Identificación de los encargados de la revisión.
	Preparar el informe de diseño.
	Preparar los escenarios base.
	Preparar los materiales.
	Presentación del método ARID.

Evaluación	Presentación del diseño.
	Lluvia de ideas y establecimiento de prioridad de escenarios.
	Realización de la revisión.
	Conclusiones.

Tabla 10 Pasos para la aplicación del método ARID

Se consideró conveniente establecer cuáles atributos de calidad se requieren satisfacer, para esto se definen los escenarios teniendo en cuenta que dichos atributos se basan fundamentalmente en los requerimientos no funcionales del software.

3.7.1 Atributos de calidad

Funcionalidad: La arquitectura planteada tiene que ser funcional, garantizando así que el sistema cumpla con todos los requisitos no funcionales y con todas las expectativas del cliente. Para lograr esto se centra el desarrollo de la misma en realizar un proceso que considere los casos de uso como un elemento de vital importancia, comenzando desde la selección de los casos de uso arquitectónicamente significativos, logrando que el proceso se desarrolle dirigido por casos de uso. A medida que se incrementa el desarrollo del proyecto se implementan nuevos casos de uso, los cuales contribuyen al crecimiento de la arquitectura inicial.

Por otra parte se basa en un proceso iterativo e incremental dando una medida de cómo la arquitectura propone un prototipo del sistema funcional escogiendo las funcionalidades básicas o casos de uso críticos y los prioriza en las primeras iteraciones dentro del ciclo de vida.

Para describir este atributo de calidad se pueden encontrar algunas características como son la adecuación, exactitud, interoperabilidad, seguridad de acceso, las cuales influyen en el desempeño del sistema. La adecuación permite proporcionar un conjunto de funciones para tareas y objetivos de usuarios especificados. En la arquitectura propuesta esto se cumple ya que se tuvieron en cuenta los requisitos funcionales y no funcionales, así como las propuestas de los usuarios. Además esta arquitectura se adecúa al desarrollo de varios simuladores, ya que provee varias funcionalidades que pueden ser utilizadas, como es; realizar DFI, creación de reportes para los resultados, entre otros que dan una medida de la

adecuación de la arquitectura. La exactitud está dada por cuán precisa es la arquitectura en cuanto al resultado final de acuerdo a lo que se está pidiendo que haga por parte del usuario. Para dar cumplimiento a esto se puede observar que los reportes emitidos no pueden contener errores ya que de ellos se toman importantes decisiones por parte de los usuarios del producto, los requisitos se diseñaron en función de la exactitud, quedando demostrado por ejemplo, en los cálculos que debe manejar la aplicación en las conversiones de unidades de medidas.

Con el cumplimiento de estas características en la propuesta de arquitectura se logra que se pueda obtener un sistema funcional que esté a la altura de lo que el cliente desee y que sea lo que tenía en mente desde un principio del ciclo de desarrollo. 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.

Rendimiento: El rendimiento de un sistema está dado por la capacidad de respuesta para ejecutar una acción dentro de un intervalo dado. Para lograr esto se mide el tiempo que tarda el sistema en responder a un evento y como respuesta el número de eventos que tiene lugar en una determinada cantidad de tiempo. En cuanto a rendimiento el lenguaje de programación seleccionado para implementar el proyecto, brinda varias facilidades que se ponen de manifiesto a la hora del análisis de este atributo de calidad con respecto a la aplicación. En primer lugar en tiempo de ejecución, el rendimiento de una aplicación Java depende más de la eficiencia del compilador, o la Máquina Virtual de Java, que de las propiedades específicas del lenguaje. Algunas características del propio lenguaje conllevan una penalización en tiempo, aunque no son únicas de Java, algunas de ellas son el chequeo de los límites de arreglos y chequeo en tiempo de ejecución de tipos. El uso de un recolector de basura para eliminar de forma automática aquellos objetos no requeridos, añade una sobrecarga que puede afectar al rendimiento, o ser apenas apreciable, dependiendo de la tecnología del recolector y de la aplicación en concreto. Las Máquinas Virtuales de Java modernas usan recolectores de basura que gracias a rápidos algoritmos de manejo de memoria, consiguen que algunas aplicaciones puedan ejecutarse más eficientemente.

Para lograr un mejor rendimiento en el software desarrollado se aprovecharon en la medida de lo posible las características de las plataformas virtuales en específico las que ofrece la de Java para favorecer su ejecución utilizando todo el conocimiento que la máquina virtual tiene sobre el comportamiento dinámico de los programas y el conocimiento que el sistema operativo tiene sobre las condiciones de ejecución.

Modificabilidad: El sistema debe ser modificable, permitir agregar nuevas funcionalidades, reutilizar los métodos definidos en las clases controladoras y así lograr que se puedan realizar futuros cambios al sistema para que pueda ser adaptado a nuevas situaciones. El desarrollo del software permite que sea flexible para adecuarse a cambios para extender, cambiar o eliminar funcionalidades del sistema, sin necesidad de volver a escribir los programas y provocando la menor alteración al sistema en su totalidad. Por esta parte el uso del patrón arquitectónico MVC, utilizado en el proyecto, provee una clara separación entre los componentes de un programa, lo cual permite implementarlo por separado y así como se puede implementar por separado si se desea hacer algún cambio futuro es una ventaja que no influya en lo que ya está implementado y que cumple con la funcionalidad requerida, dejándolo como está sin cambiar nada en su implementación. Además el uso de este patrón tiene como ventaja la flexibilidad con que se puede cambiar las vistas y los controladores, otro de los aspectos que muestra la modificabilidad del sistema.

Portabilidad: La portabilidad de un sistema se ha convertido sin dudas en un atributo que posee relevante importancia y es que es la habilidad de un sistema para ser ejecutado en diferentes ambientes de computación. Debido a la situación de Cuba con respecto a la obtención de licencias para poseer los software que se necesitan, por su costo y la situación económica existente en el país, lograr realizar software que puedan ser ejecutados en diferentes ambientes de computación sería un adelanto en el desarrollo de las tecnologías que contribuyen al mejor desempeño del país en el mundo de la computación.

¿Cómo se lograría esto en el software desarrollado? No se puede decir que un software es portable en su totalidad porque siempre va a existir algún factor que deje de cumplirse de los establecidos para que un software sea portable, pero sí se puede decir que un software es altamente portable y se podrá ver a continuación la explicación de dicha afirmación. Primeramente las tecnologías que se utilizan la mayoría son de código libre, por lo que a la hora de tenerlos en una computadora no sería un problema. Por otra parte, siendo el software una aplicación de escritorio, no necesitaría grandes recursos de red. Por último para la instalación del mismo solo se necesitaría tener en la computadora instalada la máquina virtual de JAVA, por lo que no depende tampoco de ningún sistema operativo en específico, siendo multiplataforma se lograría su uso sin tener restricciones que provoquen que no pueda ser usado.

Este atributo de calidad en el software desarrollado presenta una condición y es que para poder instalar el software tiene que estar instalada la máquina virtual de Java, por lo tanto se puede decir que el software es altamente portable porque entre los requerimientos que se plantean se especifica que tiene que estar instalada la misma en la máquina en la que se va

a utilizar. No obstante se tiene en cuenta al evaluar el atributo que tiene que cumplirse dicha condición para así lograr cumplir con el atributo. Con todo lo demás cumple de forma satisfactoria por lo tanto se puede decir que el software desarrollado es altamente portable.

Cumpliendo con estas características, la plataforma para el desarrollo de simuladores de procesos químicos, da respuesta al atributo de calidad, logrando así que sea lo más portable posible y que su uso pueda ser extensivo a las empresas del país que lo necesiten.

Eficiencia: La plataforma para el desarrollo de simuladores de procesos químicos es un software de escritorio por lo que no se necesitan grandes recursos de red, procesadores demasiado potentes, ni servidores, solamente necesita cumplir con las restricciones impuestas por el lenguaje de programación utilizado, o sea, donde se vaya a utilizar el software tiene que estar instalada la máquina virtual de JAVA. Todas estas características influyen directamente en que se pueda lograr que el software desarrollado sea tan eficiente como se requiera y no tan eficiente como sea humanamente posible, por lo tanto se puede decir que se cumple con el atributo de calidad y que la plataforma para el desarrollo de simuladores de procesos químicos es eficiente y que su uso se puede hacer extensivo a empresas donde no existan muchos recursos.

Conclusiones Parciales

En el capítulo anterior como parte de validar la arquitectura propuesta para el producto: Plataforma para el desarrollo de simuladores químicos, se pueden encontrar varios aspectos que influyen en la evaluación de la misma. Para esto se ha hecho un análisis de las técnicas y métodos de evaluación, para escoger el que más resultados pueda ofrecer en la arquitectura. Se hizo un análisis de algunos atributos de calidad y cómo la arquitectura cumple con ellos, quedando demostrado que estos se encuentran totalmente validados y que la arquitectura está apta para ser empleada en el proyecto que se quiere desarrollar.

Conclusiones

El desarrollo de este trabajo permitió elaborar una propuesta de arquitectura para la plataforma para el desarrollo de simuladores de procesos químicos. Para ello se le dio cumplimiento a los objetivos trazados y las tareas propuestas.

Como parte de lograr un mejor funcionamiento del sistema y una adaptación a las características actuales de los sistemas de computación se seleccionaron las tecnologías y herramientas que están presentes en el desarrollo de la propuesta, así como la correcta selección de los estilos y patrones arquitectónicos para conformar la línea base de la arquitectura.

Se identificaron los principales casos de uso, clases del diseño y componentes de implementación que conforman las vistas arquitectónicas del sistema dando un mejor entendimiento a los desarrolladores del software.

Se escogieron los métodos y las técnicas necesarias para la evaluación de la arquitectura, lo que permitió conocer a grandes rasgos cómo se comportará el sistema ante las acciones del cliente, permitiendo afirmar que la arquitectura propuesta cumple con los requerimientos funcionales y no funcionales que debe poseer la plataforma, quedando demostrado que la aplicación de la misma en el proyecto proporcionará una mejor organización y que está validada para ser puesta en práctica.

Recomendaciones

Se recomienda:

- Continuar el refinamiento de la arquitectura en las fases de desarrollo del proyecto.
- Aplicar la arquitectura planteada para integrar los restantes módulos que se implementen para la plataforma.