Universidad de las Ciencias Informáticas

Facultad 9

Propuesta de arquitectura de un subsistema de modelado de Diagramas de Flujo de Información

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

Autor: Reynier Arias Cid.

Tutor: Ing. Arnaldo Gandol Álvarez.

"Ciudad Habana, mayo 2009"

"Año del 50 aniversario del triunfo de la Revolución"

Declaración de Auditoría

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 _____ del año _____.

Reynier Arias Cid

_____

(Autor)

Arnaldo Gandol Álvarez

_____

(Tutor)

Es mejor saber después de haber pensado y discutido, que aceptar los saberes que nadie discute para no tener que pensar.

Fernando Savater

AGRADECIMIENTOS

Agradecer primeramente a mi familia por siempre estar presente en cada una de mis decisiones con respecto a mi vida estudiantil dándome sus consejos, por hacerme entender que en la vida solo lograremos lo que seamos capaces de ser, por darme su apoyo incondicional y por la confianza que en mí han depositado.

Agradecer a los amigos que siempre estuvieron presentes en las buenas y en las malas, a esos que siempre están cuando los necesitas. A los profesores que más que profesores son amigos también y han hecho posible parte de este logro.

A todos…Gracias.

DEDICATORIA

A mi madre…mamá. Por la confianza que en mi has depositado, por entenderme como soy y por hacerme ver sin imposiciones cual es el camino correcto. Por dejarme decidir, por ser la protagonista de mis días, por luchar contra esta vida cuando te ha dado la espalda, tú mamá, eres la razón por la que estoy aquí, eres el impulso de mis aspiraciones. Mamá Eres Todo…y todo precisamente es por ti.

A mis hermanas de verdad sé que gracias a ustedes estoy aquí. Gracias por dejarme ser el niño malcriado ese al que todo se lo consienten, gracias por hacer de mí una mejor persona y guiarme por la vida; hoy comprendo todo lo que se quitaron para darme a mí, hoy sé que este título, este momento no es mío es de todos ustedes yo solo he hecho la parte más fácil, el esfuerzo detrás de este logro ha sido de ustedes, una vez más gracias por ser…, gracias por estar…esto va especialmente dedicado a ustedes.

A mi papá, ya no estás te fuiste muy temprano, pero la educación y conducta que sembraste viven en mí, este título también es tuyo donde quieras que estés, sí me vez papá…te dedico también este triunfo.

A toda mi familia a mi tío Chepo, siempre has sido y serás un padre para mí, este logro también va dedicado a ti…a mi hermano gracias por los consejos, formas parte de mí y de mis actos. A mi padrastro gracias por darle a mairim lo que no pude tener, trata de ser un ejemplo para ella que solo lo que siembres hoy, será la cosecha de mañana…

A mis hermanos de la 313, amigos de los reales, gracias por comprenderme como soy y por tenderme la mano cuando la he necesitado, ustedes son parte de mí…gracias doy por haberlos conocido.

A Mairim mi niñita linda, la luz de mis ojos, vivo por ti mairi siempre voy a estar aquí para ti…siempre intentaré ser contigo como Lely y Coky han sido conmigo, las mejores del mundo…solo deseo ser un ejemplo para ti…

A mi compañera, mi otra niñita, eres quien ha estado conmigo y ha confiado en mí en los momentos más difíciles, gracias por estos años juntos y por aguantarme durante todo este tiempo mis malcriadeces., yo sé que soy difícil, y tu sabes que Te Amo.

Con respeto, cariño y amor va esta especial dedicación a mi mamá y a mis hermanas, las amo con toda mi fe sin medidas…siempre serán mi vida y a mi compañera, quien ha estado conmigo todo este tiempo en la misma batalla, siempre serás mi chucha., solo soy por ustedes, sin ustedes no sería… A todas Las Amo.

Pacho. *14/mayo/2009.*

RESUMEN

La simulación de procesos representa un campo que ha crecido en las últimas décadas como una herramienta para la toma de decisiones apoyadas por el ordenador. Simular es a grandes rasgos modelar el diseño de un proceso real pero a una menor escala.

En Cuba se han utilizado y desarrollado simuladores en ramas como el sector azucarero, en la medicina, en la industria militar y en la química entre otras. Los factores que han influenciado en el crecimiento del desarrollo de simuladores han sido, la velocidad de cálculo alcanzada por los ordenadores, las salidas y entradas de datos a través de las interfaces gráficas, las animaciones, la orientación al usuario y a la facilidad de uso de dichas interfaces, incrementando el valor de uso de los simuladores a nivel internacional y nacional.

Los simuladores de procesos industriales cubanos poseen opciones y herramientas para el análisis y cálculo de indicadores de los componentes, pero se caracterizan por poseer interfaces gráficas carentes de facilidades para los clientes a la hora de realizar el modelado de los diagramas de flujo de información. El uso y desarrollo de estos simuladores está limitado porque la arquitectura no es modificable ni flexible a nuevos cambios.

De lo anterior surge la necesidad del desarrollo de un subsistema de interfaz gráfica para el simulador DFISim. El objetivo de la presente investigación es definir la arquitectura del subsistema de interfaz gráfica de forma tal que permita cumplir con los requerimientos funcionales y no funcionales del modelado de DFI a través de todo el ciclo de desarrollo.

Palabras claves: simulación, procesos, información, arquitectura, interfaces.

Índice de Ilustraciones

Índice de Contenido

Índice de Tablas

Introducción

La simulación de procesos representa un gran campo que ha crecido en las últimas décadas como una poderosa herramienta a la hora de tomar decisiones apoyadas por el ordenador. Simular es a grandes rasgos modelar el diseño de un proceso real con sus datos reales pero a una menor escala.

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. [1]

En Cuba se han utilizado y desarrollado varios simuladores sobre todo en la rama del sector azucarero, en la medicina, en la rama militar, en la industria química, entre otras. Existe una gran experiencia acumulada en cuanto a la rama simulación de procesos y desarrollo de los mismos. Algunos de los factores que han influenciado en el crecimiento del desarrollo de simuladores han sido, la gran velocidad de cálculo alcanzada por los ordenadores, las salidas y entradas de datos a través de las interfaces gráficas, las animaciones, la orientación al usuario y a la facilidad del mismo a la hora de utilizar dichas interfaces, lo que ha incrementado el valor de uso de los simuladores a nivel internacional y nacional.

Actualmente existe en Cuba el Centro de Investigación y Desarrollo de Simuladores (SIMPRO), perteneciente a las Fuerzas Armadas Revolucionarias (FAR), en el cual se han desarrollado varios simuladores con interfaces de usuarios muy personalizadas, como son simuladores de tanques de guerra con el fin de aprender a utilizarlos sin correr el riesgo de heridos o muertes humanas en las maniobras, simuladores de tiro, representación de terrenos en tres dimensiones (3D), entre otros, obteniéndose resultados avanzados en la rama de la realidad virtual, en la representación en (3D) y en el tratamiento de imágenes, demostrado además a partir de diversos proyectos llevados a cabo en la Universidad de la Ciencias Informáticas (UCI), como el producto CASANDRA con un fuerte contenido de tratamiento de

imágenes a pesar de no ser precisamente un simulador de procesos industriales. El STA v4.0 basado en la industria química es un simulador de procesos industriales que cuenta con posibilidades avanzadas de cálculo y análisis de resultados, sin embargo con poca orientación al usuario en lo que a su interfaz respecta. Los simuladores de procesos industriales desarrollados en Cuba poseen opciones y herramientas avanzadas para el análisis y cálculo de indicadores de los diferentes componentes, pero se caracterizan por poseer la desventaja de contar con interfaces gráficas carentes de facilidades para los clientes los cuales sugieren mayor sencillez a la hora de realizar el modelado de los diagramas de flujo a nivel de interfaz.

De lo anteriormente expuesto surge entonces la **situación problemática** de la presente investigación, la cual está basada fundamentalmente en que en Cuba el uso y desarrollo de simuladores como principal herramienta de ayuda a la toma de decisiones está limitado principalmente porque los simuladores cubanos carecen de muchas herramientas que poseen sus homólogos extranjeros, cuestión que se ha intentado erradicar en los últimos años y en la que se ha obtenido un gran avance aunque aún falta mucho por recorrer en el camino de la simulación de procesos industriales, por otra parte, la arquitectura de los simuladores extranjeros no es modificable por estar sujetos a licencias de software propietario en la mayoría de los casos, aunque no en todos y 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.

Se ha identificado como **Problema Científico** de la investigación:

La Necesidad del desarrollo de un subsistema de modelado, entrada y salida de datos de diagramas de flujo de información para el simulador DFISim del polo Simulación de la facultad 9 de la Universidad de las Ciencias Informáticas.

Objetivo General:

Definir la arquitectura de un subsistema que permita modelar los Diagramas de Flujo de Información del simulador DFISim de forma tal que permita realizar las funcionalidades críticas del mismo a través de todo el ciclo de desarrollo.

Objetivos Específicos:

- Identificar los patrones arquitectónicos adecuados para lograr una organización del subsistema de forma tal que permita agregar nuevas funcionalidades sin necesidad de realizar cambios significativos en la arquitectura.
- Identificar los estilos arquitectónicos que puedan ser utilizados en la descripción de la arquitectura del subsistema a desarrollar.
- Comparar la arquitectura propuesta con otras soluciones similares.

Objeto de Estudio:

Proceso de desarrollo de los simuladores de procesos industriales.

Campo de acción:

Descripción de la arquitectura de los subsistemas de interfaz gráfica de modelado de diagramas en los simuladores de procesos industriales.

Idea a Defender:

La definición arquitectónica de un subsistema de interfaz gráfica para el simulador DFISim permitirá modelar los diagramas de flujo de información a partir de la realización de las funcionalidades críticas del subsistema.

Tareas de la investigación:

- Caracterizar las interfaces gráficas de simuladores de procesos industriales.
- Seleccionar las herramientas a utilizar en el proceso de desarrollo del subsistema de modelado de diagramas de flujo de información.
- Seleccionar los estilos arquitectónicos que puedan ser utilizados en el desarrollo del subsistema de modelado de DFI.
- Identificar los patrones arquitectónicos que puedan ser utilizados en la arquitectura del subsistema de interfaz gráfica para el simulador DFISim según las características del mismo.
- Identificar los aspectos significativos presentes en el desarrollo del Simulador DFISim que condicionan la arquitectura del subsistema de interfaz grafica a desarrollar.
- Describir la arquitectura del subsistema a desarrollar a través de las fases de desarrollo.

- Comparar la arquitectura propuesta con soluciones arquitectónicas similares.

Posibles resultados:

- Descripción de la arquitectura del subsistema de modelado de DFI.
- Propuesta de la arquitectura para el desarrollo del subsistema de modelado de DFI.
- Identificación de componentes reutilizables a nivel del Simulador DFISim dado el grado de significación arquitectónico del subsistema a desarrollar.

Métodos de la Investigación:

Métodos Teóricos:

Análisis Histórico Lógico:

En la investigación se analiza la trayectoria del desarrollo de simuladores de procesos industriales hasta la actualidad. Se toma en cuenta además su nivel alcanzado, valorando las opciones arquitectónicas, organizaciones de los componentes y partes de simuladores, así como sus relaciones.

Analítico-Sintético:

En la investigación se analizan cada uno de los componentes de la arquitectura del subsistema a desarrollar por separado teniendo en cuenta las partes básicas fundamentales que componen y constituyen las interfaces gráficas de simuladores de procesos industriales, sus relaciones y organización. Además se toma en cuenta estas partes arquitectónicas como la integración de un todo que permita una visión general dentro del proceso de desarrollo del subsistema así como un entendimiento común por parte de los implicados.

Modelación:

Dentro de la investigación se utilizan diferentes abstracciones como son los modelos de los estilos arquitectónicos, el prototipo inicial del subsistema que debe surgir en la primera etapa de su desarrollo y que a pesar de no ser el subsistema final a integrar con simulador como un todo, si constituirá un modelo semejante al núcleo del mismo permitiendo de esa manera evaluar la arquitectura propuesta en cuanto a los requerimientos que debe cumplir el subsistema.

Métodos Empíricos:

Entrevista:

En la investigación se le aplicará diferentes tipos de entrevistas al personal implicado y a usuarios finales de la aplicación a desarrollar con el fin de obtener retroalimentación de las operaciones y eventos que se llevan a cabo dentro el ambiente del subsistema a desarrollar a través de todo el ciclo de desarrollo del mismo.

1. Capítulo 1

Introducción

En este capítulo se abordarán temas relacionados con el dominio de la investigación, se realizará un acercamiento teórico a los conceptos de arquitectura de software, estilos y patrones arquitectónicos seleccionando además los que resulten adecuados para el desarrollo del subsistema de modelado de DFI. El objetivo fundamental del presente capítulo es lograr un buen entendimiento de las temáticas fundamentales que se relacionan con el desarrollo de software y la arquitectura de software así como con las herramientas utilizadas en ambas y seleccionar las adecuadas en dependencia del subsistema que se desea desarrollar. Se aborda además desde una perspectiva histórico–lógica, las diferentes concepciones, definiciones y conceptos de estas temáticas mencionadas.

Marco Teórico Conceptual: Estado del Arte y Fundamentación Teórica.

1.1. Arquitectura de Software

Los orígenes de la arquitectura de software, (AS), datan del 1968, cuando Edsger Dijkstra, 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. [2]. Dijkstra, quien sostenía que las ciencias de la computación eran una rama aplicada de las matemáticas y sugería seguir pasos formales para descomponer problemas mayores, fue uno de los introductores de la noción de sistemas operativos organizados en capas que se comunican sólo con las capas adyacentes y que se superponen “como capas de cebolla”,[2], a pesar de que solo se refería a lo que luego se conoció como “programación en grande”, buenas prácticas de programación que fuesen acogidas por los programadores de la época y luego por los arquitectos.

En la conferencia de NATO 1969 se formularon sorprendentes afirmaciones acerca de las ideas de Dijkstra, entre ellas que dentro de la disciplina se le prestaba en aquel entonces muy poco tiempo al diseño y organización basándose solo en el cumplimiento de lo que debe hacer un programa y no en cómo debe hacerlo, alegando en que no interesaba el embellecimiento de los productos de tipo software, aludiendo entonces al termino, Arquitectura de Software como el ente que se encargaría de darle la debida belleza a determinada estructura de componentes y a la relación entre ellos.

A través de los años surgen diferentes personalidades que publican ensayos y proponen diferentes definiciones; una novedad importante en la década de 1970 fue el advenimiento del diseño estructurado y de los primeros modelos explícitos de desarrollo de software. Estos modelos comenzaron a basarse en una estrategia más orgánica, evolutiva, cíclica, dejando atrás las metáforas del desarrollo en cascada que se inspiraban más bien en la línea de montaje de la ingeniería del hardware y la manufactura. Surgieron entonces las primeras investigaciones académicas en materia de diseño de sistemas complejos o "intensivos". Poco a poco el diseño se fue independizando de la implementación, y se forjaron herramientas, técnicas y lenguajes de modelado específicos.

En la década de 1980, los métodos de desarrollo estructurado demostraron no escalar suficientemente y fueron dejando el lugar a un nuevo paradigma, el de la programación orientada a objetos, (POO). En teoría, parecía posible modelar el dominio del problema y el de la solución en un lenguaje de implementación. La investigación que llevó a lo que después sería el diseño orientado a objetos puede remontarse incluso a la década de 1960 con Simula, un lenguaje de programación de simulaciones, el primero que proporcionaba tipos de datos abstractos y clases, y después fue refinada con el advenimiento de Smalltalk. Paralelamente, hacia fines de la década de 1980 y comienzos de la siguiente, la expresión arquitectura de software comienza a aparecer en la literatura para hacer referencia a la configuración morfológica de una aplicación.

El primer estudio en que aparece la expresión "arquitectura de software", en el sentido en que hoy se le conoce es en el de Perry y Wolf en 1992, aunque el trabajo se fue gestando desde 1989. En él, los autores proponen concebir la AS por analogía con la arquitectura de edificios. Es la década de los 90 una etapa que permitió la consolidación y diseminación de la AS es una época rica en acontecimientos relacionados con la industria del software y con la arquitectura. Surge entonces la programación basada en componentes y los patrones otro de los grandes y polémicos temas dentro del campo originado por Christopher Alexander, quien incidentalmente fue arquitecto de edificios.

Muchas son las definiciones de AS surgidas en las diferentes épocas, el número de definiciones circulantes alcanza un orden de tres dígitos, amenazando llegar a cuatro. De hecho, existen grandes compilaciones de definiciones alternativas o contrapuestas. En general, las definiciones interrelacionan el trabajo dinámico de estipulación de la arquitectura dentro del proceso de ingeniería o el diseño y la

configuración o topología estática de sistemas de software contemplada desde un elevado nivel de abstracción.

Una definición reconocida es la de Clements en el texto “A Survey of Architecture Description Languages” en 1996 donde 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.

David Garlan en el libro “Software Architecture: A Roadmap” en el año 2000, 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.

Muchas son las definiciones de AS pero la definición “oficial” se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, que define: 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.

1.2. Lenguajes de Descripción de Arquitectura ADLs

Los lenguajes de descripción de arquitecturas, o ADLs, ocupan una parte importante del trabajo arquitectónico desde la fundación de la disciplina. Se trata de un conjunto de propuestas de variado nivel de rigurosidad, casi todas ellas de extracción académica, que fueron surgiendo desde comienzos de la década de 1990 hasta la actualidad, más o menos en contemporaneidad con el proyecto de unificación de los lenguajes de modelado bajo la forma de UML. Los ADLs difieren sustancialmente de UML, que al menos en su versión 1.x se estima inadecuado en su capacidad para expresar conectores en particular y en su modelo semántico en general para las clases de descripción y análisis que se requieren. 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.

Los ADLs son bien conocidos en los estudios universitarios de AS, pero muy pocos arquitectos de industria parecen conocerlos y son menos aún quienes los utilizan como instrumento en el diseño arquitectónico de sus proyectos.

Los ADLs tienen características fundamentales que los distinguen de otros lenguajes como son los de programación y los de modelado entre estas se encuentran:

- La abstracción que proveen al usuario es de naturaleza arquitectónica.
- La mayoría de las vistas provistas por estos lenguajes contienen información predominantemente arquitectónica.
- El análisis provisto por el lenguaje se fundamenta en información de nivel arquitectónico.

Mary Shaw define que un lenguaje de descripción arquitectónica debe poseer además ciertas características entre las cuales destaca:

- Capacidad para representar componentes (primitivos o compuestos), así como sus propiedades, interfaces e implementaciones.
- Capacidad de representar conectores, así como protocolos, propiedades e implementaciones.
- Abstracción y encapsulamiento.
- Tipos y chequeo de tipos.
- Capacidad de integración de herramientas de análisis.

Existen hoy en día diversos lenguajes de descripción arquitectónica dentro de la industria del software, algunos son considerados ADLs, otros son tomados como lenguajes que de alguna manera permiten la modelación de elementos arquitectónicos a pesar de tener marcadas diferencias algunos de los ADLs se presentan en la Tabla 3.

ADL	Fecha	Investigador - Organismo	Observaciones
Acme	1995	Monroe & Garlan (CMU), Wile (USC)	Lenguaje de intercambio de ADLs
Aesop	1994	Garlan (CMU)	ADL de propósito general, énfasis en estilos
ArTek	1994	Terry, Hayes-Roth, Erman (Teknowledge, DSSA)	Lenguaje específico de dominio - No es ADL
Armani	1998	Monroe (CMU)	ADL asociado a Acme
C2 SADL	1996	Taylor/Medvidovic (UCI)	ADL específico de estilo
CHAM	1990	Berry / Boudol	Lenguaje de especificación
Darwin	1991	Magee, Dulay, Eisenbach, Kramer	ADL con énfasis en dinámica
Jacal	1997	Kicillof , Yankelevich (Universidad de Buenos Aires)	Adl - Notación de alto nivel para descripción y prototipado
LILEANNA	1993	Tracz (Loral Federal)	Lenguaje de conexión de módulos
MetaH	1993	Binns, Englehart (Honeywell)	ADL específico de dominio
Rapide	1990	Luckham (Stanford)	ADL & simulación
SADL	1995	Moriconi, Riemenschneider (SRI)	ADL con énfasis en mapeo de refinamiento
UML	1995	Rumbaugh, Jacobson, Booch (Rational)	Lenguaje genérico de modelado – No es ADL
UniCon	1995	Shaw (CMU)	ADL de propósito general, énfasis en conectores y estilos
Wright	1994	Garlan (CMU)	ADL de propósito general, énfasis en comunicación
xADL	2000	Medvidovic, Taylor (UCI, UCLA)	ADL basado en XML

Figura 1: Representación de lenguajes de descripción de arquitectura (ADLs). Fuente [2].

1.3. Tendencias Actuales de la Arquitectura de Software

1.3.1. Arquitectura como etapa de ingeniería y diseño orientado a objetos

Es el modelo de James Rumbaugh, Ivar Jacobson, Grady Booch, Craig Larman y otros, ligado al mundo de UML (Lenguaje Unificado de Modelado) y Rational Unified Process (Proceso Unificado de Rational). Se trata de una corriente específica:". En esta postura, la arquitectura se restringe a las fases de inicio y preliminares del proceso y concierne a los niveles más elevados de abstracción, pero no está sistemáticamente ligada al requerimiento que viene antes o a la composición del diseño que viene después. Se denomina arquitectura fundamentalmente a la organización y estructura de las piezas del sistema. Es una tendencia que reconoce el valor primordial de la abstracción y del ocultamiento de información, pero estos conceptos tienen que ver más con el encapsulamiento en clases y objetos que con la visión de conjunto arquitectónica. Para este movimiento, la arquitectura se confunde también con el modelado y el diseño, los cuales constituyen los conceptos dominantes. Esta corriente manifiesta

predilección por un modelado denso y una profusión de diagramas. Importa más la abundancia y el detalle de diagramas y técnicas disponibles que la simplicidad de la visión de conjunto. No utiliza ADLs, reconocidos y utilizados por la academia de AS; se utiliza un lenguaje unificado de modelado (UML) para las descripciones arquitectónicas. La definición de arquitectura que se promueve tiene que ver con aspectos formales a la hora del desarrollo; esta arquitectura es isomorfa a la estructura de las piezas de código. Una definición típica y demostrativa sería la de Grady Booch; la AS es la estructura lógica y física de un sistema, forjada por todas las decisiones estratégicas y tácticas que se aplican durante el desarrollo. Otras definiciones revelan que la AS, en esta perspectiva, concierne a decisiones sobre organización, selección de elementos estructurales, comportamiento, composición y estilo arquitectónico susceptibles de ser descriptas a través de las cinco vistas clásicas del modelo 4+1 de Kruchten.

1.3.2. Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.

Constituye la corriente fundacional y clásica de la disciplina. Los representantes de esta corriente son todos académicos, mayormente de la Universidad Carnegie Mellon en Pittsburgh: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom. Se trata también de la visión de la AS dominante en la academia, y aunque es la que ha hecho el esfuerzo más importante por el reconocimiento de la AS como disciplina, sus categorías y herramientas son todavía mal conocidas en la práctica industrial. Hay una variante informal y una vertiente más formalista. En principio se pueden reconocer tres modalidades en cuanto a la formalización; los más informales utilizan descripciones verbales o diagramas de cajas, los de talante intermedio se sirven de ADLs y los más exigentes usan lenguajes formales de especificación. En toda la corriente, el diseño arquitectónico no sólo es el de más alto nivel de abstracción, sino que además no tiene por qué coincidir con la configuración explícita de las aplicaciones; rara vez se encontrarán referencias a lenguajes de programación o piezas de código, y en general nadie habla de clases o de objetos. Mientras algunos participantes excluyen el modelo de datos de las incumbencias de la AS (Shaw, Garlan, etc.), otros insisten en su relevancia (Medvidovic, Taylor). [2]

1.3.3. Estructuralismo arquitectónico radical.

Se trata de un desprendimiento de la corriente anterior, mayoritariamente europeo, que asume una actitud más confrontante con el mundo UML. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos (OO) para la AS y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación[2].

1.3.4. Arquitectura basada en patrones.

Reconoce la importancia de un modelo emanado del diseño OO, esta corriente surgida hacia 1996 no se encuentra tan rígidamente vinculada a UML. El texto sobre patrones que esta variante reconoce como referencia es la serie POSA de Buschmann y secundariamente el texto de la Banda de los Cuatro referido a patrones en el texto Design Patterns, Elements of reusable object-oriented software. En esta manifestación de la AS prevalece cierta tolerancia hacia modelos de procesos tácticos, no tan macroscópicos. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

1.3.5. Arquitectura procesual.

Desde comienzos del siglo XXI, con centro en el SEI y con participación de algunos arquitectos de Carnegie Mellon: Rick Kazman, Len Bass, Paul Clements, intenta establecer modelos de ciclo de vida y técnicas de elicitación de requerimientos, brainstorming, diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica específicas para la arquitectura de software. Otras variantes dentro de la corriente procesual caracterizan de otras maneras de etapas del proceso: extracción de arquitectura, generalización, reutilización.

1.3.6. Arquitectura basada en escenarios.

Es la corriente más nueva. Se trata de un movimiento predominantemente europeo, con centro en Holanda. Recupera el nexo de la AS con los requerimientos y la funcionalidad del sistema, ocasionalmente borroso en la arquitectura estructural clásica. El movimiento constituye una especialización de la AS procesual. Suele utilizarse diagramas de casos de uso, UML como herramienta informal u ocasional, dado que los casos de uso son uno de los escenarios posibles. Los casos de uso no están orientados a objeto. Los autores vinculados con esta modalidad han sido, en su mayoría los codificadores de ATAM, CBAM, QASAR y demás métodos de evaluación de arquitectura del SEI.

Han existido intentos de fundación de otras variedades de AS además de las antes planteadas, tales como una arquitectura adaptativa inspirada en ideas de la programación genética o en teorías de la complejidad, la auto-organización, el caos y los fractales, una arquitectura centrada en la acción que recurre a la inteligencia artificial heideggeriana o al postmodernismo, y una arquitectura epistemológicamente reflexiva que tiene a Popper o a Kuhn entre sus referentes. Existe además un

movimiento cismático más serio y formalista y es probable que pueda hablarse también de una anti-arquitectura, que en nombre de los métodos heterodoxos se opone tanto al modelado orientado a objetos y los métodos sobre documentados impulsados por consultoras corporativas como a la abstracción arquitectónica que viene de la academia. Discernir la naturaleza de cada una de las tendencias puede tener efectos prácticos a la hora de comprender antagonismos, concordancias, sesgos, disyuntivas, incompatibilidades y fuentes de recursos dentro de la Arquitectura de Software.

1.4. Estilos y Patrones Arquitectónicos

Estilo Arquitectónico:

El éxito de una arquitectura de software dependerá de la elección correcta de los estilos y patrones arquitectónicos a utilizar en dependencia del tipo de aplicación a desarrollar.

¿Qué se define como un estilo arquitectónico?

Mary Shaw y Paul Clements 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 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.

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. [2]

Roy Thomas Fielding sintetiza la definición de estilo pleonásticamente, diciendo que un estilo arquitectónico es un conjunto coordinado de restricciones arquitectónicas que restringe los roles/rasgos de los elementos arquitectónicos y las relaciones permitidas entre esos elementos dentro de la arquitectura que se conforma a ese estilo.

A pesar de la diversidad de criterios y definiciones, lo cierto es que existen una serie de elementos comunes en las definiciones dígase entonces que los estilos arquitectónicos especifican y describen los componentes y conectores que pueden conformar el sistema o subsistema, es decir los componentes y los conectores que especifican el modo en que estos interactúan conformando además las restricciones que acotan las interacciones y relaciones entre dichos componentes dentro de la arquitectura del estilo a utilizar. Además se debe tomar en cuenta en la selección de un estilo u otro, el tipo de aplicación y las ventajas que aporte dicho estilo a la misma. Los estilos arquitectónicos más utilizados en la actualidad se encuentran divididos en función de algunos criterios en clases o grupos entre estos se encuentran:

1.4.1. Estilo Flujo de Datos

Esta familia de estilos enfatiza la reutilización y la capacidad de modificación. Es apropiada para sistemas que implementan transformaciones de datos en pasos sucesivos. Ejemplares de la misma serían las arquitecturas de tubería-filtros y las de proceso secuencial en lote.

- Tubería y filtros:

Una tubería (pipeline) es una popular arquitectura que conecta componentes computacionales (filtros) a través de conectores (pipes), de modo que las computaciones se ejecutan a la manera de un flujo. Los datos se transportan a través de las tuberías entre los filtros, transformando gradualmente las entradas en salidas. 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. En el estilo secuencial por lotes (batch sequential) los componentes son programas independientes; el supuesto es que cada paso se ejecuta hasta completarse antes que se inicie el paso siguiente. Una desventaja referida a la utilización de estilos concierne a que eventualmente pueden llegar a requerirse buffers de tamaño indefinido, no es un estilo apto para manejar situaciones interactivas, sobre todo cuando se requieren actualizaciones incrementales de la representación en pantalla.

1.4.2. Estilos Centrados en Datos

Esta rama de estilos enfatiza la integridad de los datos. Se estima apropiada para sistemas que se fundan en acceso y actualización de datos en estructuras de almacenamiento. Sub-estilos característicos de estos serían los repositorios, las bases de datos, las arquitecturas basadas en hipertextos y las arquitecturas de pizarra.

❖ Arquitectura de Pizarra o Repositorio:

En esta arquitectura hay dos componentes principales: una estructura de datos que representa el estado actual y una colección de componentes independientes que operan sobre él. En base a esta distinción se han definidos dos subcategorías principales del estilo:

- Si los tipos de transacciones en el flujo de entrada definen los procesos a ejecutar, el repositorio puede ser una base de datos tradicional (implícitamente no cliente-servidor).
- Si el estado actual de la estructura de datos dispara los procesos a ejecutar, el repositorio es lo que se llama una pizarra pura o un tablero de control.

Estos sistemas se han usado en aplicaciones que requieren complejas interpretaciones de proceso de señales (reconocimiento de patrones, reconocimiento de habla, etc.), o en sistemas que involucran acceso compartido a datos con agentes débilmente acoplados.

1.4.3. Estilos de Llamada y Retorno

Esta familia de estilos enfatiza la capacidad de modificación y la escalabilidad de la arquitectura. Son los estilos más generalizados en sistemas en gran escala. Miembros de esta familia de estilos, son las arquitecturas de programa principal y subrutina, los sistemas basados en llamadas a procedimientos remotos, los sistemas orientados a objeto y los sistemas jerárquicos en capas.

❖ Arquitectura en Capas:

El estilo en capas está estructurado 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. Las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas. 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. Además admite muy naturalmente optimizaciones y refinamientos y proporciona amplia reutilización y permite la construcción de sistemas débilmente acoplados, por lo que minimiza las dependencias entre capas y resulta más fácil sustituir la implementación de una capa sin afectar al resto del sistema.

- Arquitecturas Orientadas a Objetos:

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos y los componentes se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo 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. Los objetos interactúan a través de invocaciones de funciones y procedimientos.

- Arquitectura Basada en Componentes:

En este estilo arquitectónico los componentes son las unidades de modelado, diseño e implementación, su funcionalidad y propiedades puedan ser descubiertas y utilizadas en tiempo de ejecución. Las interfaces están separadas de las implementaciones, y sus interacciones son el centro de incumbencias en el diseño arquitectónico.

1.4.4. Estilos de Código Móvil

Esta familia de estilos enfatiza la portabilidad. Ejemplos de la misma son los intérpretes, los sistemas basados en reglas y los procesadores de lenguaje de comando. Fuera de las máquinas virtuales y los intérpretes, los otros miembros del conjunto han sido rara vez estudiados desde el punto de vista estilístico de la arquitectura.

❖ Arquitectura de Maquinas Virtuales:

Arquitectura de Máquinas Virtuales: Esta arquitectura se conoce como intérpretes basados en tablas ó sistemas basados en reglas. Estos sistemas se representan mediante un seudo-programa a interpretar y una máquina de interpretación. Estas variedades incluyen un extenso espectro que está comprendido desde los llamados lenguajes de alto nivel hasta los paradigmas declarativos no secuenciales de programación. Las aplicaciones inscritas 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.[2]

1.4.5. Arquitectura Basada en Atributos.

La arquitectura basada en atributos o ABAS fue propuesta por Klein y Klazman [2]. La intención de estos autores es asociar a la definición del estilo arquitectónico un Framework de razonamiento (ya sea cuantitativo o cualitativo) basado en modelos de atributos específicos. Su objetivo se funda en la premisa que dicha asociación proporciona las bases para crear una disciplina de diseño arquitectónico. Se relacionan fuertemente los atributos de calidad con la arquitectura de software y se estima que un estilo arquitectónico basado en atributos posee:

- La topología de los tipos de componentes y una descripción del patrón de los datos y control de interacción entre ellos.
- Un modelo específico de atributos de calidad que provee un método de razonamiento acerca del comportamiento de los tipos de componentes que interactúan en el patrón definido.
- El razonamiento que resulta de la aplicación del modelo específico de atributos de calidad a la interacción de los tipos de componentes.

Un estilo arquitectónico basado en atributos (ABAS) consta de las cinco partes que se muestran en la tabla 1.

Elemento	Descripción

Descripción del problema	Describe el problema de diseño que el ABAS pretende resolver incluyendo el atributo de calidad de interés, el contexto de uso, y requerimientos específicos relevantes al atributo de calidad asociado.
Medidas del atributo de calidad	Contiene los aspectos medibles del modelo de atributos de calidad. Incluye una discusión de los eventos que causan que la arquitectura responda o cambie.
Estilo Arquitectónico	Descripción del estilo arquitectónico en términos de componentes, correctores, propiedades de los componentes y conexiones, así como patrones de datos de control de iteraciones.
Parámetros de atributos de calidad	Especificación del estilo arquitectónico en términos de los parámetros del modelo de calidad.
Análisis	Descripción de cómo los modelos de atributos de calidad están formalmente relacionados con los elementos del estilo arquitectónico y las conclusiones acerca del comportamiento arquitectónico que se desprende de los modelos.

Tabla 1: Partes que conforman una arquitectura basada en atributos (ABAS), fuente[3] y propia.

1.4.6. Estilos Peer to Peer

Esta agrupación de estilos, 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 mediante broadcast.

- Arquitecturas Basadas en Eventos.

Las arquitecturas basadas en eventos se han llamado también de invocación implícita. Otros nombres propuestos para el mismo estilo han sido integración reactiva o difusión (broadcast) selectiva. Existen estrategias de programación basada en eventos, sobre todo referida a interfaces de usuario, y hay además eventos en los modelos de objetos y componentes, pero no es a eso a lo que se refiere primariamente el estilo, aunque esa variedad no está del todo excluida. En términos de patrones de diseño, el patrón que corresponde más estrechamente a este estilo es el que se conoce como Observer, un término que se hizo popular en Smalltalk a principios de los ochenta; en el mundo de Java se le conoce como modelo de delegación de eventos.

La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos) un componente puede anunciar mediante difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento. Desde el punto de vista arquitectónico, los componentes de un estilo de invocación implícita son módulos cuyas interfaces proporcionan tanto una colección de procedimientos (igual que en el estilo de tipos de datos abstractos) como un conjunto de eventos. Los procedimientos se pueden invocar a la manera usual en modelos orientados a objeto, o mediante el sistema de suscripción que se ha descripto.

Los ejemplos de sistemas que utilizan esta arquitectura son numerosos. El estilo se utiliza en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia, en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos entre otros.

- Arquitecturas Orientadas a Servicios (SOA en Ingles)

Este estilo está apoyado fundamentalmente es los Web services basados en XML, en los cuales los formatos de intercambio se basan en XML 1.0 Namespaces y el protocolo de elección es SOAP que no es más un formato de mensajes que es XML, comunicado sobre un transporte que por defecto es HTTP, pero que puede ser también HTTPs, SMTP, FTP, IIOP, MQ entre otros. Un Web service es un sistema de software diseñado para soportar interacción máquina-a-máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina específicamente WSDL en el caso de SOA. Otros sistemas interactúan con el Web service de una manera prescripta por su descripción utilizando mensajes

SOAP, típicamente transportados usando HTTP con una serialización en XML en conjunción con otros estándares relacionados.

1.5. Relación entre Estilos Arquitectónicos y Patrones Arquitectónicos

Patrón Arquitectónico:

Existen varias definiciones de patrón arquitectónico entre ellas Buschmann define patrón como una regla que consta de tres partes, la cual expresa una relación entre un contexto, un problema y una solución. [3]

En líneas generales, un patrón sigue el siguiente esquema:

- Contexto. Es una situación de diseño en la que aparece un problema de diseño.
- Problema. Es un conjunto de fuerzas que aparecen repetidamente en el contexto.
- Solución. Es una configuración que equilibra estas fuerzas. Ésta abarca: Estructura con componentes y relaciones Comportamiento a tiempo de ejecución: aspectos dinámicos de la solución, como la colaboración entre componentes, la comunicación entre ellos, etc.

Partiendo de esta definición, Buschmann plantea los patrones arquitectónicos como descripción de un problema particular y recurrente de diseño, que aparece en contextos de diseño específico, y que presenta un esquema genérico demostrado con éxito para su solución. El esquema de solución se especifica mediante la descripción de los componentes que la constituyen, sus responsabilidades y desarrollos, así como también la forma como estos colaboran entre sí. Además los patrones expresan el esquema de organización estructural fundamental para sistemas de software. Provee un conjunto de subsistemas predefinidos, especifica sus responsabilidades e incluye reglas y pautas para la organización de las relaciones entre ellos. Propone que son plantillas para arquitecturas de software concretas, que especifican las propiedades estructurales de una aplicación con amplitud de todo el sistema y tienen un impacto en la arquitectura de subsistemas. La selección de un patrón arquitectónico es, por lo tanto, una decisión fundamental de diseño en el desarrollo de un sistema de software.

Los patrones son similares a los estilos en la medida en que definen una familia de sistemas de software que comparte características comunes, pero también difieren en dos importantes aspectos. Un estilo

representa específicamente una familia arquitectónica, construida a partir de bloques de construcción arquitectónicos, tales como los componentes y los conectores. Los patrones, en cambio, atraviesan diferentes niveles de abstracción y etapas del ciclo de vida partiendo del análisis del dominio, pasando por la arquitectura de software y yendo hacia el nivel de los lenguajes de programación[2]. A partir de la definición de Buschmann se ha confeccionado la siguiente tabla que contiene las principales diferencias que poseen los conceptos de patrón y estilo arquitectónicos.[3]

Estilo Arquitectónico	Patrón Arquitectónico
Describe el esqueleto estructural y general para aplicaciones.	Existen en varios rangos de escala, comenzando con patrones que definen la estructura básica de una aplicación.
Son independientes del contexto al que puedan ser aplicados.	Requieren de la especificación de un contexto del problema.
Cada estilo es independiente de otros.	Depende de patrones más pequeños que contiene, patrones con los interactúan o de patrones que lo contengan.
Son una categorización de sistemas.	Son soluciones recurrentes generales a problemas que se han vuelto comunes.

Tabla 2: Diferencias entre estilo arquitectónico y patrón arquitectónico, fuente [3] y propia.

Los patrones se pueden dividir a partir de su nivel de abstracción según Buschmann en las categorías de Patrones de Diseño, Patrones de Arquitectura e Idiomas. [4]. Además es precisamente el nivel de abstracción el aspecto fundamental que relaciona a los mismos siendo la parte más baja o ínfima del esquemas los patrones de diseño poco significativos para la arquitectura con la excepción de algunas ramas como las propuestas por RUP utilizando el UML como lenguaje de modelación de la arquitectura.

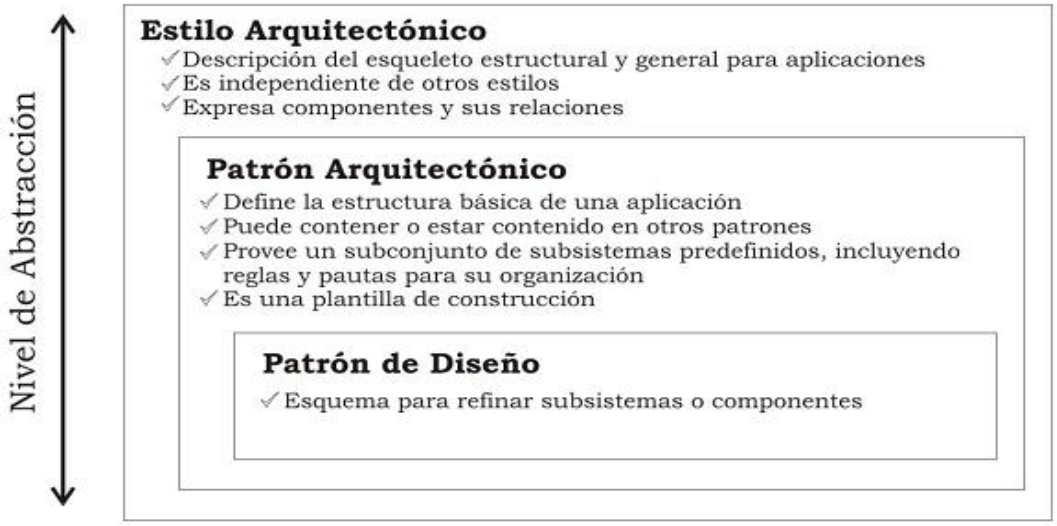

Figura 2: Relación entre Estilo Arquitectónico, Patrón Arquitectónico y Patrón de Diseño. fuente [3].

Existen dos corrientes básicas en términos de patrones de arquitectura:

Patrones Orientados a Arquitectura de Software (POSA)**:** Estos patrones agrupan una vista general de distintos niveles de abstracción en concepción de patrones, entre ellos un grupo orientado a un alto nivel de arquitectura. Los patrones aquí definidos son también tratados indistintamente como estilos arquitectónicos.

Ejemplos:

- Layers
- Pipes and Filters
- Blackboard
- Broker
- Model-View-Controller
- Presentation-Abstraction-Control

- Microkernel
- Reflection

Patrones de Arquitectura Empresarial (PEEA): Se centran en unificar las partes de una aplicación empresarial mediante formas comunes. Se basan en experiencias recopiladas sobre formas de modelar partes de un sistema, fundamentalmente mediante capas y la forma en que estas internamente se organizan.

Ejemplos de patrones PEAA:

Capa de Dominio

- Transaction Script
- Domain Model
- Table Module

Servidor Web

- Page Controller
- Template View
- Two Step View

❖ **Patrones de Diseño.**

Un patrón de diseño provee un esquema para refinar los subsistemas o componentes de un sistema de software, o las relaciones entre ellos. Describe la estructura comúnmente recurrente de los componentes en comunicación, que resuelve un problema general de diseño en un contexto particular. [3].

Existen gran cantidad de patrones de diseño es casi imposible revisarlos todos a la hora de desarrollar una determinada aplicación, por eso se recomienda el uso de patrones de diseño que estén asociados a

cada uno de los estilos y patrones arquitectónicos que se seleccionen para el desarrollo de la arquitectura de la aplicación a desarrollar. Según su propósito o su ámbito los patrones de diseño poseen diferentes agrupaciones como se muestra a continuación:

Los patrones de diseño según su propósito se agrupan en:

- Patrones de Creación
- Patrones Estructurales
- Patrones de Comportamiento

Según su ámbito se agrupan en:

- De clase: Basados en la herencia de clases
- De objeto: Basados en la utilización dinámica de objetos

Son reconocidos dentro del campo del desarrollo de software los patrones definidos por la “Banda de los Cuatro” cuando definen una recopilación de patrones en el libro “Design Patterns” entre los cuales se pueden encontrar:

Creacionales

- Abstract Factory
- Builder
- Factory Method
- Prototype
- Singleton

Estructurales

- Adapter
- Bridge
- Composite
- Decorator
- Facade
- Flyweight
- Proxy

Composición

- Chain of Responsibility
- Command
- Interpreter
- Iterator
- Mediator
- Memento
- Observer
- State
- Strategy
- Template Method

- Visitor

1.6. Metodologías de Desarrollo de Software.

Los sistemas automatizados de tratamiento de la información cada vez son mayores y más complejos. Ha surgido entonces la necesidad de establecer procesos de organización que ayuden y guíen el desarrollo de software. Es en este marco que surgen las metodologías para el desarrollo de software con sus características y especificidades. En un proyecto de desarrollo de software la metodología define Quién debe hacer Qué, Cuándo y Cómo debe hacerlo. No existe una metodología de software universal. Las características de cada proyecto (equipo de desarrollo, recursos, etc.) exigen que el proceso sea configurable.

Hoy en día existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Un ejemplo de ellas son las propuestas tradicionales centradas específicamente en el control del proceso. Estas han demostrado ser efectivas y necesarias en un gran número de proyectos, sobre todo aquellos proyectos de gran tamaño respecto a tiempo y recursos ya que se caracterizan por poseer grandes volúmenes de información, artefactos, planillas entre otros. Destacándose la Metodología “Proceso Unificado de Rational”. Esta metodología se caracteriza por ser:

- Dirigida por Casos de Uso
- Centrada en la Arquitectura
- Iterativa e Incremental

RUP se basa en la definición de diferentes roles y flujos de trabajos donde se desarrollan las actividades que arrojan los diferentes artefactos que deben realizar los trabajadores que se encuentran dentro de cada rol en las diferentes etapas guiados por etapas que conforman el ciclo de vida del software.

Sin embargo en muchas ocasiones las metodologías tradicionales como la mencionada anteriormente no ofrecen una buena solución para proyectos donde el entorno es volátil y donde los requisitos no se conocen con exactitud, porque no están pensadas para trabajar con incertidumbre. A modo de necesidad de solventar esos problemas surgen entonces las metodologías agiles las cuales se basan sobre todo en

la reacción ante los cambios que puede sufrir un proyecto. Entre estas metodologías agiles se destaca la metodología Programación Extrema (XP), especialmente diseñada para la resolución de problemas en el menor tiempo posible y con una alta probabilidad de cambio de requisitos, de ahí que exista una ausencia de énfasis en la arquitectura durante las primeras iteraciones y por tanto un déficit de métodos de diseño arquitectónico. Entre las metodologías agiles se encuentran:

- Xp.

Extreme Programming: Se caracteriza por poseer bajo riesgo, por ser flexible, formada por valores, principios y prácticas. Centrada en potenciar las relaciones entre cliente y equipo de desarrollo como clave para el éxito en desarrollo de software, es adecuada para proyectos con requisitos imprecisos y muy cambiantes, y donde existe un alto riesgo técnico.

- Scrum.

El desarrollo de software se realiza mediante iteraciones, denominadas sprints o carreras cortas, con una duración de 30 días. Antes de que comience una carrera se define la funcionalidad requerida para esa carrera y entonces se deja al equipo para que la entregue. El punto es estabilizar los requisitos durante la carrera. El resultado de cada sprint es un incremento ejecutable que se muestra al cliente. No es propiamente un método o metodología de desarrollo, e implantarlo como tal resulta insuficiente, Scrum define métodos de gestión y control para complementar la aplicación de otros métodos ágiles como XP o alguna otra metodología de desarrollo.

- Crystal

Las metodologías de Crystal se basan en el principio de que tipos diferentes de proyectos requieren tipos diferentes de metodologías. La metodología escogida debe depender de dos factores, el número de personas del proyecto y las consecuencias de los errores diferenciando con colores en dependencia de la cantidad de integrantes, por ejemplo Cristal Clear (3 a 8 personas), seguido por Yellow (10 a 20 personas), Crystal Orange (25 a 50 personas) y así consecutivamente hasta el Violet.

- DSDM – Dynamic Systems Development Method.

- FDD – Feature Driven Development.
- ASD – Adaptive Software Development.
- XBreed.
- Extreme Modeling.

Estas metodologías agiles todas van encaminadas a poder desarrollar el software en menos tiempo de que lo desarrollan las metodologías tradicionales o las metodologías pesadas. El resultado de todo esto es que los métodos ágiles cambian significativamente algunos de los énfasis de los métodos ingenieriles. La diferencia inmediata es que son menos orientados al documento, exigiendo una cantidad más pequeña de documentación para una tarea dada. Posee la desventaja de estar guiada por estándares propuestos por el equipo de desarrollo y no por estándares establecidos institucionalmente en el desarrollo de software. El cliente forma parte del equipo por lo que el mismo debe trabajar en el mismo sitio a diferencia de las tradicionales que cada miembro tiene bien definido su rol dentro del proceso y lo puede desarrollar con entradas y salidas de artefactos generados los cuales las metodologías agiles presentan pocos. Las metodologías ágiles son más difíciles de controlar y no poseen mucho énfasis en la arquitectura de software ni poseen contratos tradicionales o los que poseen son bastante flexibles.

1.7. Herramientas de Ingeniería de Software Asistidas por Computadora (CASE)

El modelado orientado o objetos sufrió un gran auge con la aparición de UML como estándar internacional acogido por la gran mayoría de los productores de software a nivel mundial como el sistema de notaciones para documentar, visualizar, especificar y construir los artefactos de los sistemas de un software utilizando conceptos del paradigma del modelado orientado a objetos. El objetivo principal de las herramientas CASE es automatizar el proceso de desarrollo del software a través del ciclo de vida del mismo. Entre las más utilizadas en la actualidad podemos encontrar:

1.7.1. Rational Rose Enterprise Edition

- Herramienta propietaria, perteneciente a la familia Rational Rose.
- Madura, bien establecida y con gran aceptación en el mercado.

- Incluye una Modelación Añadida de la Web que proporciona visualización, modelación y herramientas para el desarrollo de aplicaciones Web.

- Ofrece habilidades de análisis de calidad de código y generación del código, con capacidades de sincronización, así como manejo más granular y uso de modelos.

1.7.2. **Enterprise Architect**

- Interfaz de usuario intuitiva

- Documentación flexible y comprensible.

- Ingeniería de Código Directa e Inversa.

- Plugins para vincular EA a Visual Studio.NET o Eclipse

- Modelado de Base de Datos.

- Soporte de esquema XML.

- Soporte en Administración de Requisitos.

Soporta Línea Base, seguridad de usuario, prueba, mantenimiento, administración de proyecto, información de estado del sistema, control de versiones, entre otras. En sus versiones más actuales soporta ayuda comprensiva para UML 2.1 además de soportar la mayoría de los tipos de sus diagramas de modelado.

1.7.3. **Visual Paradigm for UML**

Visual Paradigm para UML es una herramienta UML profesional que soporta el ciclo de vida de desarrollo de software: análisis y diseño orientados a objetos, construcción, pruebas y despliegue. La herramienta también proporciona abundantes tutoriales de UML, demostraciones interactivas de UML y proyectos que utilicen UML y paradigma orientado a objetos. Entre sus principales características se encuentran:

- Ofrece entorno de creación de diagramas para UML 2.0, 2.1

- Es una herramienta multiplataforma
- Disponibilidad de integrarse en los principales IDEs de desarrollo
- Soporta diferentes lenguajes en la Generación de Código e Ingeniería Inversa como, Java, C++, CORBA IDL, PHP, Esquema de XML, Ada y Python.
- La Generación de Código soporta C #, VB .NET, Lenguaje de Definición de Objeto (ODL), Flash Action Script, Delphi, Perl, Objetivo-C, y Ruby.

1.8. Plataformas de desarrollo

Existen diversas plataformas de desarrollo idóneas para desarrollar software en dependencia de las diferentes características de la aplicación que se esté desarrollando. Algunas brindan mecanismos adecuados a utilizar en dependencia de la arquitectura de la aplicación a desarrollar, sistema operativo donde correrá la aplicación, entre otras muchas características. Entre las principales plataformas y además tomando en cuenta las características del subsistema de modelado de DFI, y las ventajas que estas brindan las plataformas de desarrollo candidatas son:

1.8.1. Plataforma .Net

Microsoft.NET es el conjunto de tecnologías en las que Microsoft ha estado trabajando durante años con el objetivo de obtener una plataforma sencilla y potente para distribuir el software de forma tal que puedan ser suministrados remotamente y que puedan comunicarse y combinarse unos con otros de manera totalmente independiente de la plataforma, lenguaje de programación y modelo de componentes con los que hayan sido desarrollados. La plataforma .NET ofrece un entorno gestionado de ejecución de aplicaciones, nuevos lenguajes de programación y compiladores, y permite el desarrollo de todo tipo de funcionalidades: desde programas de consola o servicios Windows, hasta aplicaciones para dispositivos móviles, pasando por desarrollos de escritorio o para Internet.

Para crear aplicaciones para la plataforma .NET, tanto servicios Web como aplicaciones tradicionales (aplicaciones de consola, aplicaciones de ventanas, servicios de Windows NT, etc.), Microsoft ha publicado el denominado kit de desarrollo de software conocido como .NET Framework SDK, que incluye

las herramientas necesarias tanto para su desarrollo como para su distribución y ejecución y Visual Studio.NET, que permite hacer todo la anterior desde una interfaz visual basada en ventanas.

.NET ofrece un entorno de ejecución para sus aplicaciones conocido como *Common Language Runtime* o CLR. La CLR es la implementación de Microsoft de un estándar llamado *Common Language Infrastructure* o CLI. Este fue creado y promovido por la propia Microsoft pero desde hace años es un estándar reconocido mundialmente.

El CLR/CLI esencialmente define un entorno de ejecución virtual independiente en el que trabajan las aplicaciones escritas con cualquier lenguaje .NET. Este entorno virtual se ocupa de elementos importantes para una aplicación: desde la gestión de la memoria y la vida de los objetos hasta la seguridad y la gestión de subprocesos.

Todos estos servicios unidos a su independencia respecto a arquitecturas computacionales convierten la CLR en una herramienta extraordinariamente útil puesto que, en teoría, cualquier aplicación escrita para funcionar según la CLI puede ejecutarse en cualquier tipo de arquitectura de hardware. Por ejemplo Microsoft dispone de implementación de .NET para Windows de 32 bits, Windows de 64 bits e incluso para Windows Mobile, cuyo hardware no tiene nada que ver con la arquitectura de un ordenador común. .Net dispone de varios lenguajes entre ellos los más conocidos son C#, J#, Visual Basic.Net entre otros pero lo más importante es que independientemente de en que lenguaje se desarrolle una aplicación o un componente de una aplicación podrá ser utilizado de manera transparente en cualquier otro lenguaje de .Net.

1.8.2. Plataforma Java

La plataforma Java es el nombre de un entorno o plataforma de computación originaria de Sun Microsystems, capaz de ejecutar aplicaciones desarrolladas usando el Lenguaje de programación Java u otros lenguajes que compilen a byte code y un conjunto de herramientas de desarrollo. En este caso, la plataforma no se ejecuta en un hardware o un sistema operativo específico, sino más bien una máquina virtual encargada de la ejecución de aplicaciones, y un conjunto de librerías estándar que ofrecen funcionalidad común. Java es una tecnología orientada al desarrollo de software con el cual podemos realizar aplicaciones y servicios Web, aplicaciones Cliente/Servidor, aplicaciones distribuidas, así como

aplicaciones de escritorio y aplicaciones para móviles. El corazón de la Plataforma Java es el concepto común de un procesador “virtual” que ejecuta programas. En concreto, ejecuta el código resultante de la compilación del código fuente, conocido como byte code. Este “procesador” es la máquina virtual de Java o JVM (Java Virtual Machine), que se encarga de traducir (interpretar o compilar) el byte code en instrucciones nativas de la plataforma destino. Esto permite que una misma aplicación Java pueda ser ejecutada en una gran variedad de sistemas con arquitecturas distintas, siempre con una implementación adecuada de la JVM. Este hecho es lo que ha dado lugar a la famosa frase: “write once, run anywhere” (escribir una vez, ejecutar en cualquier parte). La condición es que no se utilicen llamadas nativas o funciones específicas de un sistema operativo.

Actualmente la implementación de la máquina virtual incluye un compilador JIT (Just In Time). De esta forma, en vez de la tradicional interpretación del código byte code, que da lugar a una ejecución lenta de las aplicaciones, el JIT convierte el byte code a código nativo de la plataforma destino ganando en robustez de la compilación.

Java posee tres tipos de subestándares, donde cada uno es representa una tecnología disponible para desarrollar familias de aplicaciones.

Los subestándares principales de Java son:

- JSE (Java Standard Edition). Plataforma base del lenguaje Java donde se pueden desarrollar aplicaciones Stand Alone, tanto JEE y JME se basan en JSE. Es gratuito y de libre distribución. Dentro del JSE se incluyen el compilador y la JVM (Java Virtual Machine), que también se conoce como Intérprete de Java, Java Standard Edition, comprende además las librerías básicas del lenguaje, así como las APIs (Application Programming Interface) más comunes.

- JEE (Java Enterprise Edition). Plataforma del lenguaje Java que provee una especificación de cómo debe construirse una aplicación empresarial. Esta especificación describe como los servidores de aplicación deben proporcionar seguridad, escalabilidad, portabilidad, consistencia, manejo transaccional robusto e independencia de la plataforma tanto de hardware como de sistemas operativos, y al mismo tiempo para que el desarrollador final pueda desarrollar aplicaciones empresariales con menor esfuerzo.

- JME (Java Micro Edition). Plataforma del lenguaje Java destinada al desarrollo de aplicaciones para dispositivos móviles. JME es una versión reducida del conjunto de clases de JSE aunque con algunas librerías extra pensando sobre todo en el desarrollo de aplicaciones para dispositivos móviles. Se trata de una edición especial debido a las limitaciones de recursos en diversas familias de dispositivos (celulares, PDAs, Pockets, PCs, televisiones, relojes, sistemas de ayuda para automóviles, tarjetas, robótica).

Entre las ventajas principales de Java se encuentran:

- Licencias: Java es Open Source (Software libre o Código abierto). La infraestructura para desarrollar en JAVA es gratuita, aunque algunos proveedores de servidores de aplicaciones o Framework de desarrollo tienen costo.

- Multiplataforma: Representa un lenguaje independiente de la plataforma. Una aplicación Java puede correr sobre diversos ambientes sin modificaciones como Windows, Linux, AS400, entre otros.

- Interoperabilidad Multilenguaje: A partir de la versión 6 JAVA soporta aplicaciones con componentes en otros lenguajes.

1.8.3. Plataforma Mono

Mono es el nombre de un proyecto de código abierto para crear un grupo de herramientas libres, basadas en GNU/Linux y compatibles con .NET.

Los paquetes que componen la distribución de la plataforma Mono tienen un compilador C#, una máquina virtual (que permite ejecutar las aplicaciones), y un conjunto de librerías de clases que proporcionan funciones. Con Mono se pueden escribir aplicaciones en múltiples lenguajes de programación, incluyendo entre ellos Python, Object Pascal, Nermele, y C#. Una vez escritas las aplicaciones se traducen a CIL (Common Intermediate Language), que es un lenguaje intermedio que no tiene particularidades de ninguna arquitectura. Una vez compilado en CIL la aplicación se traduce al lenguaje específico de la arquitectura final donde será ejecutado. Este sistema permite distribuir un único programa binario para todas las arquitecturas en vez de un programa específico para cada plataforma. Pero no menos es

importante es la libertad de escoger cualquier lenguaje de programación o combinación de ellos y poder ejecutar la aplicación en cualquiera de las plataformas en las que Mono se encuentra disponible, entre las que se incluyen Intel, AMD64, SPARC, StrongArm y S390x. Mono actualmente proporciona las herramientas para crear aplicaciones para Linux (diversas distribuciones), Solaris, Windows, Mac/OS, y mainframes de IBM. A diferencia de los programas tradicionales que se ejecutan sobre el sistema directamente, los programas en la plataforma Mono se ejecutan sobre un entorno controlado de ejecución conocido como la máquina virtual.

Mono proporciona las funciones necesarias para crear servicios web, esto incluye las tecnologías XML, SOAP, ASP.NET, y Remoting así como el acceso a bases de datos, a través ADO.NET, de tipo Oracle, MySQL, DB2, SQL Server, o Progress. Con estas librerías se pueden portar servicios desarrollados en Windows o escribir servicios propios utilizando Apache como servidor web. Para el desarrollo de interfaces de usuario para aplicaciones cliente las opciones son múltiples. Por un lado proporciona una implementación en código abierto de System.Windows.Forms (que es la opción que proporciona Microsoft) para que los desarrollos realizados en plataforma .Net sobre Windows puedan ser ejecutados en otras plataformas como Linux o Mac OS. También proporciona GTK# una librería que expone toda la funcionalidad del entorno gráfico de Gnome y que permite realizar aplicaciones multiplataforma.

1.9. Lenguajes de Programación

A partir de las especificidades de la aplicación a desarrollar y tomando en cuenta las ventajas y desventajas de los lenguajes de programación, la coordinación e interoperabilidad entre módulos o componentes desarrollados en diferentes lenguajes de programación y las plataformas de desarrollo tomadas en cuenta con posibilidad a utilizarse en el desarrollo de subsistema de modelado de DFI, se estima que los lenguajes candidatos a utilizar en el desarrollo de la aplicación son:

1.9.1. C#

A continuación se enumeran las principales características que definen al lenguaje de programación C#. Algunas de las cuales no son propias del lenguaje, sino de la plataforma .NET, aunque se listan aquí debido a que tienen una implicación directa en dicho lenguaje.

- Sencillez de uso: C# elimina muchos elementos añadidos por otros lenguajes lo que facilita su uso y compresión. Es por ello que se dice que C# es auto contenido. Además, no se incorporan al lenguaje elementos poco útiles, como por ejemplo macros, herencia múltiple u operadores diferentes al operador de acceso a métodos (operador punto) para acceder a miembros de espacios de nombres.

- Modernidad: Incorpora elementos que se ha demostrado a lo largo del tiempo que son muy útiles, como tipos decimales o booleanos, un tipo básico string, así como una instrucción que permita recorrer colecciones con facilidad (instrucción foreach).

- Orientado a objetos: C# no permite la inclusión de funciones ni variables globales que no estén incluidos en una definición de tipos, por lo que la orientación a objetos es más pura y clara que en otros lenguajes como C++. Soporta abstracción, herencia y polimorfismo.

- Orientado a componentes: La propia sintaxis de C# incluye elementos propios del diseño de componentes por ejemplo formas de definir propiedades, eventos o atributos.

- Recolección de basura: Todo lenguaje incluido en la plataforma .NET tiene a su disposición el recolector de basura del CLR. No es necesario incluir instrucciones de destrucción de objetos en el lenguaje.

- Seguridad de tipos: C# incluye mecanismos de control de acceso a tipos de datos, lo que garantiza que no se produzcan errores difíciles de detectar. Para ello, el lenguaje provee de una serie de normas de sintaxis, como por ejemplo no realizar conversiones entre tipos que no sean compatibles. Además, no se pueden usar variables no inicializadas previamente, y en el acceso a tablas se hace una comprobación de rangos para que no se excedan los índices de la misma. Se controla así mismo los desbordamientos en operaciones aritméticas, produciéndose excepciones cuando se produzcan.

- Instrucciones seguras: Se han impuesto una serie de restricciones en el uso de instrucciones de control más comunes. Por ejemplo, la evaluación de toda condición ha de ser una expresión condicional y no aritmética. Así se evitan errores por confusión del operador igualdad con el de asignación. Otra restricción que se impone en la instrucción de selección switch, imponiendo que toda

selección de la instrucción finalice con una instrucción break o goto que indique cuál es la siguiente acción a realizar.

- Unificación de tipos: En C# todos los tipos derivan de una superclase común llamada System. Object. A diferencia de Java, en C# esta característica también se aplica para los tipos básicos.

- Extensión de los operadores básicos: Para facilitar la legibilidad de código y conseguir que los nuevos tipos de datos que se definan a través de las estructuras estén al mismo nivel que los elementos predefinidos en el lenguaje, al igual que C++ pero a diferencia de Java, C# permite redefinir el significado de la mayoría de los operadores cuando se apliquen a diferentes tipos de objetos. Las redefiniciones de operadores se hacen de manera inteligente, de modo que a partir de una única definición de los operadores (++) y (--) el compilador puede deducir automáticamente cómo ejecutarlos de manera prefija y postfija. Además, para asegurar la consistencia, el compilador exige que los operadores con opuesto (como por ejemplo el operador igualdad == y su opuesto !=) siempre se redefinan por parejas. También se da la posibilidad, a través del concepto de indexadores, de redefinir el significado del operador corchetes [] para los tipos de dato definidos por el usuario, con lo que se consigue que se pueda acceder al mismo como si fuese una tabla.

- Extensión de modificadores: C# ofrece, a través de los atributos, la posibilidad de añadir a los metadatos del módulo resultante de la compilación de cualquier fuente información adicional a la generada por el compilador, que luego podrá ser consultada en tiempo de ejecución a través de la biblioteca de reflexión de .NET.

- Eficiente: Por cuestiones de seguridad no permite el uso de punteros. Sin embargo, existen modificadores para saltarse esta restricción. Para ello basta identificar regiones de código con el identificador unsafe (inseguro), y podrán usarse en ellas punteros de forma similar a como se hace en C++. Característica útil en situaciones en las que se necesite gran velocidad de procesamiento.

- Compatible: Para facilitar la migración de programadores de C++ o Java a C#, no sólo se mantiene una sintaxis muy similar a la de los dos anteriores lenguajes, sino que el CLR también ofrece la posibilidad de acceder a código nativo escrito como funciones sueltas no orientadas a objetos, tales como las DLLs de la API de Win32.

Además de estas características existen otras más actuales como las de C# 3 (Visual Studio.Net 2008 con el Framework 3.5) que posee algunos aspectos adicionales al C# 2 (Visual Studio.Net 2005 con el Framework 2.0). Algunas de esas características forman parte del trabajo realizado para el Windows Vista de Microsoft, pero hay muchas otras de igual o mayor interés para los desarrolladores de software como son:

Variables sin tipo, tipos anónimos e inicializadores de objeto: Es una característica de C# 3 que permite crear una variable sin especificar el tipo de dato que almacenará. Entonces es el lenguaje el que determina el tipo real de los datos que la variable podrá contener. Es utilizada en los momentos en que el desarrollador no tiene a priori el conocimiento del tipo de dato que almacenará. El compilador decide en tiempo de ejecución el tipo de dato de la variable.

Los tipos o clases anónimas, por definición, no tienen nombre. La verdadera importancia de estos "tipos anónimos" aparece cuando consideramos el LINQ, Language Integrated Queries, ya que dependiendo de la consulta que se realice, obtendremos unos datos u otros, de forma que C# crea una clase anónima con los campos devueltos por la consulta LINQ para que se pueda acceder a ellos fácil y rápidamente. Además las clases anónimas se inicializan sin necesidad de un constructor lo que representa otra de las características de C# 3. Esto es lo que se conoce como inicializadores de objeto, y básicamente permiten asignar el valor que queramos a un objeto de cualquier tipo que haya sido creado utilizando el constructor por defecto. Otro de los aspectos de interés en el lenguaje son las Expresiones Lambda. En C#3 las funciones anónimas se escriben con el operador "=>", de forma que lo que queda a la izquierda del operador son los parámetros, y lo que queda a la derecha es el cuerpo de la función anónima, estas funciones anónimas se denominan Expresiones Lambdas.

1.9.2. Java

Java es un lenguaje potente que posee entre sus características principales:

- Orientado a objetos: Java fue diseñado como un lenguaje orientado a objetos desde el principio. Los objetos agrupan en estructuras encapsuladas tanto sus datos como los métodos (o funciones) que manipulan esos datos.

- Distribuido: Java proporciona una colección de clases para su uso en aplicaciones de red, que permiten abrir sockets y establecer y aceptar conexiones con servidores o clientes remotos, facilitando así la creación de aplicaciones distribuidas.

- Interpretado y compilado a la vez: Java es compilado, en la medida en que su código fuente se transforma en una especie de código máquina, los byte codes, semejantes a las instrucciones de ensamblador. Por otra parte, es interpretado, ya que los byte codes se pueden ejecutar directamente sobre cualquier máquina a la cual se hayan portado el intérprete y el sistema de ejecución en tiempo real (run-time).

- Robusto: Proporciona numerosas comprobaciones en compilación y en tiempo de ejecución. Se ha prescindido por completo de los punteros y sus complicaciones, y la recolección de basura elimina la necesidad de liberación explícita de memoria.

- Seguro: Java implementa barreras de seguridad en el lenguaje y en el sistema de ejecución en tiempo real.

- Indiferente a la arquitectura: Soporta aplicaciones para ser ejecutadas en diferentes redes, desde Unix a Windows NT, pasando por Mac y estaciones de trabajo, sobre arquitecturas distintas y con sistemas operativos diversos. El compilador de Java genera byte codes: un formato intermedio indiferente a la arquitectura, diseñado para transportar el código eficientemente a múltiples plataformas hardware y software. El resto de problemas los soluciona el intérprete de Java.

- Portable: La indiferencia a la arquitectura representa sólo una parte de su portabilidad. Además, Java especifica los tamaños de sus tipos de datos básicos y el comportamiento de sus operadores aritméticos, de manera que los programas son iguales en todas las plataformas.

- Multihebra: Java soporta sincronización de múltiples hilos de ejecución (multithreading) a nivel de lenguaje, especialmente útiles en la creación de aplicaciones de red distribuidas.

- Dinámico: El lenguaje Java y su sistema de ejecución en tiempo real son dinámicos en la fase de enlazado. Las clases sólo se enlazan a medida que son necesitadas. Se pueden enlazar nuevos módulos de código bajo demanda, procedente de fuentes muy variadas, incluso desde la Red.

- Curva de aprendizaje: Es lenguaje de programación un poco más complejo que C#, de ahí que a los desarrolladores experimentados en C++ les sea más sencillo emigrar a java.

1.10. UML Como lenguaje de modelado

UML, Lenguaje Unificado de Modelado: se define como un "lenguaje que permite especificar, visualizar y construir los artefactos de los sistemas de software..." Destinado al modelado de sistemas que utilizan conceptos orientados a objetos [5]. El UML constituye un estándar para construir modelos orientados a objetos. Es un lenguaje de modelado independiente del lenguaje de programación que se utilice así como de la plataforma en la que se desarrolla la aplicación. UML está definido como un ADLs pero a pesar de no ser según diferentes arquitectos guiados por la tendencia de la academia de arquitectura de software un lenguaje ideal para describir la arquitectura de software, hoy es el más utilizado y difundido para realizar dicha descripción. UML documenta toda la información estática y dinámica de un sistema, su distribución, estructura y comportamiento, de ahí que logre que los implicados tengan un entendimiento común independiente del código de la aplicación que se desee implementar. Este lenguaje esta compuesto por tres elementos fundamentales, los elementos, los diagramas y las relaciones. UML ha sido seleccionado como lenguaje de modelado del subsistema de modelado de DFI debido a que es una forma de documentar el desarrollo del subsistema, además es un lenguaje sencillo, visual, fácil de utilizar y comprender, que omite detalles específicos y permite la comprensión necesitada para el desarrollo de un sistema en concreto. En otras palabras simplifica la complejidad real, permite la reutilización de soluciones o de modelos, permite descubrir fallas y ahorra tiempo de desarrollo.

1.11. Selección de la Metodología de Desarrollo de Software

Tomando en cuenta las ventajas de RUP ya mencionadas en con anterioridad (ver epígrafe 1.6) y reparando además en que haciendo uso de las 4 +1 vistas (lógica, despliegue, implementación, proceso, casos de usos) y las principales características de RUP existe la posibilidad de las seleccionar los CU que se denominan como críticos en cada una de las iteraciones y definiendo actividades, métodos y artefactos a generar por cada uno de los roles se puede ir implementando el producto final de una manera controlada y organizada, de ahí que se estimó seleccionar el Proceso Unificado de Rational como la metodología de desarrollo del subsistema de modelado de DFI. Están bien definidas en RUP cada una de las actividades que debe realizar cada trabajador, RUP define seis flujos de trabajo de ingeniería de

software y tres de apoyo que trabajan más en la gestión de configuración del software y la administración del proyecto así como el mantenimiento del mismo estos flujos se desarrollan a través de cuatro etapas fundamentales (inicio, elaboración, construcción y transición), donde cada una posee un hito significativo para el desarrollo del software. Se valoró también que RUP permite generan todos los modelos como principales artefactos en cada uno de los flujos de trabajo garantizando además una arquitectura robusta desde las primeras etapas del ciclo de vida a partir de la línea base de la arquitectura, manteniendo una interacción directa entre CU y la arquitectura de la aplicación. Los resultados arrojados finalmente son las 4+1 vistas que describen la arquitectura:

Vista Lógica: Contiene los elementos del diseño más significativos para la arquitectura.

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: Suministra una base para la comprensión de la distribución física de un sistema a través de nodos. 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: Describe la descomposición del software en capas y subsistemas 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. Estas vistas se van conformando en de las iteraciones del software a través las fases de RUP.

A continuación se ilustra las fases y flujos de trabajo que propone la metodología RUP.

Figura 3: Fases y Flujos de trabajo propuestos por la metodología RUP.

1.12. Selección de la Plataforma y Lenguaje de Desarrollo

Tomando en cuenta las ventajas brindadas por la plataforma .Net (ver epígrafe 1.8), la actual tecnología disponible para el desarrollo de la aplicación, así como facilidad de uso de su principal Entorno Integrado de Desarrollo Visual Studio .Net 2005, se estimó utilizar esta combinación, lenguaje-plataforma-IDE para el desarrollo del Subsistema de modelado de DFI, por representar la variante más adecuada, fundamentalmente porque las otras opciones no se ajustaban a las características de hardware existente y disponible actualmente para la implementación de la aplicación. Si bien la plataforma Java en conjunto con el lenguaje de programación Java representaban una opción adecuada dado que permiten un fuerte tratamiento de imágenes a partir de varias bibliotecas de clases que posee este lenguaje para este tipo de trabajo, sería bajo en rendimiento del hardware disponible y las limitaciones de la tecnologías se convertirían en una barrera significativa para el desarrollo de la aplicación. Se estimó utilizar el IDE Visual

Studio 2005 aprovechando al máximo las ventajas del lenguaje C# y las librerías de clases que posee .Net para el tratamiento de imágenes.

1.13. Selección de de Estilos y Patrones Arquitectónicos

En la mayor parte de la bibliografía consultada el término de estilo o patrón arquitectónico se utiliza indistintamente y quizás en alguna de las diferentes tendencias o clases que agrupan los tipos de arquitectura sea más significativo este hecho de marcar las diferencias que además solo radican fundamentalmente en el nivel de abstracción. En la práctica dentro la metodología RUP y la corriente de la arquitectura de software como etapa de la ingeniería y el diseño orientado a objetos la utilización del término estilo arquitectónico o patrón arquitectónico indistintamente no sobrepone relevancias, de ahí que los patrones y estilos arquitectónicos adecuados a utilizar en el desarrollo de la aplicación sean:

1.13.1. Estilo de llamadas y Retornos: Arquitectura Orientada a Objetos

Desde el punto de vista de que la arquitectura de la aplicación estará basada en los principios de la POO, Abstracción, Herencia y Polimorfismo. Los principales elementos de la arquitectura estarán centrados hacia los objetos y serán estos las unidades de modelado a partir de las clases que los definen interactuando a través de funciones y métodos.

1.13.2. Estilo Flujo de Datos: Tuberías y Filtros

En el desarrollo de la aplicación se tomará como punto de partidas la idea de utilizar una arquitectura orientada a objetos donde los objetos serán el centro de los modelos en diversas ocasiones estos elementos deberán pasar por diferentes etapas, llamadas de rutinas eventos o procedimientos que adicionaran datos a la salida de este objeto y luego pasará a una siguiente etapa donde será nuevamente modificado al estilo de tuberías por donde se filtra la información y se modifican los objetos. El estilo arquitectónico ha utilizar mayormente debe ser el estilo Tuberías y Filtros.

1.14. Selección de Herramientas CASE

Visual Paradigm for UML

Para la selección de Visual Paradigm for UML como la principal herramienta CASE (Computer Aided Process Engineering) a utilizar en desarrollo del subsistema de modelado de DFI se tomó en cuenta las

ventajas mencionadas (ver epígrafe 1.7) que posee dicha herramienta para el modelo de sistemas además brinda facilidades al usuario a partir de una interfaz amigable y fácil de utilizar. Una razón determinante en la selección de dicha herramienta lo constituyo la necesidad de un modelado rápido de la aplicación a desarrollar, aunque no es necesaria una entrega del producto en un plazo corto es bueno ganar en agilidad en cuanto al modelado dada la complejidad del subsistema a desarrollar. Además se tomó en cuenta que Visual Paradigm for UML permite generar código para C# a pesar de no poseer una integración directa con la herramienta Visual Studio.Net 2005.

1.15. Selección de Patrones de Diseño

Además de utilizar patrones muy conocidos debido a su abundancia en la mayoría de las aplicaciones como son los patrones Creador, Controlador, Experto, Bajo Acoplamiento y Alta Cohesión, se utilizan otros más específicos acordes a la aplicación que se desea desarrollar, como son:

Observer

Se utiliza para poder notificar a determinados objetos denominados subscriptores cuando otro ha sufrido un cambio de interés para estos. Permite de forma dinámica implementar dependencias entre objetos, de forma que los objetos dependientes sean notificados de los cambios que se producen en los objetos de los que dependen. En la aplicación serán utilizados para actualizar determinados elementos que dependen directamente de los datos de otros y este patrón será utilizado explícitamente para realizar este tipo de actualización. Específicamente para actualizar los datos de la GUI ante un cambio en los datos.

Singleton

El Singleton es quizás el más sencillo de los patrones que se presentan en el catálogo de los patrones GoF. Es también uno de los patrones más conocidos y utilizados. Su propósito es asegurar que sólo exista una instancia de una clase. Cuando varios elementos distintos precisan referenciar a un mismo elemento y se desea asegurar que no hay más de una instancia de ese elemento, simplemente el patrón Singleton determina un punto de acceso global a esta clase o instancia garantizando resolver el problema, este es un patrón que a menudo se utiliza en el desarrollo de aplicaciones.

Patrón Extended Handlers

La arquitectura del subsistema de modelado de DFI estará enfocada hacia el estilo arquitectónico llamadas y retornos utilizando además una arquitectura orientada a objetos unida a los paradigmas de la POO. Se utilizará dentro de la programación el objeto evento, que permite suministrar objetos de este tipo con toda la información que este pueda poseer. La representación más abstracta de este patrón de eventos se puede observar en la siguiente figura:

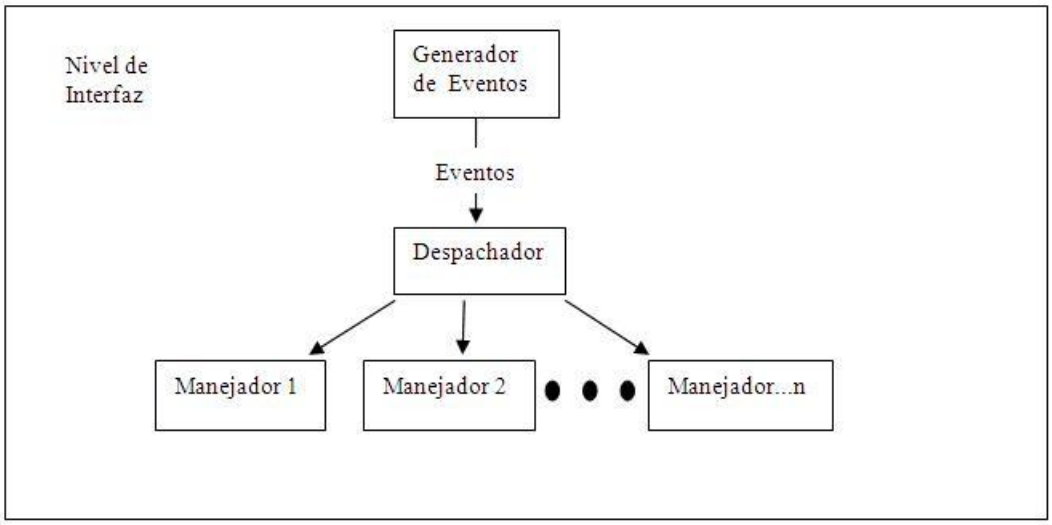

Figura 4: Patrón Extended Handlers, fuente [6] y propia.

Dentro de la distribución anterior los componentes interactúan mediante el anuncio y la notificación asíncrona o síncrona de eventos generados en su entorno y causan la reacción de otros componentes como consecuencia de sus anuncios unido al patrón Publisher/Subscribers que no es más que el patrón Observer aplicado a eventos en dependencia del tipo de los mismos. Los subscriptores registran su interés en determinados tipos de eventos y los que los conectan son precisamente los encargados de la gestión de eventos que determinan en dependencia del evento lanzado, que deberían realizar con el mismo. Observan el anuncio de eventos y generan las notificaciones correspondientes a los interesados en el mismo. El patrón Observer se encarga entonces de la difusión de los diferentes eventos a sus interesados. [6]

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.

El patrón memento define tres roles diferentes:

Originator: No es más que el objeto al cual se desean salvar sus datos.

Memento: No es más que otro objeto que contiene los mismos datos que el Originator

Caretaker: Se encarga de la gestión en el momento de salvar los datos, es decir salva el Memento y si es necesario utiliza también el Memento para restaurar al estado inicial de la fuente.[7]

La representación de las clases de los patrones antes mencionados integrados en un ejemplo del libro "Introducción al diseño de patrones en C#" y las clases quedaban como se ilustra en la figura 2 donde se vincula también otro patrón de diseño conocido como Comando.

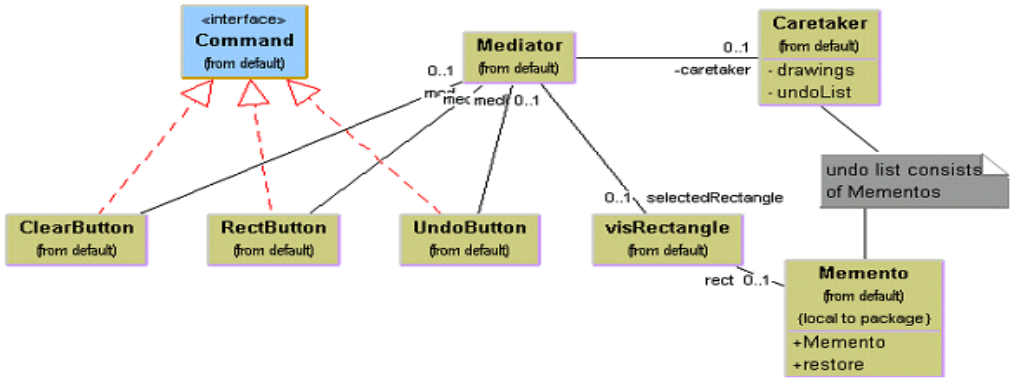

Figura 5: Patrón Memento. fuente [7].

1.16. Arquitectura de Simuladores

Al realizar un acercamiento teórico a la arquitectura de simuladores hay que basarse en el proyecto CAPE-OPEN y en la arquitectura u organización de sus elementos como Simulador Integrador de simuladores planteando la estructura de cualquier simulador moderno. El proyecto de este simulador estandariza los componentes a gran escala de un simulador presentando una división bien estructurada de cada elemento, la relación entre los mismos y la posibilidad de agregar nuevos componentes. Por arquitectura se entiende el reparto de funciones en diferentes módulos o aplicaciones de lo que constituye

el software y el tránsito de información entre ellos o comunicaciones. La arquitectura está más relacionada con la concepción, prestaciones y desarrollo del software que con la funcionalidad final del simulador. Obviamente arquitectura y funcionalidad están íntimamente ligadas y la primera se orienta primeramente dependiendo de la segunda, pero la arquitectura tiene otras condicionantes y objetivos. El factor más importante a considerar en el diseño de una arquitectura es su grado de concentración o, inversamente, su grado de distribución. Entendemos por aplicación el módulo de software que puede funcionar autónomamente en un solo ordenador. Una arquitectura concentrada es aquella en la que hay una o pocas aplicaciones que concentran cada una un número elevado de funcionalidades, funcionan en uno o pocos ordenadores con poco flujo de comunicación exterior permitiendo desarrollos simples y de muy corto alcance pero quedándose corta en simuladores complejos. Una arquitectura distribuida tiene varias o muchas aplicaciones funcionando en varios ordenadores con un flujo importante de comunicaciones y tiene como fundamental objetivo obtener prestaciones razonables a un coste bajo, permitiendo mayor la reutilización de código que la concentrada y además evita la carga del procesos en cálculos de los modelos matemáticos. Sin embargo independientemente del tipo de arquitectura que utilice un determinado simulador existen una serie de condiciones que debe cumplir la arquitectura como son satisfacer todas las funcionalidades del simulador, optimizar el uso del hardware aprovechando la capacidad del software referente al lenguaje de programación y las características intrínsecas del sistema operativo. Además de facilitar el flujo de información para la simulación en tiempo real: lo que constituye una de las más difíciles tareas de una arquitectura. Requiere un gran esfuerzo minimizar el traspaso de datos entre las diferentes partes del software sin que ninguna tenga déficit o exceso, y no surjan problemas de sincronismo y saturación del hardware de comunicaciones.

1.17. Caracterización de las Interfaces de simuladores de procesos.

Podemos definir una interfaz de usuario como el conjunto de elementos (hardware y software) que actúan como frontera entre el dominio del usuario final y el de la máquina o sistema que posee además la misión de lograr y facilitar la comunicación entre ambos. La interfaz incluye las pantallas, ventanas, controles, menús, la ayuda en línea, la documentación y el entrenamiento o formación del usuario, siempre garantizando que el usuario se sienta organizado, que pueda realizar cómodo el trabajo a través de una interfaz sencilla y fácil de utilizar, previendo que no se convierta la misma en problema adicional para el usuario además de los que intenta resolver en un determinado sistema**.**

La interfaces de simuladores de procesos permiten la mayoría de las opciones que proporcionen los mismos y las diferencias entre estas vienen dadas mayormente en la facilidad de uso y sencillez que brindan para realizar las operaciones. Se encuentran orientadas a un carácter funcional u operativo es decir están basadas en las funcionalidades y no siempre en las representaciones y técnicas que logren un mejor entendimiento del usuario. Existen en el mundo simuladores con interfaces avanzadas ya que tiene la capacidad de brindar diversas opciones al usuario. Consecuentemente a esto las grandes empresas desarrolladoras de simuladores en el mercado han incrementado la competencia y las interfaces cada vez son más orientadas al usuario, de ahí que existan simuladores conocidos como de realidad virtual clasificados así a partir de que la interacción con el usuario se realiza a partir de una interfaz avanzada e inteligente que simula ampliamente la realidad. La mayoría de estas aplicaciones son videojuegos o simuladores de carácter académicos con entornos parecidos a los videojuegos. En otras ramas como en los simuladores químicos y en los simuladores médicos, las representaciones 3D junto a la realidad virtual constituyen una gran opción para orientar las interfaces a la comodidad que los usuarios necesitan.

Ya tratándose de interfaces en términos arquitectónicos y estructuras, relaciones entre clases y componentes, que es a lo que mayormente refieren la “banda de los cuarto” al hablar de arquitectura de software, el futuro de la simulación de procesos y de la estructura en general de un simulador está marcada en la actualidad por el proyecto CAPE-OPEN que intenta establecer estándares informáticos que permitan la interoperabilidad de simuladores de procesos y de modelos especializados para operaciones unitarias, paquetes termodinámicos y paquetes numéricos. Las empresas adelantadas en el mercado de la simulación de procesos son AspenTech (Hyprotech) y Simulation Sciences. Cada una de ellas ha desarrollado sus simuladores de procesos, utilizando tecnologías informáticas muy diversas, que comprenden desde programas tradicionales escritos en FORTRAN hasta aplicaciones diseñadas con tecnología orientada a objetos (por ejemplo, C ++, C#), siempre respetando la mayoría de los estándares acogidos a nivel mundial e implantados por el proyecto europeo CAPE-OPEN y su estructura propuesta. En su mayoría los simuladores de estas compañías están sujetos al pago de las respectivas licencias de uso, que no siempre está al alcance de las instituciones universitarias y comerciales en países en vías de desarrollo por lo que la simulación se extiende.

1.18. Conceptos del Dominio

Diagrama de Flujo

El Diagrama de Flujo es una representación gráfica de la secuencia de pasos que se realizan para obtener un cierto resultado. Este puede ser un producto, un servicio, o bien una combinación de ambos. Un diagrama de flujo debe representar fielmente el proceso real que representa, está compuesto por símbolos que deber ser intuitivos, claros en lo que desean comunicar. Debe además poseer una interpretación para cada uno de sus símbolos existiendo estándares en el mundo acerca de los mismos.

Diagrama de Flujo de Procesos

En una simulación de procesos industriales a partir de la información típica del proceso como la que se genera desde un diagrama de flujo, se obtiene el diagrama de simulación, el cuál es básicamente igual al de proceso, pero en él aparecen los equipos virtuales, tales como mezcladores y divisores de corrientes. En los diagramas de procesos solo aparecen las corrientes de datos es solo una representación que da paso luego al diagrama de simulación.

Diagrama de Flujo de Información

Ya teniendo los datos entonces se concluyen los diagramas de una simulación con el DFI que no es más que un conjunto de .nodos que contienen el orden de la información a partir de identificadores a los procesos reales de la simulación. Existen dos tipos de DFI con ciclos repetidos o sin estos, en dependencia del tipo entonces llevará una resolución diferente.

1.19. Resumen

En el capítulo se han abordado las principales temáticas que permiten comprender en que basa la descripción de una arquitectura de software dentro del desarrollo de software a nivel mundial. Además se describió brevemente las metodologías de desarrollo como proceso organizativo de la producción de software así como de Las herramientas, lenguajes de programación e IDEs de desarrollo y fundamentales adecuados con las características que posee la aplicación a desarrollar. Se pudo de esa manera seleccionar las estrategias y estándares a partir de los que se regiría la presente investigación en el desarrollo del Subsistema de modelado de DFI apoyándose en la situación en cuanto a tecnología disponible, recursos humanos y necesidades del cliente en general. Se puede llegar a la conclusión de que la arquitectura es un eslabón fundamental en el desarrollo de aplicaciones y es el aspecto que regirá el futuro desempeño de la aplicación que se modele y desarrolle en consecuencia de la selección

acertada de una línea base de la arquitectura y sus componentes en cuanto estilos y patrones. El presente capítulo constituye un acercamiento teórico a los tópicos relacionados con el desarrollo de software, con la arquitectura de software y la arquitectura de simuladores, así como las principales características que debe poseer una poseer una interfaz de cualquier simulador de procesos en general para cumplir con las expectativas del usuario final.

2. Capítulo 2: Descripción de la Arquitectura

Introducción

En el presente capítulo se abordará la descripción de la arquitectura desde el punto de vista práctico, se detallarán los aspectos significativos para la arquitectura del subsistema que desea desarrollar. Para cumplir los objetivos de la presente investigación es necesario proponer una arquitectura que permita el cumplimiento de las funciones básicas de la aplicación a desarrollar. En este capítulo se propondrá dicha arquitectura así como la organización de todos los componentes que forman parte de la misma, fundamentalmente las 4+1 vistas arquitectónicas.

Descripción de la Arquitectura

2.1. Propósito

En el presente documento se ofrece una visión de la arquitectura del subsistema a desarrollar con el objetivo de representar las principales decisiones que han girado alrededor de los aspectos significativos del subsistema de modelado de Diagramas de Flujo de Información. Se brindará en el presente capítulo la explicación detallada de cada una de las vistas arquitectónicas de la aplicación a desarrollar con el fin de lograr un correcto entendimiento del equipo de desarrollo de las mismas.

2.2. Alcance

El presente documento contiene los detalles de la arquitectura del Subsistema de Modelado de Diagramas de Flujo de Información que se encuentra en actual desarrollo por el equipo de trabajo del Polo Simulación de la Facultad 9 de la Universidad de las Ciencias Informáticas en Ciudad de la Habana, Cuba. En el mismo están contenidas las principales decisiones tomadas con respecto al subsistema a desarrollar en cuanto a la arquitectura de software. Se explica la utilización de los estilos arquitectónicos así como patrones arquitectónicos y de diseño que han ayudado a resolver diferentes problemas surgidos en el desarrollo de la arquitectura del Subsistema de Modelado de DFI.

2.3. Definiciones, Acrónimos, y Abreviaciones

Corriente: Tipo de módulo que puede servir de entrada de datos o salida de los mismos. En los simuladores de procesos industriales sirven para la entrada del flujo de información.

Conexión: Relación establecida entre un Módulo y otro o con una Corriente.

Puerto: Punto de Conexión entre dos Módulos o entre un Módulo con una Corriente.

Etiqueta: Constituye una nota dentro del diagrama que se puede asociar o no a un módulo o corriente determinada.

CU: Casos de Uso.

DFI: Diagrama de Flujo de Información.

POE: Programación Orientada a Eventos.

2.4. Referencias

Las referencias aplicables son:

1. Plantilla de Requerimientos, Especificación de Requisitos. Simulación de Procesos Industriales 0.1, 2009 Martínez J. P.

2.5. Representación de la Arquitectura

Este documento representa la Arquitectura de Software desde el punto de vista de la metodología RUP basada en diferentes vistas que juntan los elementos más significativos del desarrollo de un sistema. Las vistas mencionadas están organizadas en el documento en el siguiente orden:

- 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 básicas del sistema. Contiene los CU significativos para la arquitectura que son aquellos CU que dan respuesta directa a requisitos funcionales y no funcionales que condicionan la arquitectura del sistema a desarrollar en dependencia de su impacto dentro del mismo.
- Vista Lógica: Describe un subconjunto del modelo de diseño representa elementos que son arquitectónicamente significativos. Contiene las clases significativas con su explicación detallada la descomposición de las mismas en subsistemas y paquetes y las relaciones que se establecen entre ellas.

- 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 componentes y subsistemas de implementación. También provee una vista de la trazabilidad de los elementos de diseño de la vista lógica para la implementación. Contiene entre otros:
    - Subsistemas de Implementación.
    - Diagramas de componentes que ilustran las jerarquías de los subsistemas.
    - Dependencia entre subsistemas.

2.6. Metas y Limitaciones de la Arquitectura

2.6.1. Requerimientos Funcionales:

F:1) Funcionalidad Gestionar Módulo

RF:1. Dibujar Módulo.
RF:2. Seleccionar Módulo.
RF:3. Mover Módulo.
RF:4. Ocultar Módulo.
RF:5. Eliminar Módulo.
RF:6. Redimensionar Módulo.
RF:7. Rotar Módulo.
RF:8. Hacer espejo a Módulo.
RF:9. Traer Módulo al frente
RF:10. Enviar Módulo al fondo
RF:11. Mostrar Módulo

F:2) Funcionalidad Gestionar Corriente

RF:12. Dibujar Corriente.

RF:13. Seleccionar Corriente.

RF:14. Mover Corriente.

RF:15. Ocultar Corriente.

RF:16. Eliminar Corriente.

RF:17. Redimensionar Corriente.

RF:18. Rotar Corriente.

RF:19. Hacer espejo a Corriente.

RF:20. Traer Corriente al frente

RF:21. Enviar Corriente al fondo

RF:22. Mostrar Corriente

F:3) Funcionalidad Gestionar Etiqueta

RF:23. Seleccionar Etiqueta

RF:24. Mover Etiqueta

RF:25. Eliminar Etiqueta

RF:26. Modificar Etiqueta

RF:27. Asociar Etiqueta a Componente

F:4) Funcionalidad Gestionar Conexión

RF:28. Conectar corriente-módulo

RF:29. Conectar módulo-módulo

RF:30. Mover conexión

RF:31. Seleccionar conexión

RF:32. Eliminar conexión

F:5) Funcionalidad Gestionar Capas

RF:33. Agrupar en Capa.

RF:34. Desagrupar Capa.

RF:35. Seleccionar Capa.

RF:36. Bloquear Capa.

RF:37. Ocultar Capa.

RF:38. Editar Capa.

RF:39. Mover Capa.

RF:40. Eliminar Capa.

RF:41. Mostrar Capa.

F:6) Funcionalidad Gestionar Área de Sensibilidad

RF:42. Mostrar Área de Sensibilidad de un Módulo

RF:43. Mostrar Área de Sensibilidad de una Corriente

RF:44. Mostrar Área de Sensibilidad de una Conexión

F:7) Funcionalidad Gestionar DFI

RF:45. Cambiar porcentaje de visualización del DFI.

RF:46. Permitir Vista en Miniatura del Diagrama DFI.

RF:47. Realizar Operación Deshacer (Undo).

RF:48. Realizar Operación Rehacer (Redo).

RF:49. Salvar el DFI.

RF:50. Realizar zoom al Diagrama DFI.

RF:51. Cargar el DFI.

RF:52. Actualizar el DFI.

RF:53. Copiar componentes del DFI.

RF:54. Pegar componentes del DFI.

RF:55. Cortar componentes del DFI.

2.6.2. Requerimientos no Funcionales:

Requerimientos de software

RNF: 1 El subsistema debe correr sobre la plataforma Windows XP.

RNF:2 Se requiere de la instalación del Framework .NET 2.0.

Requerimientos de hardware

RNF:3 Periféricos: Mouse, Teclado

RNF:4 256 MB de RAM o Superior.

RNF:5 150 MB de de espacio libre en el disco duro.

Restricciones de diseño o implementación:

RNF:6 El subsistema será una aplicación de escritorio.

RNF:7 El subsistema será implementado en el lenguaje C#.

RNF:8 El subsistema deberá utilizar como IDE de desarrollo el Visual Studio .NET 2005.

RNF:9 El subsistema usará como herramienta de modelación el Visual Paradigm for UML 6.0 Enterprise Edition.

Requerimientos de Usabilidad

RNF:10 El subsistema deberá tener una interfaz intuitiva, organizada y de fácil entendimiento para el usuario.

RNF:11 El subsistema deberá tener un grupo de menús que sugieran de forma intuitiva al usuario las opciones principales, además de las paletas contenedoras de módulos y corrientes, de edición por capas, y de herramientas de edición de diagramas, con íconos estén en correspondencia con lo que significan en la realidad.

RNF:12 El sistema deberá ser utilizado por usuarios que tengan conocimiento mínimo acerca del funcionamiento y procesamiento de la información con que se trabaja en el sistema.

2.7. Elementos Significativos que condicionan la Arquitectura del simulador DFISim

Subsistema de Funcionalidades o Núcleo

Es el subsistema que rige las acciones que se llevaran a cabo dentro del simulador, constituye el núcleo del mismo ya que posee la lógica de los componentes que forman parte del simulador. El núcleo del

simulador posee además de la responsabilidad organizar de los procesos la realización de los cálculos y aplicación de las fórmulas de la simulación. Cumpliendo con la arquitectura propuesta para cualquier simulador por el proyecto europeo CAPE-OPEN acogido por los estándares internacionales dentro del desarrollo de simuladores de procesos industriales el subsistema de modelado de los diagramas constituiría el módulo de la GUI (Graphic User Interface), junto además al módulo que se encargue de mostrar las interfaces de usuario para la entrada y salida de los datos de los componentes al usuario final, además debe poseer el vínculo directo de las acciones a llevar a cabo por el mismo. El subsistema del núcleo debe integrarse al subsistema de modelado de DFI y a las restantes partes del simulador.

Herramientas de desarrollo

Las herramientas de desarrollo que se vinculan directamente con la arquitectura son las citadas en el capítulo 1, las mismas fueron seleccionadas a partir de las características que presenta la aplicación que existe necesidad de desarrollar y la facilidad que brindan las herramientas seleccionadas para satisfacer estas características, (ver epígrafe 1.12 y 1.14).

2.8. Vista de Casos de Uso

Los CU seleccionados como críticos dado la importancia de los mismos dentro del subsistema a desarrollar así como su impacto en la arquitectura son:

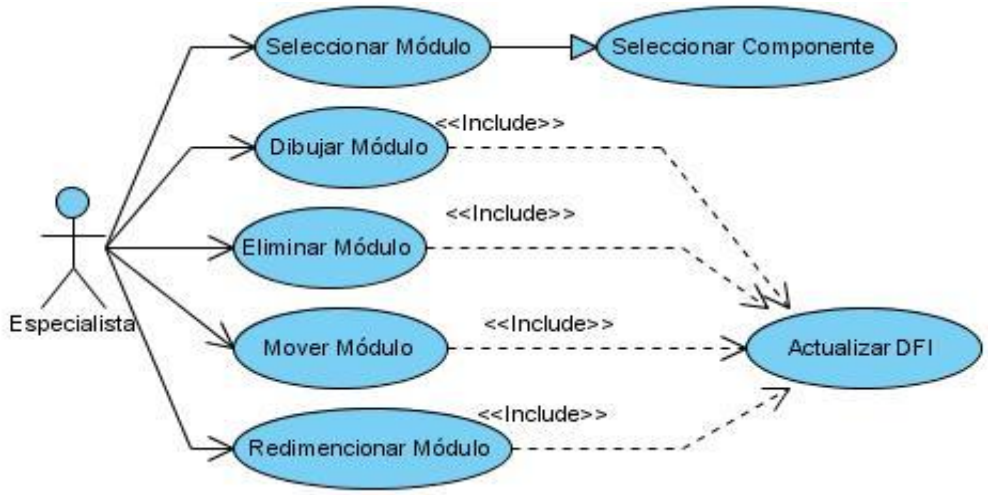

Figura 6: Sección del Diagrama de CU críticos. Gestionar Módulos, fuente propia.

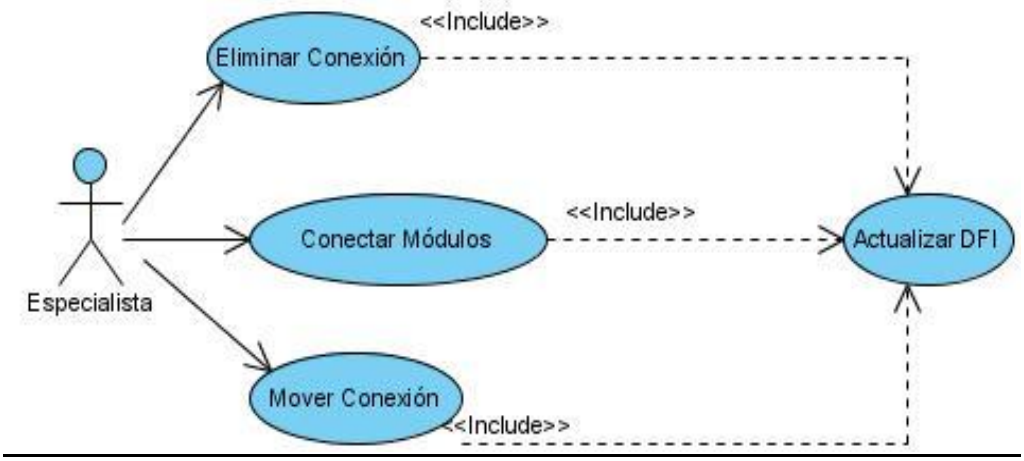

Figura 7: Sección del Diagrama de CU críticos. Gestionar Conexiones, fuente propia.

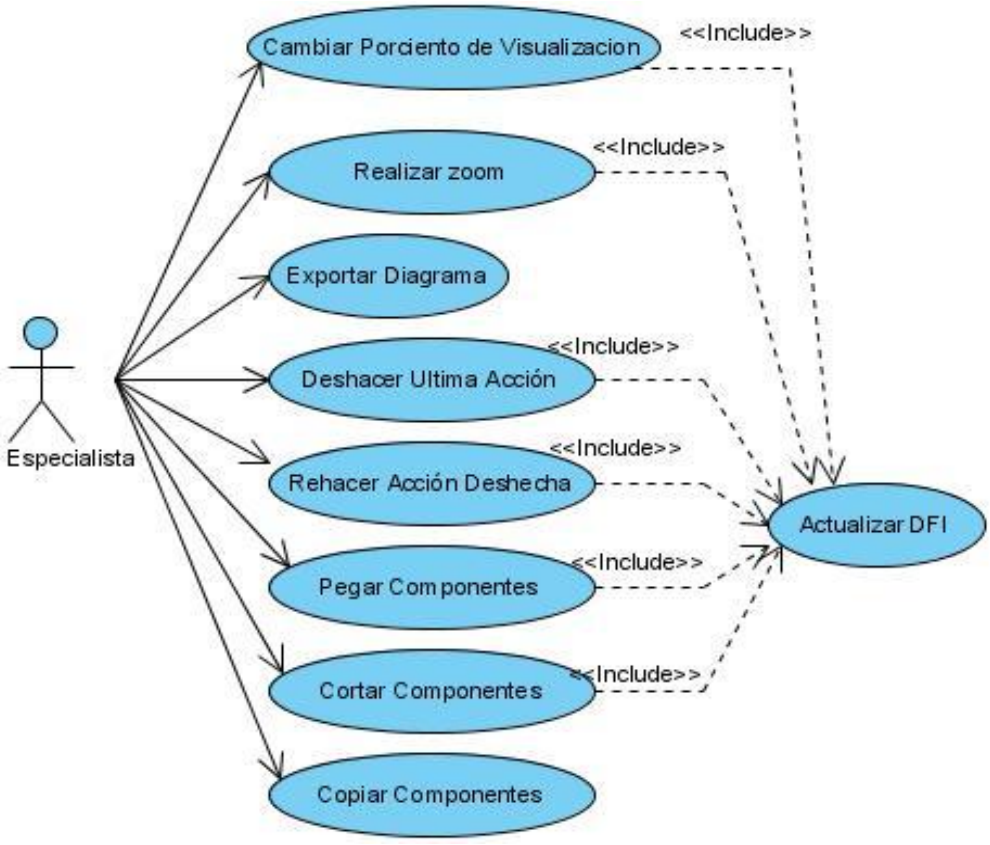

Figura 8: Sección del Diagrama de CU críticos. Gestionar DFI, fuente propia.

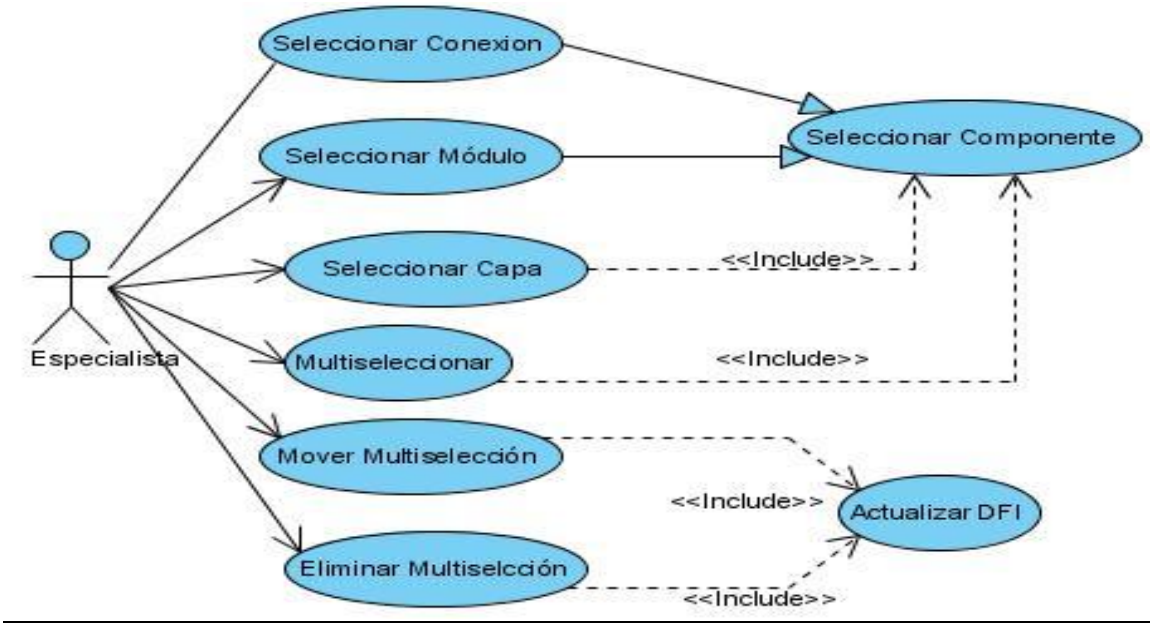

Figura 9: Sección del Diagrama de CU críticos. Gestionar Selección, fuente propia.

Dibujar Módulo

Breve descripción: El CU permite al Especialista dibujar un módulo dentro del diagrama de DFI consta de una paleta de componentes donde el Especialista seleccionará el componente que desea agregar al diagrama. Luego se actualizará la información del DFI.

Seleccionar Módulo

Breve descripción: El CU permite la selección de un módulo el sistema debe marcar la región de selección de dicho componente y posteriormente el sistema activará las opciones de Rotar, Realizar Espejo, Redimensionar, Ocultar, Mostrar y Eliminar Módulo. Además el Especialista puede mover el módulo en cualquier momento a modo de organización del diagrama. El sistema actualizará posteriormente el DFI.

Seleccionar Componente

Breve descripción: El CU permite la selección de un determinado componente el sistema debe marcar la región de selección de dicho componente y posteriormente el sistema activará las opciones de Rotar, Realizar Espejo, Redimensionar, Ocultar, Mostrar y Eliminar Módulo. Además el Especialista puede mover el componente en cualquier momento a modo de organización del diagrama. El sistema actualizará

posteriormente el DFI. Cada componente debe constituir una abstracción de lo que podría ser una etiqueta o una capa.

Eliminar Módulo

Breve descripción: Consiste en borrar el módulo del DFI actual y de la lista de componentes del mismo. Dicho componente no deberá estar ni bloqueado ni oculto dentro del diagrama. Además se actualiza el DFI al finalizar la operación.

Mover Módulo

Breve descripción: Permite al Especialista cambiar de posición el módulo que desee debe poseer al menos un módulo el DFI y el módulo a mover debe estar previamente seleccionado. El sistema deberá actualizar el DFI.

Redimensionar Módulo

Breve Descripción: Permite cambiar de tamaño el módulo seleccionado previamente por cualquiera de sus extremos, luego de esta acción se deberá actualizar el DFI.

Conectar Módulos

Breve descripción: Consiste en la selección de uno de los puntos de enlace de un Módulo conocidos como puertos y establecer una conexión entre el mismo con el puerto de otro Módulo, lo que significa enlazarlo con un punto de enlace del otro componente respectivamente el sistema dibuja una línea que representa dicha conexión. El DFI posteriormente se debe actualizar de forma automática.

Seleccionar Conexión

Breve descripción: Consiste en seleccionar el área de la línea que conecta a una corriente con un módulo o a un módulo con otro permitiendo mover esta conexión ajustándose a la comodidad del Especialista.

Mover Conexión

Breve descripción: Consiste en cambiar de posición una conexión o parte de ella de forma que el Especialista se sienta cómodo y el DFI quede a su forma y satisfacción. Luego de la operación se debe actualizar el DFI.

Eliminar Conexión

Breve descripción: El CU se inicializa cuando el Especialista intenta borrar una conexión establecida entre dos módulos el sistema deberá eliminar la conexión establecida entre los puertos de dichos módulos y luego actualizar el DFI.

Multiseleccionar

Breve descripción: El CU consiste en seleccionar varios componentes dentro del DFI de manera que los mismos puedan ser operador a partir de las operaciones permitidas con los mismo como pueden ser mover, eliminar agrupar en capa entre otras. El sistema marcara como seleccionados a todos los elementos implicados en el CU sean módulos, corrientes o ambos incluyendo a las respectivas conexiones que puedan existir entre los mismos.

Seleccionar Capa

Breve descripción: El CU consiste en que el Especialista pueda seleccionar una Capa antes agrupada por el mismo, donde el sistema deberá marcar como seleccionados todos los componentes que forman parte de la capa.

Mover Multiselección

Breve descripción: El presente CU posee la responsabilidad de cambiar la posición de una selección de varios componentes efectuada dentro del DFI, luego de esa operación el sistema deberá actualizar el DFI así como las conexiones de los módulos dentro de la capa se moverán a la par de los componentes.

Eliminar Multiselección

Breve descripción: El CU elimina del diagrama una selección de componentes antes seleccionada, eliminando además todas las conexiones que contiene dentro entre sus componentes. El sistema debe actualizar el DFI luego de esta operación.

Cambiar Porcentaje de Visualización

Breve descripción: El caso de uso se inicia cuando el Especialista cambia el porciento de visualización del diagrama. El sistema debe cambiar la escala de DFI a partir de una fórmula que especifica la nueva escala y repintar todo el diagrama.

Realizar Zoom

Breve descripción: El caso de uso se inicia cuando el Especialista cambia el área de visualización del diagrama realizando Zoom a la misma. El sistema fija las coordenadas del centro del área de visualización del DFI por las coordenadas actuales del periférico mouse y repinta el DFI.

Exportar Diagrama

Breve descripción: Brinda al especialista la posibilidad de exportar el DFI como una imagen en los formatos predefinidos por el simulador para dicha funcionalidad.

Deshacer Última Acción

Breve descripción: El CU consiste en deshacer una acción que se hecho dentro del DFI dígase cualquier modificación que haya sufrido el diagrama. El CU contará con un máximo de acciones para evitar una utilización extrema de los recursos de la PC. El presente CU solo se aplicará a acciones gráficas luego de la ocurrencia del mismo el sistema deberá actualizar el DFI.

Rehacer Última Acción Deshecha

Breve descripción: El CU consiste en poder rehacer la última acción deshecha. El sistema deberá restablecer el estado del DFI al que se encontraba antes de aplicar la acción deshacer. El DFI será actualizado después de esa acción.

Copiar Componentes

Breve descripción: El caso de uso se inicia cuando el Especialista selecciona al menos un Componente del diagrama y Selecciona la opción de Copiar Componentes. El sistema debe copiar los componentes y guardarlos temporalmente con el fin de poderlos pegar en una próxima acción. Posee como precondición que al menos un elemento del DFI este seleccionado.

Pegar Componentes

Breve descripción: El CU permite pegar los componentes que se encuentren almacenados en la lista temporal de elementos copiados al DFI, después de realizar esta operación se debe actualizar la aplicación y eliminar los elementos de esta lista temporal.

Cortar Componentes

Breve descripción: El CU permite cortar los componentes del DFI y se almacenan en la lista de elementos copiados, los elementos cortados son eliminados de la lista de de componentes del DFI. El sistema habilita desde ese momento la opción Pegar componentes. Luego de la operación de cortado se debe actualizar el DFI.

Actualizar DFI

Breve descripción: Consiste en la actualización o repintado del DFI tomando en cuenta las últimas operaciones realizadas además de tomar en cuenta solo el área que se necesita refrescar actualizando además la lista de componentes que posee el DFI, y las conexiones existentes entre módulos y conexiones entre sí. Es una CU altamente crítico porque posee la responsabilidad de actualizar todos los datos de cada uno de los componentes gráficos del DFI, dibuja toda la reorganización que se ha llevado a cabo desde una acción dada en adelante manteniendo el DFI actualizado.

2.9. Vista Lógica

Diagrama de Paquetes del diseño

Con el fin de lograr una mejor comprensión se han agrupado en paquetes las clases del diseño a partir de las funcionalidades que las mismas brindan. Cada paquete posee la responsabilidad de encapsular las funcionalidades que en conjunto brindan las clases que lo componen. De esta manera se logra la comprensión de las dependencias de los paquetes de diseño. El objetivo fundamental de dicha agrupación es que las clases a partir de las funcionalidades a las que responden obtengan una mayor modularidad y reutilización del código permitiendo además la flexibilidad de cada uno de los paquetes de diseño ante un futuro crecimiento de datos o crecimiento de funcionalidades a fines con los datos que las clases encapsulan. Los paquetes que forman parte del subsistema a desarrollar se muestran en la siguiente figura:

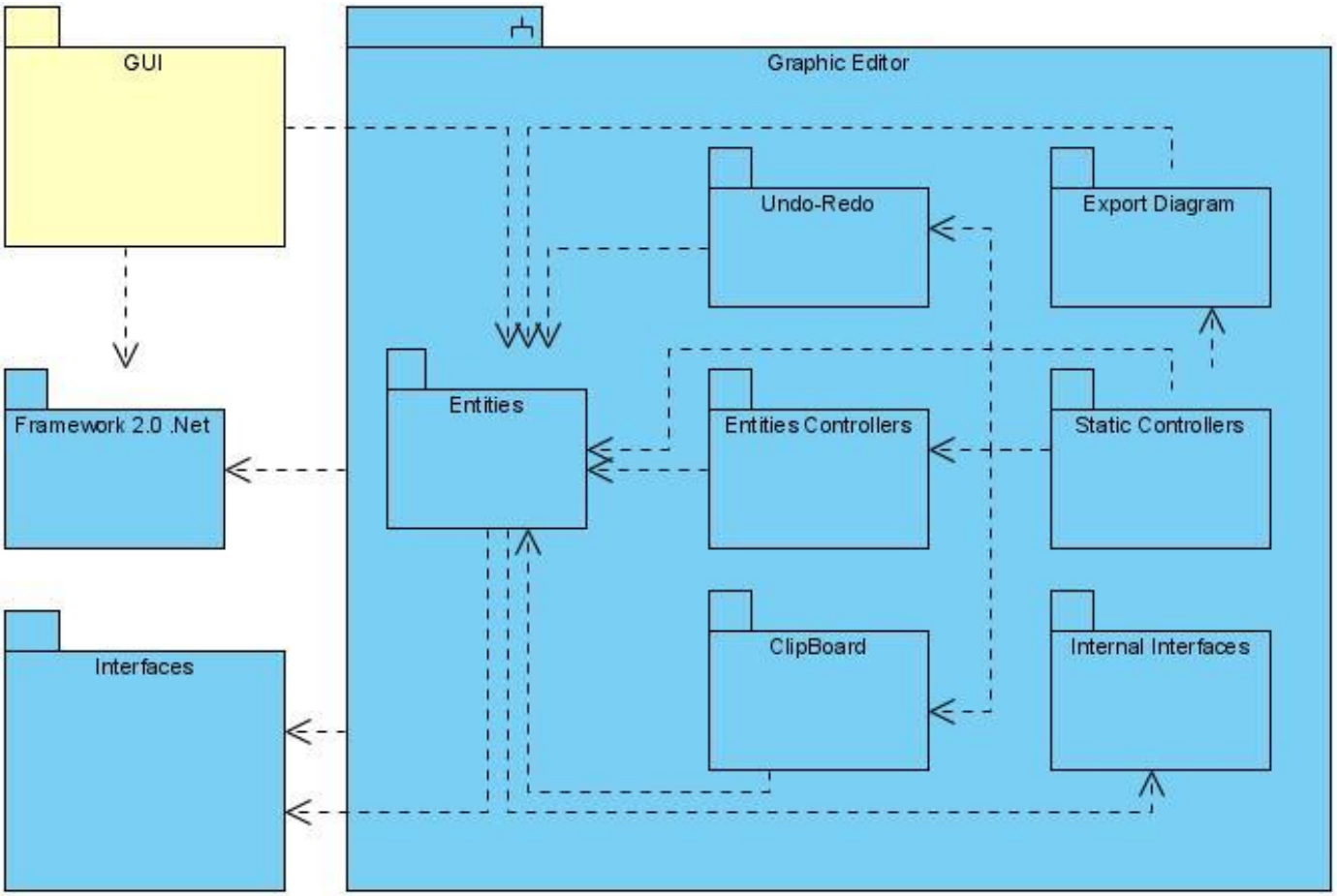

Figura 10: Diagrama de Paquetes del Modelo de Diseño.

GUI: El paquete de diseño GUI contiene el formulario principal de la aplicación así como todos los controles que brinda la plataforma .Net como son los Menús, Submenús, Botones entre otros, también contiene los controles utilizados como contenedores entre estos se encuentran los Paneles, TabControls, TabPages etc. Además es en este subsistema donde se mostrará toda la información de los componentes que se encuentren activos en un instante determinado. Dígase el diagrama de DFI con el que se encuentre trabajando el Especialista. El paquete contiene además la barra de Herramientas que posee un acceso directo a las funcionalidades del subsistema de modelado como son el Undo (Deshacer), el Redo (Rehacer), Agrupar en Capa, Eliminar Modulo entre otras. La Paleta de Componentes que permite adicionar nuevos módulos al diagrama de DFI también formará parte de este paquete de diseño. En realidad dentro de un simulador de procesos industriales esta paleta tiende a una mayor complejidad, dentro del subsistema de modelado solo se realizará la misma con el fin de poder probar de una manera sencilla las operaciones básicas del subsistema y no para representar la paleta de todo un simulador la cual debe ser mucho más complicada dada las características de los módulos que formen parte del mismo. Es este el paquete que contiene la interfaz de usuario con la que interactuará de forma directa el Especialista.

Entities: Encapsula las clases entidades que intervienen el subsistema de modelado de DFI. Estas clases no son más que clases diseñadas para encapsular a su vez los datos y mecanismos de lectura y escritura de cada uno de sus atributos.

Entities Controllers: Paquete de diseño que se encuentra compuesto por las clases controladoras de cada una de las clases entidades pertenecientes al paquete Entities, estas clases controladoras poseen los métodos de las entidades que manipulan.

Static Controllers: Paquete de diseño que encapsula las clases controladoras que responden de igual manera para cualquier entidad contiene aquellas clases más generales que son útiles pero que no se especializan en el manejo de ninguna entidad, además contiene las clases que poseen los métodos de las operaciones que se realizan a nivel de diagrama de DFI como son Dibujar Contorno, Realizar Zoom entre otras. Dentro de este paquete se encuentran los gestores de las funcionalidades principales del Subsistema de Modelado de DFI, es un paquete de suma importancia ya que contiene la clase controladora que se encarga de dibujar los módulos sobre el DFI además de actuar como despachadora de los eventos que desde la interfaz captura un editor que posea dicho diagrama.

Internal Interfaces: Contiene las clases interfaces que intervienen en el negocio de la aplicación. Son aquellas interfaces que son utilizadas por las clases del propio subsistema de modelado de DFI.

Interfaces: Como bien lo indica el nombre del paquete contiene las interfaces que implementan las entidades que forman parte del subsistema de modelado con el objetivo de que si en un futuro se desea implementar diferentes tipos de módulos se pueda utilizar las funcionalidades brindadas por el Subsistema de Modelado con tan solo definir las operaciones de las interfaces requeridas. Además constituyen el mecanismo de comunicación para interactuar con otros subsistemas ya que este paquete contiene las interfaces brindadas por el subsistema.

Undo-**Redo:** Contiene las clases controladoras que intervienen en la realización de la funcionalidad Deshacer y Rehacer. Debido a la complejidad de la funcionalidad y la carga de flujo de información que se maneja dentro de la misma se decidió que sería conveniente agrupar dichas clases en un paquete previendo un posible crecimiento futuro de dichas clases. Dígase existir necesidad de realizar las operaciones a otro tipo de entidad diferente del editor gráfico.

Export Diagram: Contiene las clases controladoras que intervienen en la realización de la funcionalidad Exportar diagrama de DFI formato imagen. Responde a la funcionalidad de tal manera que permite agregar nuevos formatos de exportación si se desea.

Clipboard: Contiene las clases que brindan las funcionalidades de copiar componentes de DFI, cortarlos y tener la posibilidad de pegarlos en otro diagrama que desee.

Framework 2.0 .Net: Contiene las clases que forman parte del Framework 2.0 de .Net el cual es necesario que se encuentre instalado en el ordenador donde se ejecute la aplicación y del cual el subsistema Graphic Editor dependerá en la medida que se utilicen los controles y clases que el mismo brinda para el desarrollo de sistemas para el desarrollo del subsistema de modelado de DFI.

Previniendo un cambio de algún requisito funcional o no funcional, la distribución de los paquetes se ha realizado de forma tal que permita agregar funcionalidades afectando lo menos posible el funcionamiento del resto de la aplicación en cualquiera de los paquetes existentes.

Realización de los principales CU:

A continuación se muestra la realización de algunos CU críticos con el objetivo de mostrar la interacción de las instancias de clases realizando las operaciones que dan respuesta a los mismos.

Sección Gestionar Módulo

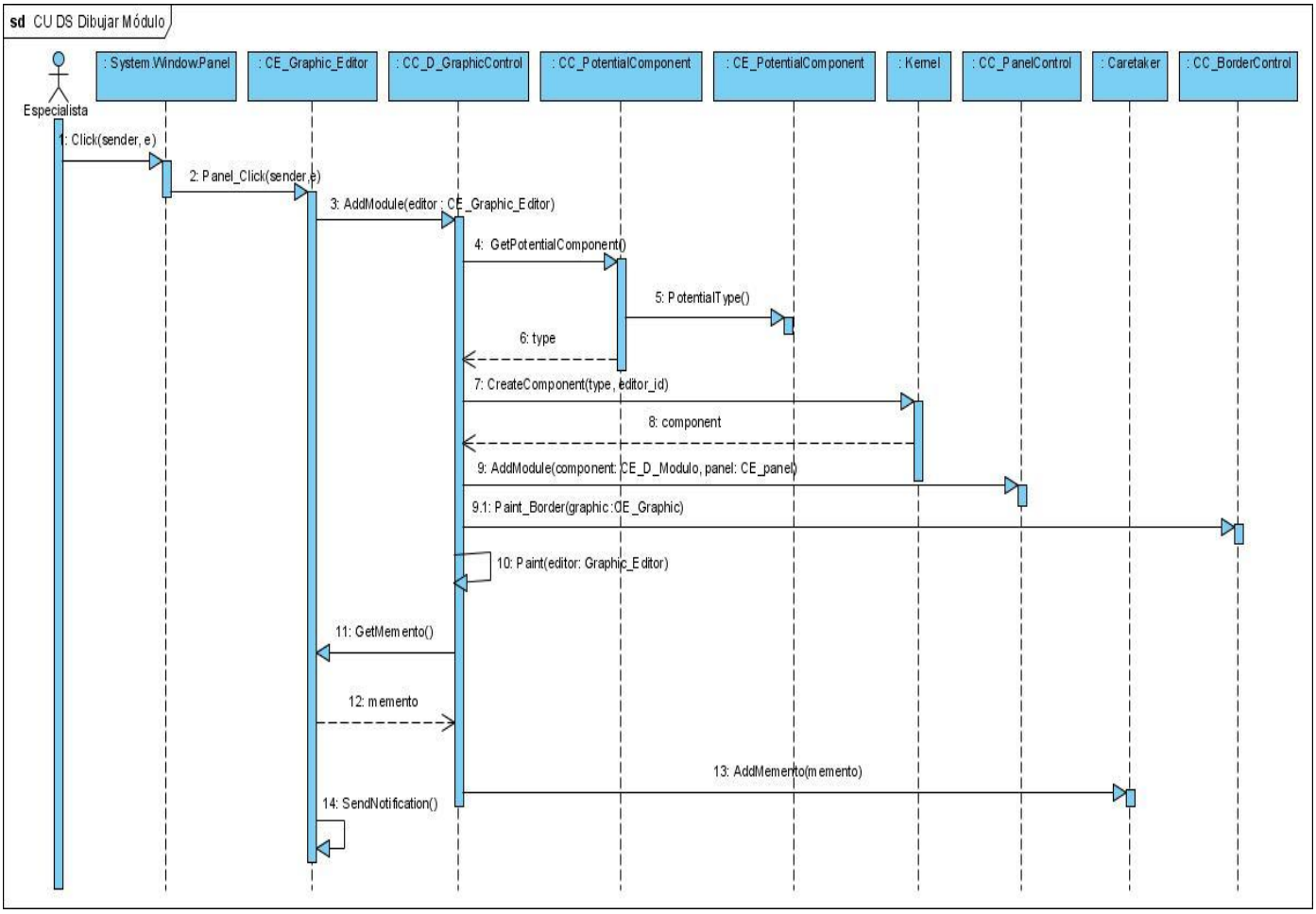

Figura 11: Diagrama de Secuencia del CU Dibujar Módulo.

Se selecciona un elemento de la paleta de componentes. El Subsistema de GUI comunica el evento a través de la interfaz IPotentialComponent e invoca al método AddPotentialComponent modificándose así el atributo de la CE_PotentialComponent. Al levantarse el evento Click sobre el panel del editor, este captura el evento y lo comunica a CC_GraphicControl quien solicita entonces el tipo almacenado en la CE_PotentialComponent e indica a quien utilice el subsistema (Kernel) que debe adicionar un elemento pasándole el tipo de dato y el identificador del editor donde se desea adicionar. Luego le indica el CC_PanelControl que adicione un elemento pasándole el mismo como parámetro. El controlador de

Bordes (CC_BorderControl), dibuja el borde de selección de dicho elemento y la clase CC_GraphicControl repinta todo el panel y al final el editor adiciona un estado dentro de clase CareTaker.

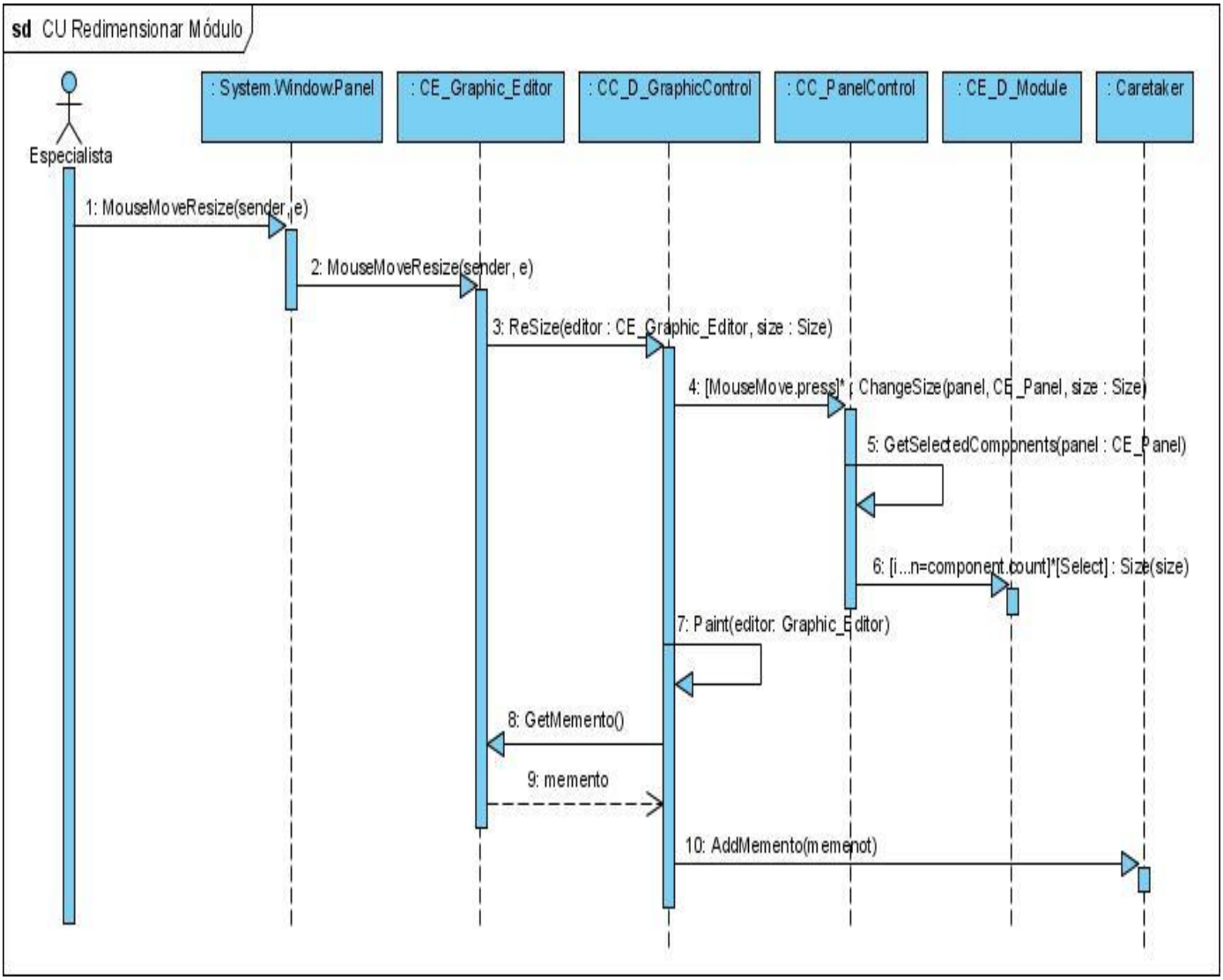

Figura 12: Diagrama de Secuencia del CU Redimensionar Módulo.

El CU consiste en cambiar el tamaño de un módulo seleccionado, una vez que el Especialista sin dejar de accionar el botón mueva el mouse por las esquinas del módulo entonces el editor captura el evento y lo envía a la clase CC_GraphicControl la cual indica al CC_PanelControl que modifique la coordenadas del tamaño (Size) del módulo seleccionado para luego repintar el panel. Esta operación se realiza mientras el

especialista no deje de presionar el mouse. Una vez que se deje de accionar el botón izquierdo del mouse entonces el editor adiciona un nuevo estado al CareTaker.

Sección Gestionar DFI

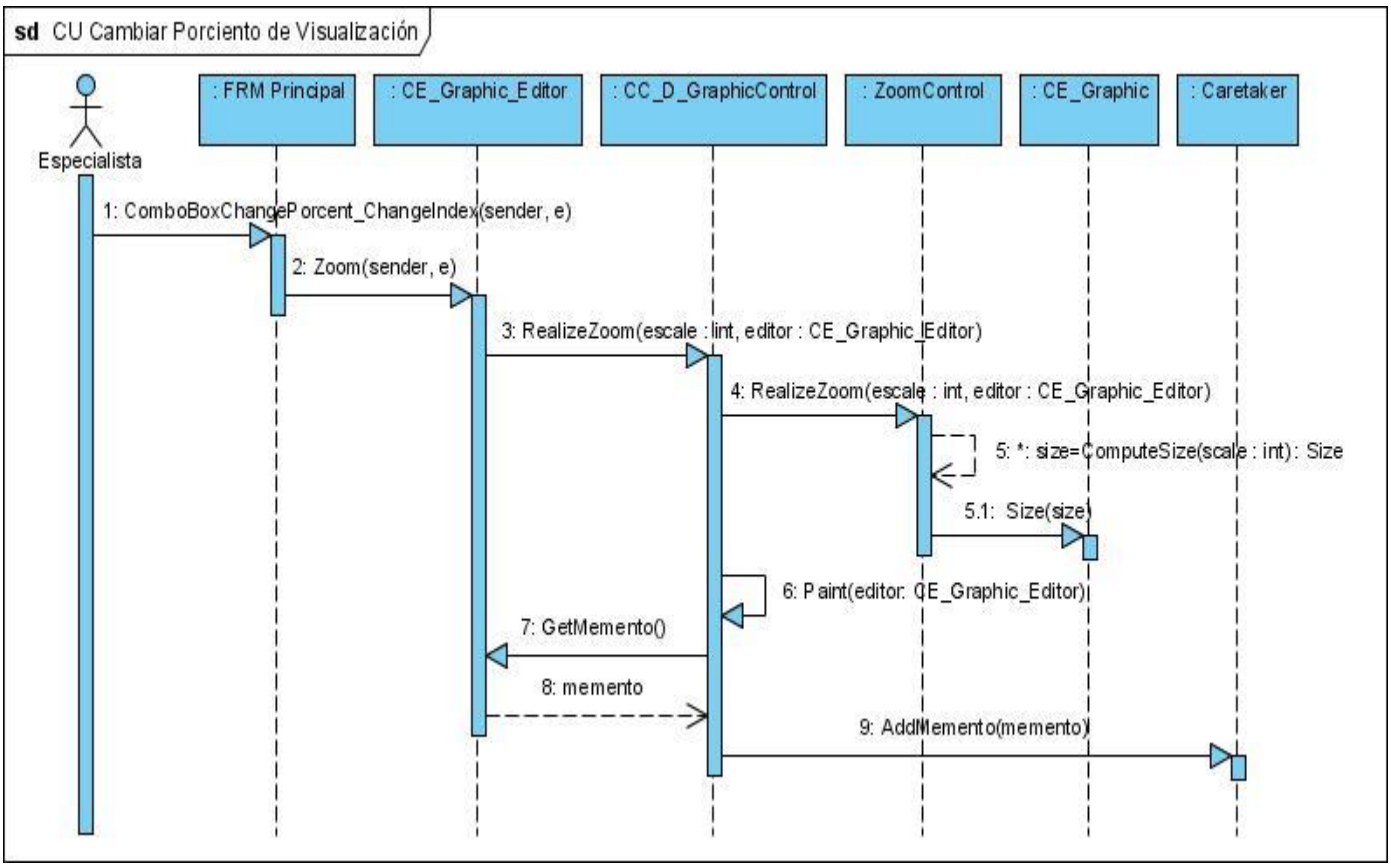

Figura 13: Diagrama de Secuencia del CU Cambiar Porcentaje de Visualización.

El CU comienza cuando se modifica el ComboBox que contiene la escala de visualización del DFI, este evento es capturado por el editor que indica a CC_GraphicControl que realice la operación Zoom y le pasa la escala y el editor, GraphicControl comunica al Control de Zoom pasándole los atributos y es este quien finalmente calcula el nuevo tamaño que deben poseer los atributos gráficos en dependencia de la escala a la que se encontraba visualizado el DFI y modifica los elementos gráficos. Luego CC_GraphicControl redibuja el DFI y el editor adiciona un estado a su clase CareTaker.

El diagrama de Secuencia del CU Realizar Zoom es análogamente similar al CU Cambiar Porcentaje de Visualización la única diferencia es la procedencia de la escala de aumento del DFI ya que en el caso de el CU Realizar Zoom proviene del periférico mouse específicamente de la acción sobre la rueda del mouse, de ahí que diagrama de secuencia del dicho CU sea omitido del presente documento.

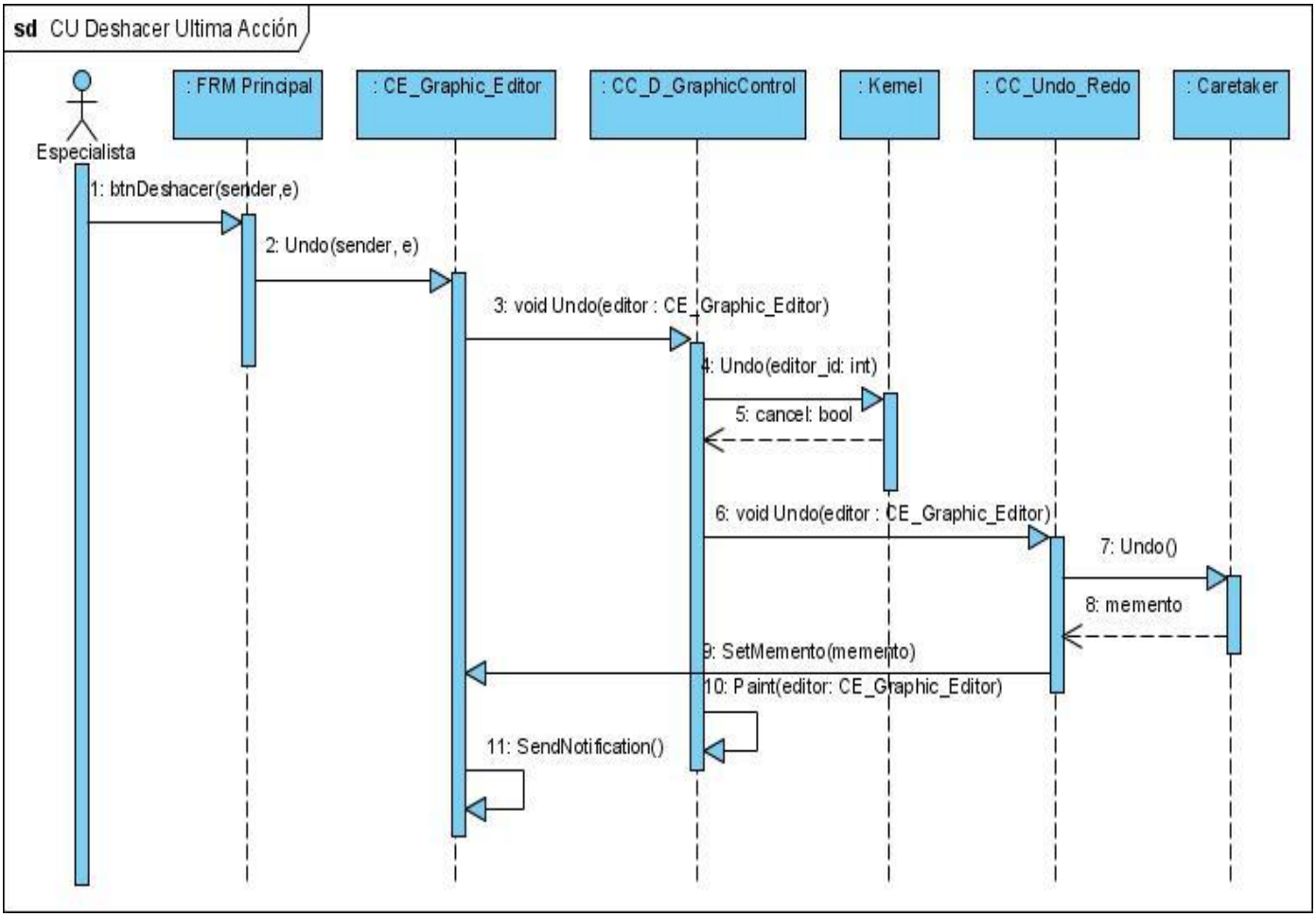

Figura 14: Diagrama de Secuencia del CU Deshacer Última Acción.

El Especialista acciona la opción rehacer acción, este evento lo capta el editor quien lo comunica a la clase CC_GraphicControl. Luego esta le indica la operación a la clase controladora de las operaciones CC_UndoRedo y le pasa como parámetro el editor. CC_UndoRedo busca dentro del CareTaker del editor el estado anterior y le indica al mismo que modifique sus datos a partir de los del estado devuelto. Después de esto la clase CE_Graphic_Editor repinta el panel del editor y el editor notifica el cambio.

El CU Rehacer Última Acción es análogamente similar al Deshacer Última Acción debido a esto se omitió el mismo del presente documento.

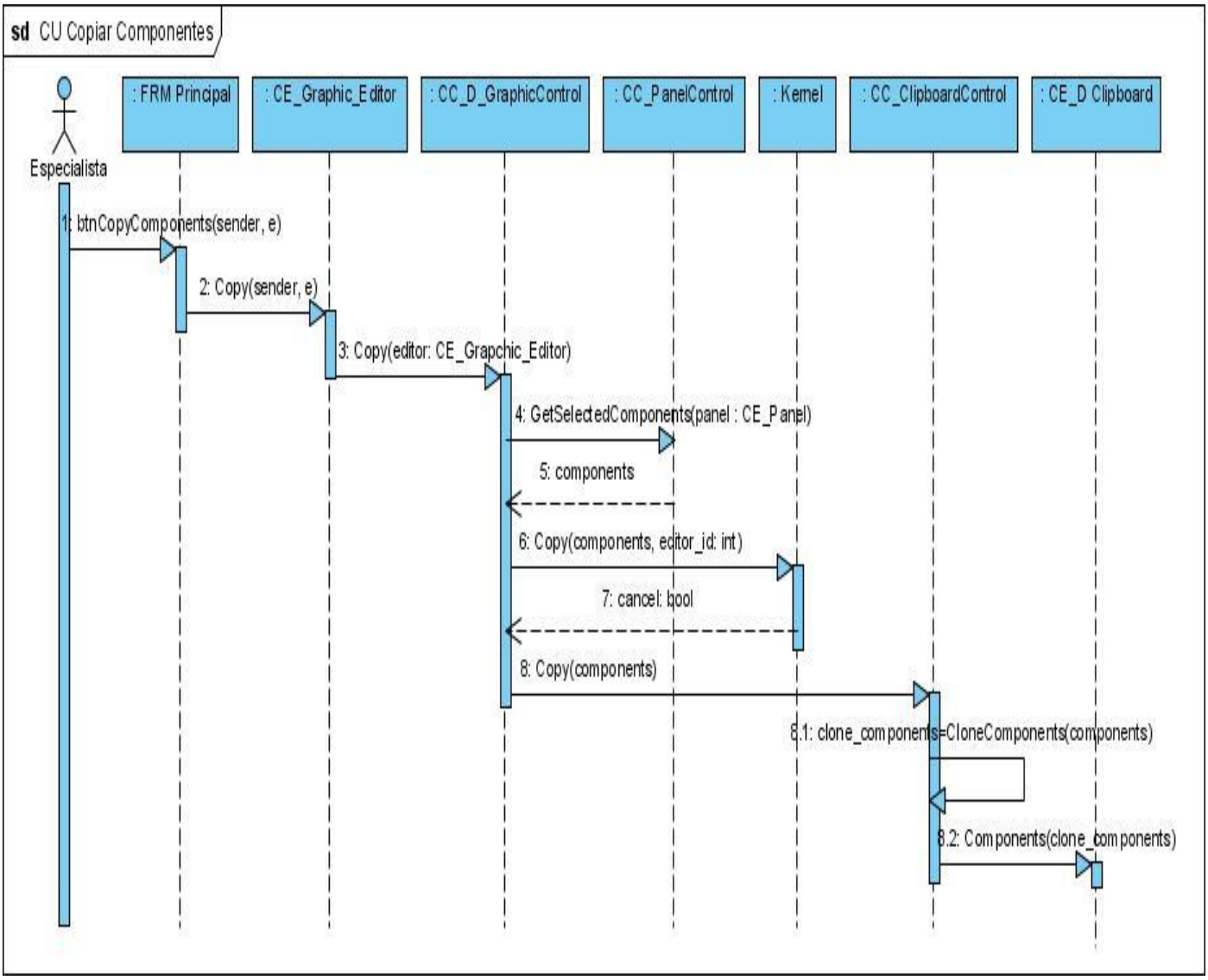

Figura 15: Diagrama de Secuencia del CU Copiar Componentes.

El CU comienza cuando el Especialista acciona la opción copiar o levanta el evento del teclado Ctrl + C, esas acciones son captadas por el editor que indica a la controladora gráfica que copie los componentes y le pasa el editor, esta solicita entonces al CC_PanelControl los elementos seleccionados los cuales pasa

entonces al a la clase CC_ClipBoardControl para que realice el Copy (Copiar) de estos elementos que no es más que escribir los elementos en la entidad CE_ClipBoard de donde se obtendrán una vez que levanten el evento Pegar Componentes.

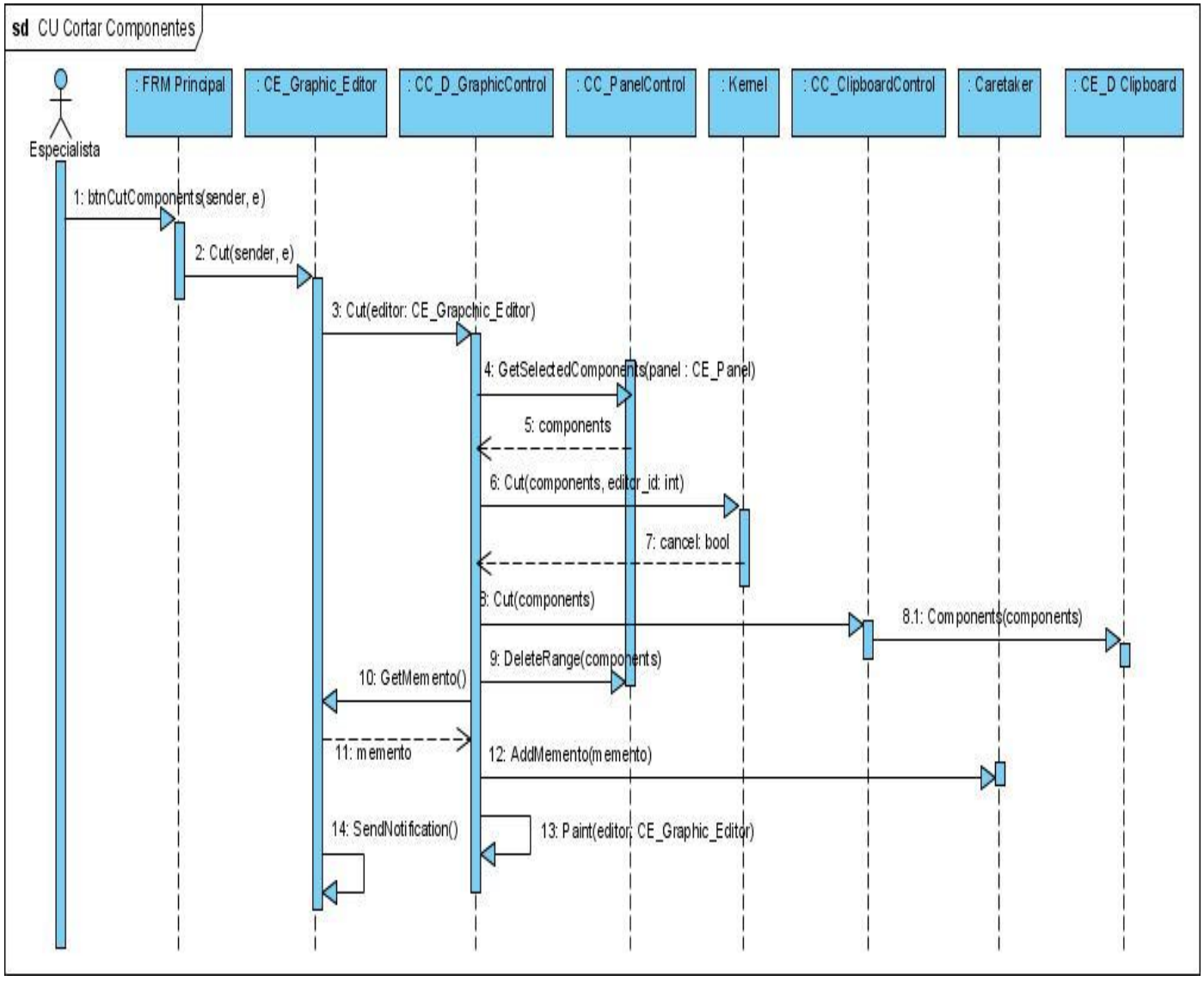

Figura 16: Diagrama de Secuencia del CU Cortar Componentes.

El CU comienza cuando el Especialista acciona la opción cortar o levanta el evento del teclado Ctrl + X, esas acciones son captadas por el editor que indica a la controladora gráfica que corte los componentes y

le pasa el editor, esta solicita entonces al CC_PanelControl los elementos seleccionados, pide la autorización al Núcleo (Kernel) para cortarlos y que el mismo también realice la operación Cortar pasándole los elementos a Cortar del editor y el identificador del mismo , luego la CC_ClipBoard realiza la operación Cortar (Cut) de estos elementos que no es más que escribir los elementos en la entidad CE_ClipBoard de donde se obtendrán una vez que levanten el evento Pegar.

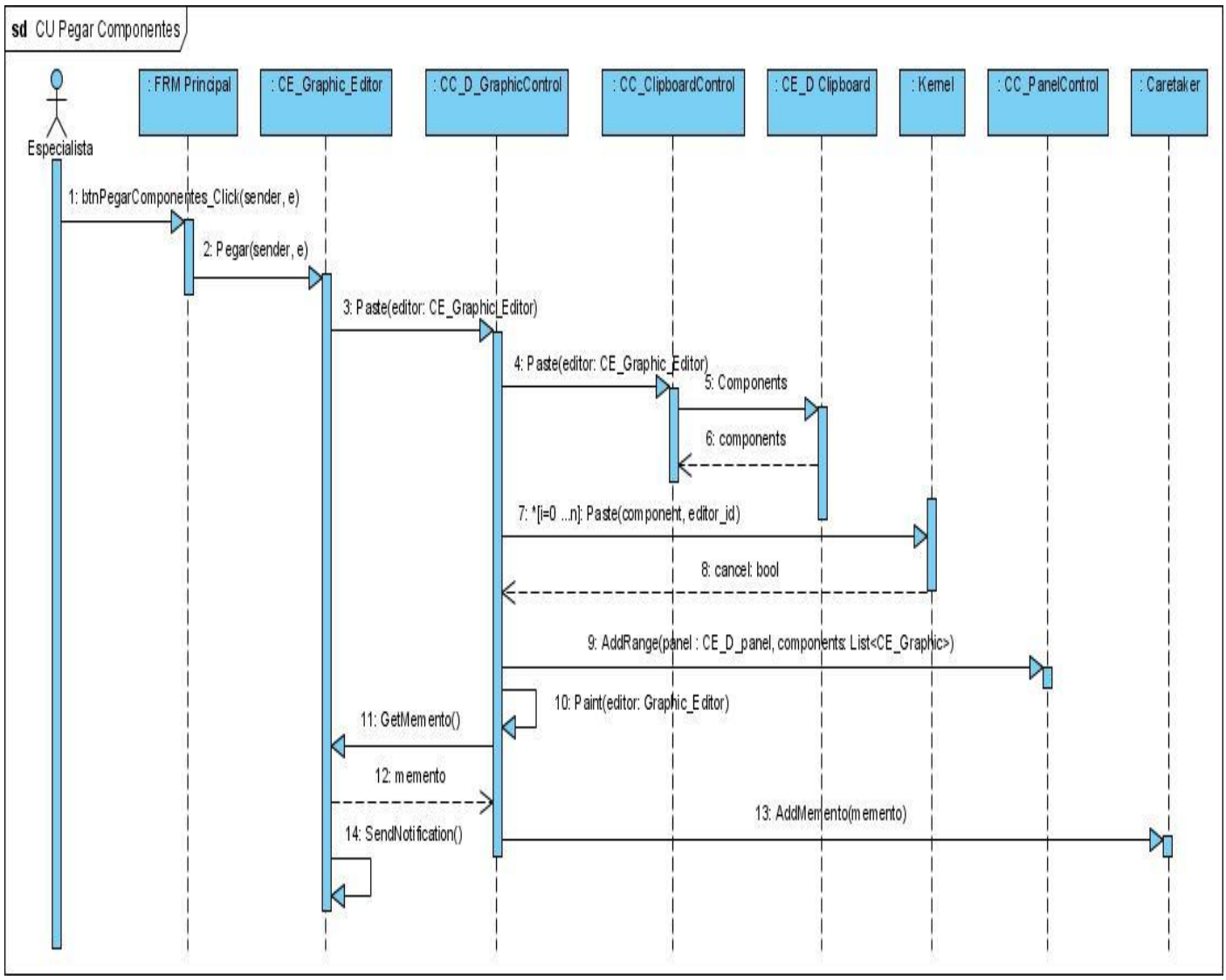

Figura 17: Diagrama de Secuencia del CU Pegar Componentes.

El CU comienza cuando el Especialista acciona la opción pegar o levanta el evento del teclado Ctrl + v, esas acciones son captadas por el editor que indica a la controladora gráfica que pegue los componentes y le pasa el editor, esta ultima comunica entonces al Núcleo (Kernel), del simulador la operación que debe realizar si el Kernel autoriza la operación entonces indica a la clase CC_ClipBoardControl que realice la operación Pegar y es esta quien obtiene los elementos de la entidad CE_ClipBoard y comunica uno a uno los elementos al Kernel a medida que este los adiciona en el editor especificado entonces el subsistema los adiciona también al editor, con sus respectivos datos. Finalmente el editor adiciona un nuevo estado a su CareTaker.

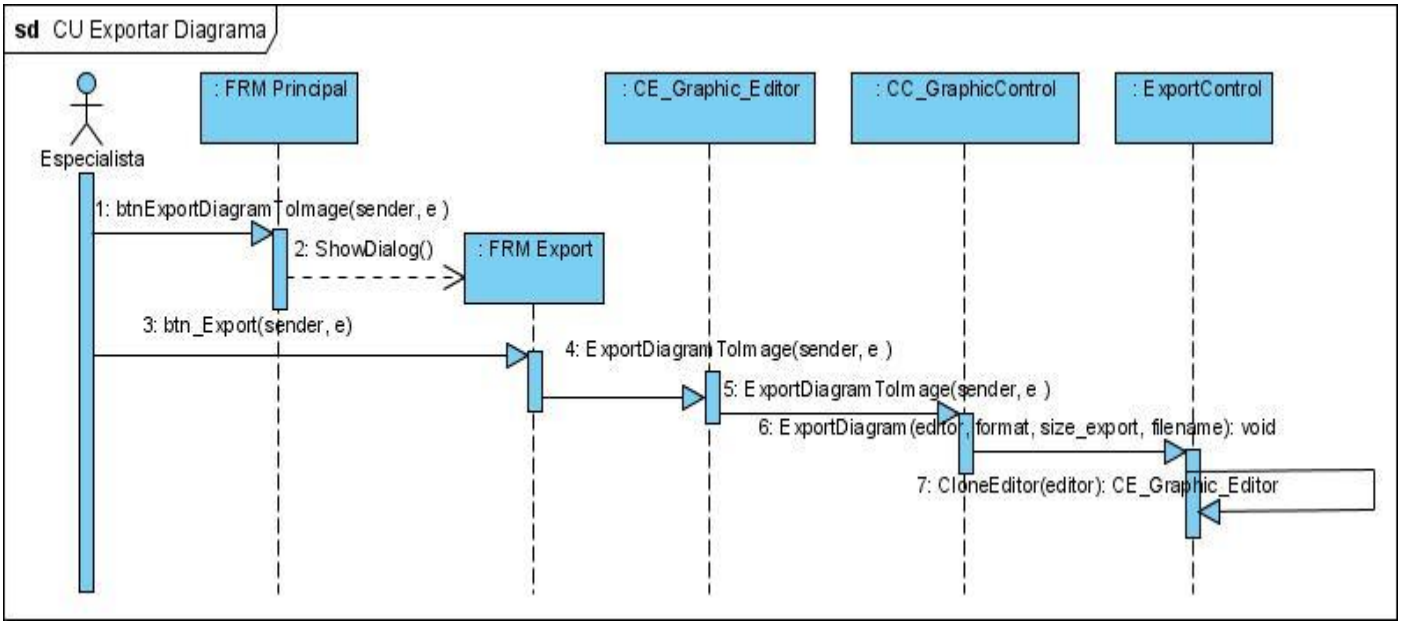

Figura 18: Diagrama de Secuencia del CU Exportar Diagrama.

El especialista selecciona la opción exportar diagrama como Imagen, este evento lo captura el editor y lo comunica a la CC_GraphicControl que a su vez le indica la acción a la clase ExportControl que gestiona todo el proceso de exportación esta clase posee varios métodos que constituyen opciones de exportación a diferentes formatos y con diferentes tamaños en el caso de la imagen. Esta clase verifica los elementos del panel del editor que le es pasado por parámetro una vez que se le indica que exporte la imagen y se le pase como parámetro la dirección donde se salvará la imagen entonces este la construye escribe en la

dirección indicada. Por parte de la interfaz se creará para probar la integración un formulario que gestione la exportación y los formatos de la dicha exportación.

Sección Gestionar Selección

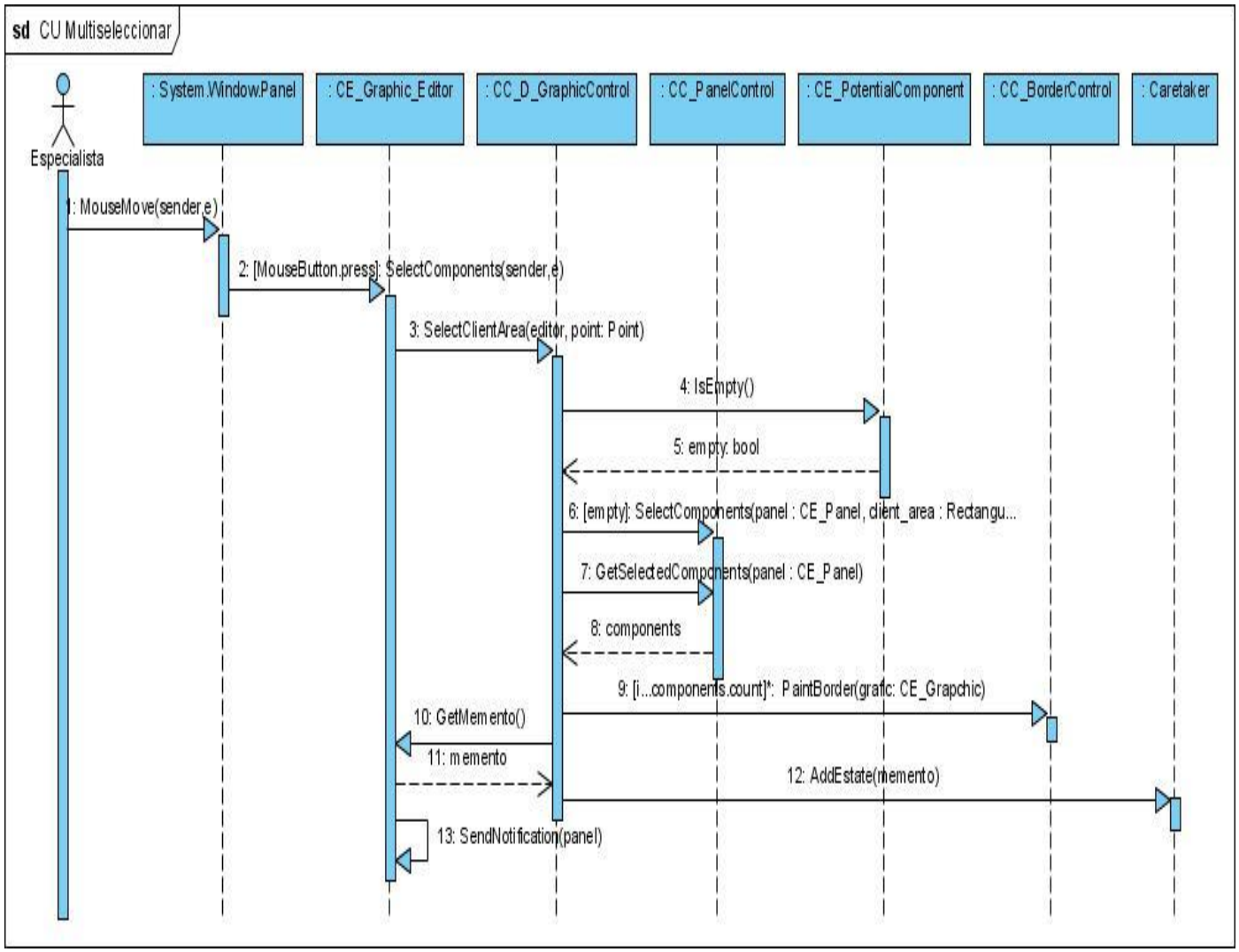

Figura 19: Diagrama de Secuencia del CU Multiseleccionar.

El Especialista marca un área de selección en el panel, el editor captura el evento y lo comunica a CC_GrapchicControl esta entonces le indica a el CC_PanelControl que seleccione los elementos dentro del área seleccionada. Luego le solicita estos elementos y se los pasa al controlador de borde

CC_BorderControl y este le dibuja los bordes de seleccionados a todos los elementos indicados cuidando además los tipos de elementos que se pasen como parámetro. Ya que si el elemento es una capa el deberá seleccionar todos los elementos de la capa.

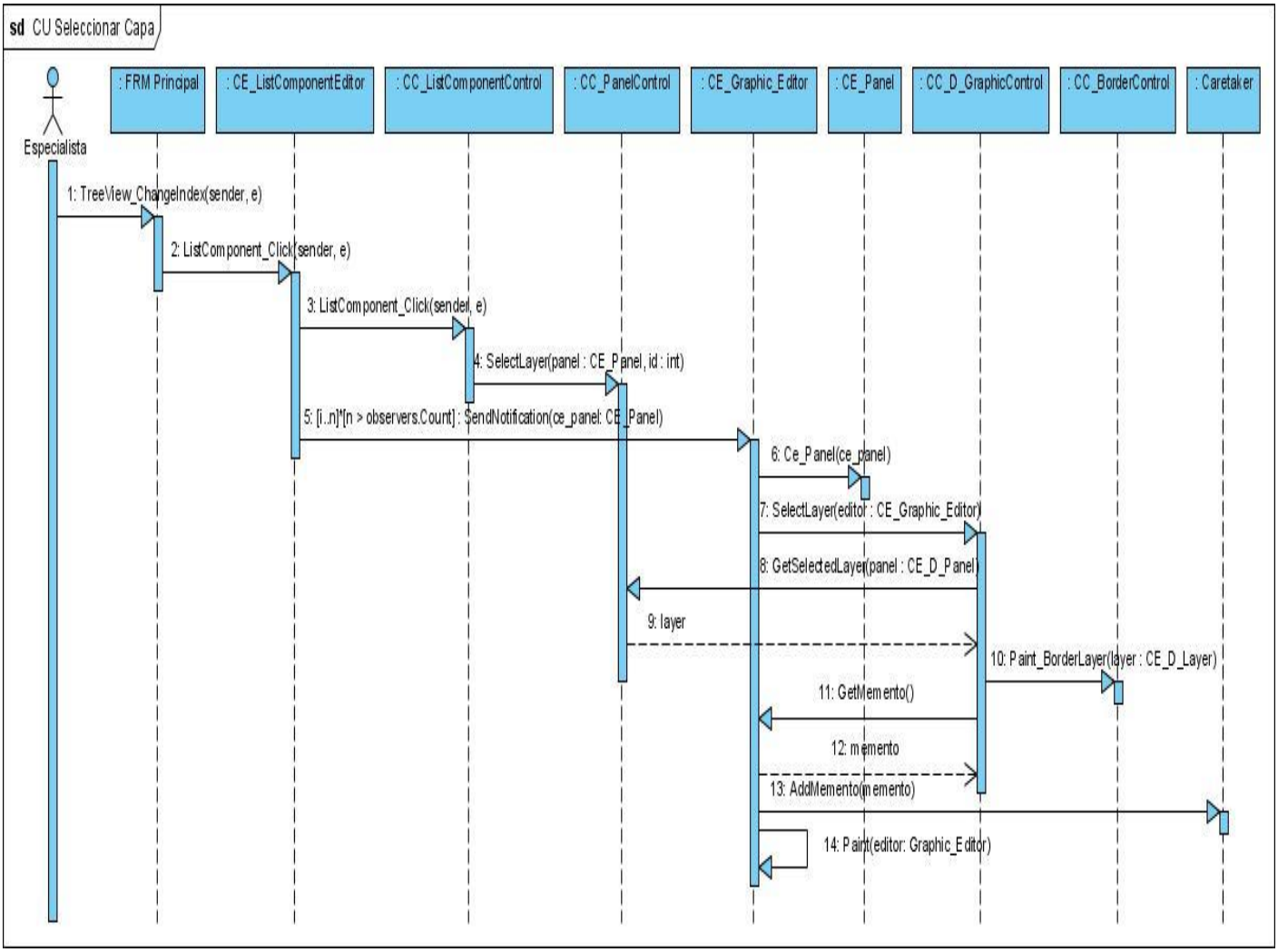

Figura 20: Diagrama de Secuencia del CU Seleccionar Capa.

El CU comienza cuando el especialista acciona el Click del periférico mouse sobre el Treeview que contiene la lista de capas este evento es captado por la clase CE_ListComponentEditor y esta comunica el evento a su clase controladora. CC_ListComponentControl le comunica a CC_PanelControl que seleccione la capa y este deberá poner en verdadero todos los componentes de la capa. Luego se envía

la notificación de que se modificó la capa y es el editor quien actualiza los datos de su panel, luego pasa esta capa al CC_BorderControl quien se encarga entonces de dibujarle el contorno de selección al componente que se le indica (la capa) este también deberá dibujar el contorno de todos los elementos de la capa y finalmente CC_GrapchicControl redibuja el panel.

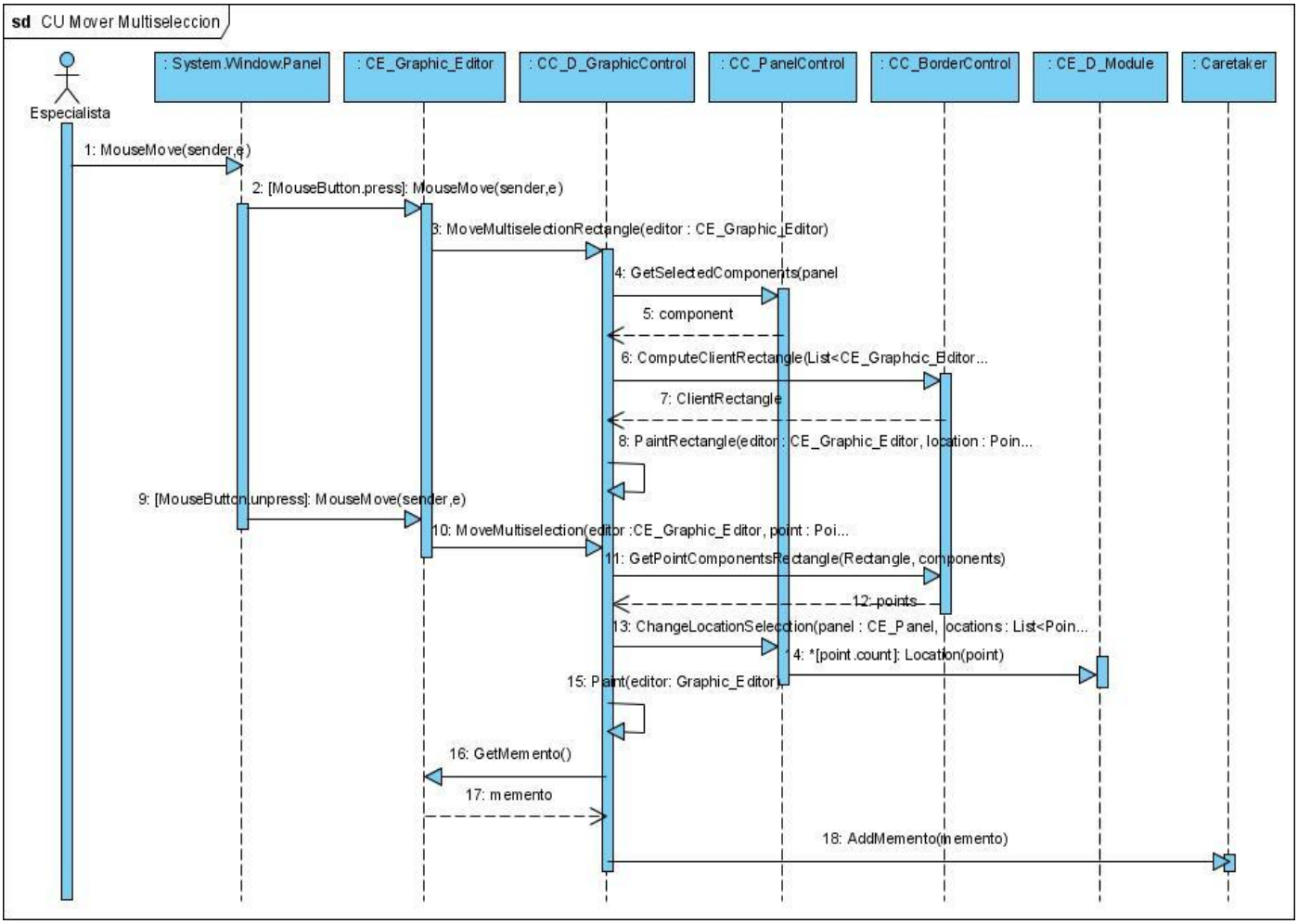

Figura 21: Diagrama de Secuencia del CU Mover Multiselección.

El especialista previamente deberá tener al menos un elemento seleccionado, cuando el especialista mueva el periférico mouse con el botón izquierdo presionado entonces la clase CC_GraphicControl que el editor le comunicó el evento solicita al controlador de borde CC_BorderControl que le devuelva el rectángulo de la selección que le fue dada anteriormente por el controlador de panel CC_PanelControl.

Dicho rectángulo se dibuja sobre el panel y mientras el especialista este moviendo el mouse se repetirá esta acción. Una vez que el mismo deje de presionar el botón del mouse, entonces a partir de la lista de puntos que ocupan los elementos dentro del rectángulo se les modifica el atributo localización (location) con el valor de la nueva ubicación. Luego CC_GraphicControl repinta el panel y el editor adiciona un nuevo estado al Caretaker.

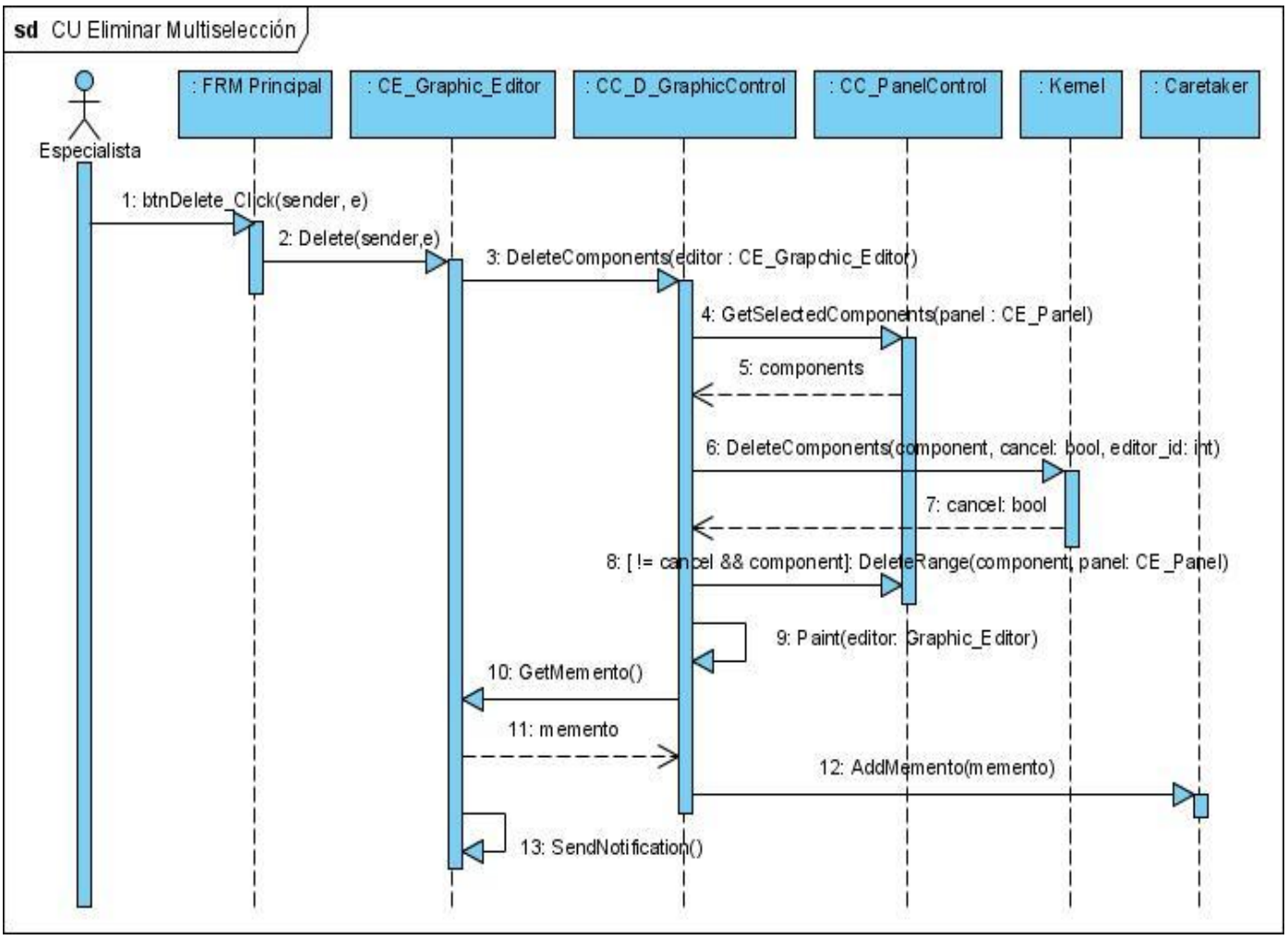

Figura 22: Diagrama de Secuencia del CU Eliminar Multiselección.

El Especialista acciona la opción eliminar de la GUI ese evento es captado por editor quien comunica del mismo a la clase CC_GraphicControl, esta solicita a la clase CC_PanelControl los elementos seleccionados ya con estos elementos le indica al Kernel que los debe eliminar además indicándole el

editor de donde debe eliminarlos. Finalmente le indica al controlador de panel que elimine los componentes seleccionados, para el caso de los módulos se eliminan también las conexiones del mismo y en el caso de las capas se eliminan los componentes de las mismas finalmente CC_GrapchicControl repinta el panel y el editor adiciona un nuevo estado en su CareTaker y envía notificación de que ha sido modificado.

Sección Gestión de Conexiones

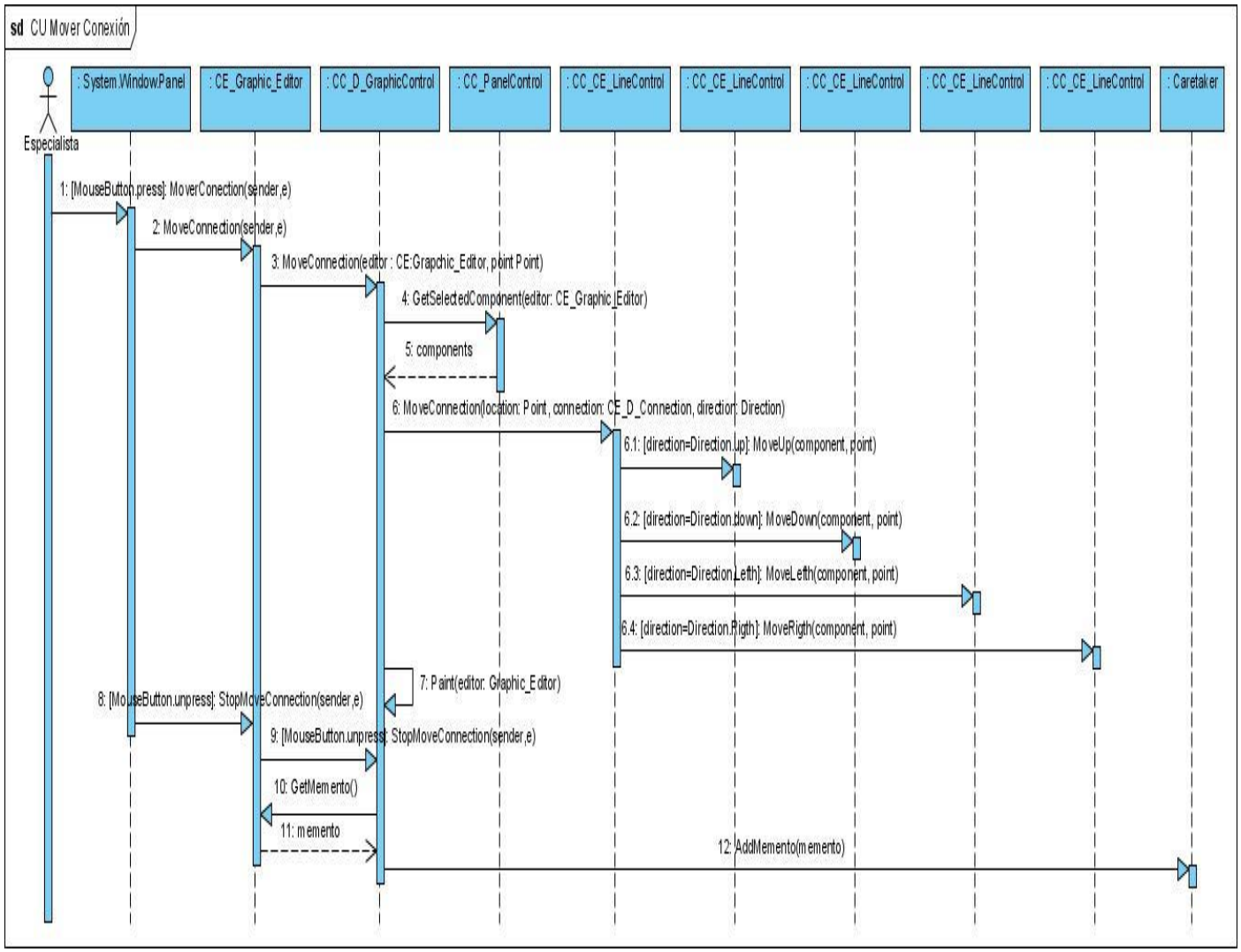

Figura 23: Diagrama de Secuencia del CU Mover Conexión.

El especialista previamente deberá tener el elemento seleccionado, cuando el especialista mueva el periférico mouse con el botón izquierdo presionado entonces la clase CC_GrapchicControl le pide el elemento seleccionado al CC_PanelControl e indica a la clase CC_LineControl que se intenta mover la línea es esta clase quien modifica las coordenadas de la conexión en cuestión. Una vez que el Especialista deje de presionar el botón del mouse la clase CC_GrapchicControl repinta el panel y el editor adiciona un nuevo estado al Caretaker.

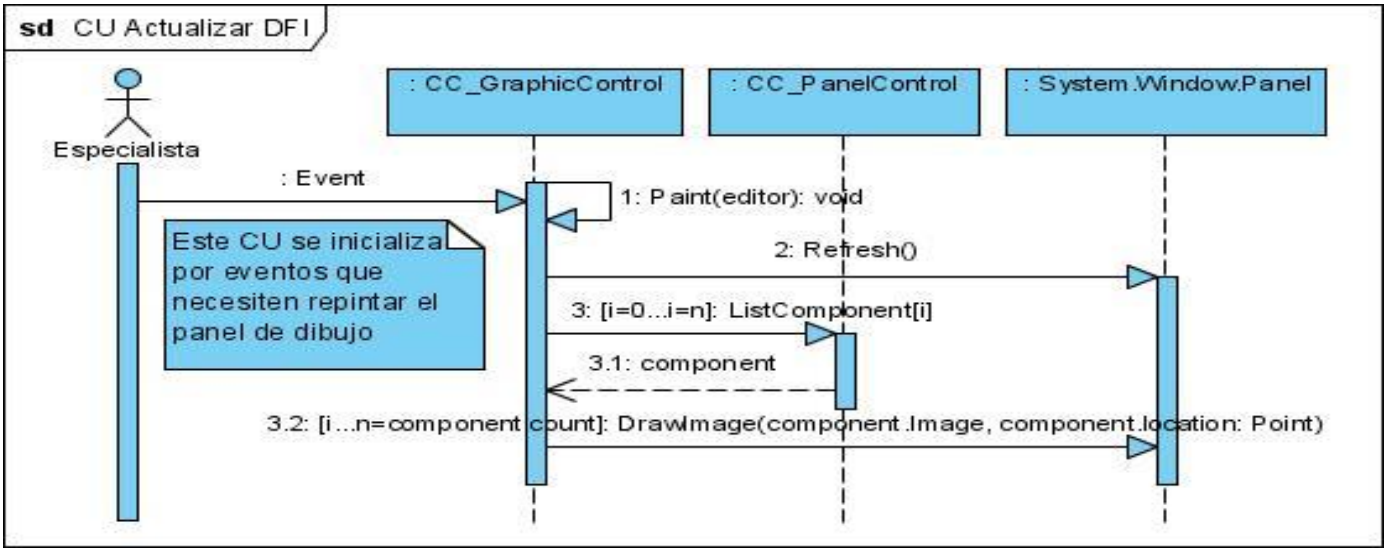

Figura 24: Diagrama de Secuencia del CU Actualizar DFI.

El CU comienza cuando algún evento dentro del DFI ha realizado una modificación en algún componente de forma tal que necesita ser repintado el panel dibujo a modo de que el usuario conozca que la acción que el mismo deseaba ha sido realizada satisfactoriamente. La clase CC_GrapchicControl dibuja después de haber refrescado el panel o la parte del panel que necesite ser redibujada, dibuja cada una de las imágenes de los componentes implicados en dicha modificación.

2.10. 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. Puede considerarse que un componente es la parte física de un sistema. Las clases son conceptos de abstracción que poseen atributos y métodos que se implementan o materializan en los componentes.

Se pueden encontrar tres grandes grupos fundamentales de componentes:

Los componentes de distribución: Son aquellos que conforman el sistema como son programas ejecutables y librerías dinámicas entre otros.

Los componentes de trabajo: Son aquellos que se utilizan para conformar los componentes de distribución como son las bases de datos, los programas fuentes etc.

Los componentes de ejecución: Son aquellos que se crean en tiempo de ejecución de forma dinámica, como los índices que crean los motores de búsqueda como resultado de alguna consulta.

A continuación se presenta el diagrama de componentes resultado de la vista de de implementación de la aplicación a desarrollar, el mismo muestra la distribución física del subsistema de modelado de DFI. Además ilustra la organización de los componentes y las dependencias que se establecen entre los mismos. Dentro de cada componente se encuentran las clases que los mismos realizan dentro de la aplicación en términos de componentes que poseerían estereotipo <<file>> ya que solo presenta un ejecutable.

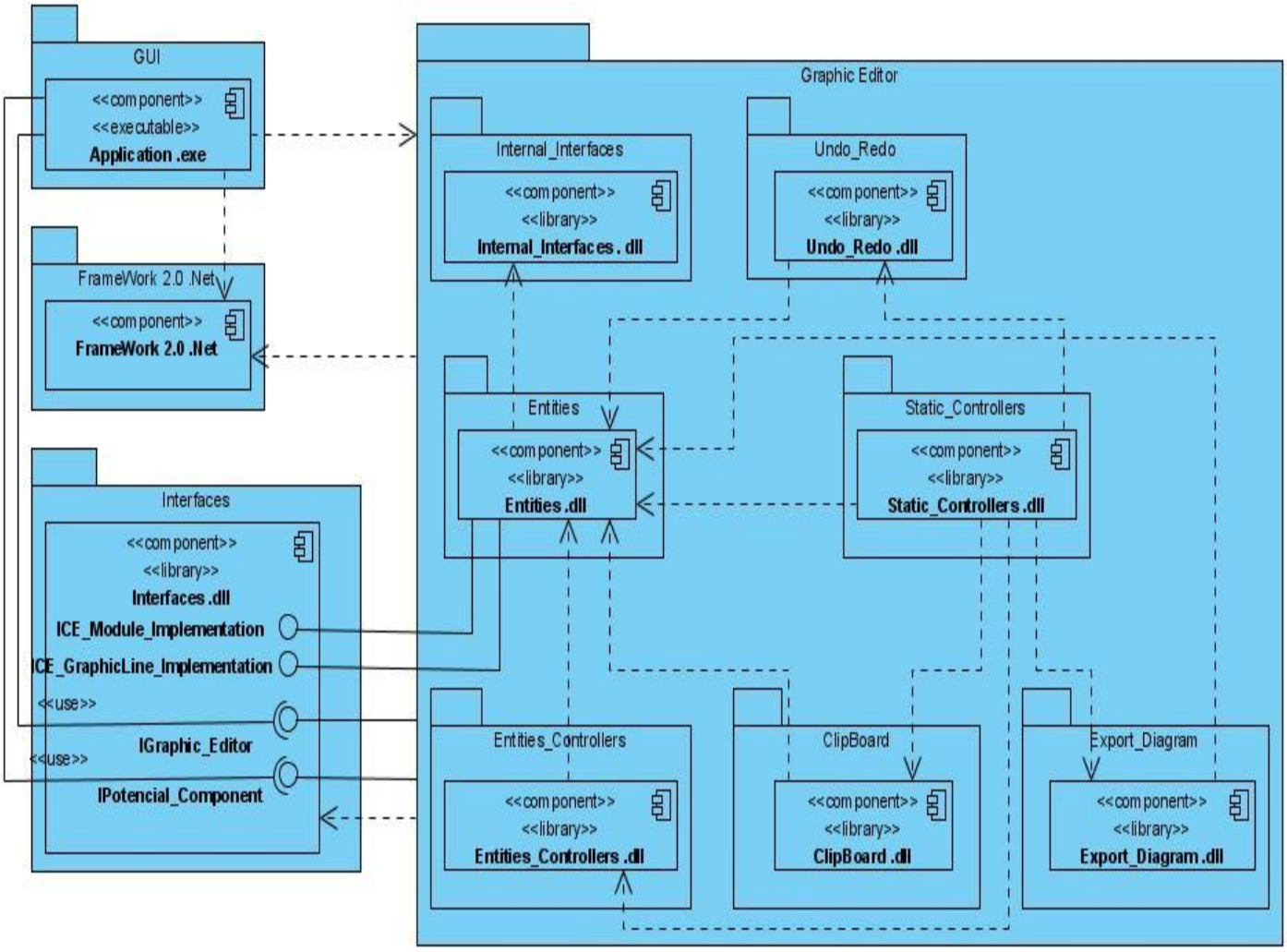

Figura 25: Diagrama de Componentes del Subsistema de modelado de DFI.

El subsistema a desarrollar se ha concebido como la integración de varias bibliotecas de enlace dinámico (dlls), las cuales se han realizado a partir de la agrupación de los componentes y clases según las funcionalidades que cumplen. Estas librerías podrán ser usadas por los subsistemas que importen los servicios del subsistema de modelado de DFI brindando además posibilidades de que se pueda de alguna manera redefinir la implementación de las operaciones brindadas. Se decidió utilizar un conjunto de librerías ya que permiten aislar las funcionalidades e independizar el código de cierta funcionalidad a la misma vez que se reutiliza el código. A continuación una breve descripción de las librerías desarrolladas junto a los componentes de trabajo más críticos dentro de las mismas:

Application.exe: Este componente constituye un ejecutable de la aplicación se encuentra dentro del paquete GUI (interfaz gráfica de usuario), que contiene además los componentes que forman parte de la presentación al usuario. Encapsula los elementos controles de .Net como son las ventanas de interacción con el usuario, los botones, menús, barras de estados y la paleta de componentes entre otros.

Framework 2.0 .NET: Componente que representa el Framework 2.0 de la plataforma .Net que el mismo debe estar instalado en la PC donde se instale la aplicación para garantizar el correcto funcionamiento de la misma.

Interfaces: Es una librería dinámica que contiene las interfaces que brinda el subsistema de modelado para la comunicación e integración con otros subsistemas.

Entities: Componente que contiene las clases entidades de la aplicación, son las clases que contienes los datos de los elementos gráficos, los módulos, las etiquetas o notas, los puertos que son contenidos por los módulos, las capas, conexiones y el editor gráfico como un todo entre otras.

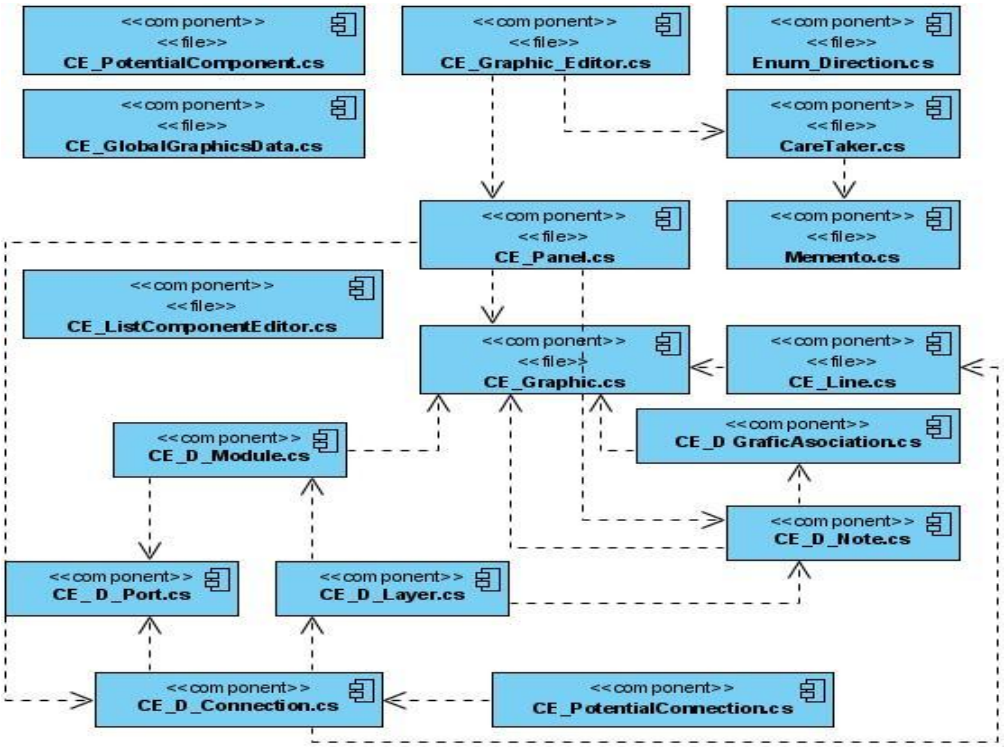

Figura 26: Realización del Componente Entities.dll

Entities_Controllers: Componente realiza las clases controladoras que se especializan en la realización de las operaciones de las clases entidades respectivas. Son estas clases las que realizan las operaciones sobre las entidades.

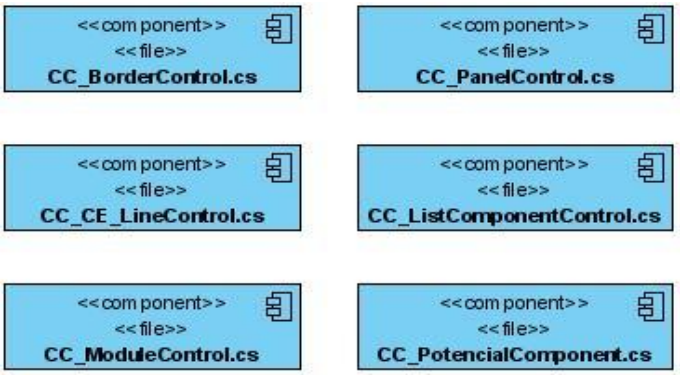

Figura 27: Realización del Componente EntitiesController.dll

Static_Controllers: Es una librería que contiene las clases controladoras que se encargan de la gestión de las funcionalidades del Diagrama de Flujo de Información en general como son las operaciones de Zoom y dibujo gráfico del DFI, además controla el flujo de información entre las diferentes librerías a modo de mediador entre los componentes.

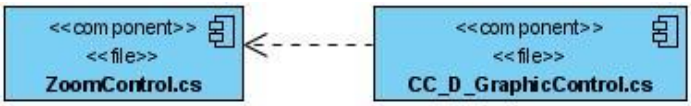

Figura 28: Realización del Componente Static_Controller.dll

Undo_Redo: Componente que contiene las clases que intervienen en la realización de la operación Deshacer y Rehacer acciones procesadas sobre la aplicación.

Figura 29: Realización del Componente Undo_Redo.dll

Export_Diagram: Es un componente que contiene las clases que intervienes en la realización de la operación de exportar el DFI a un determinado formato.

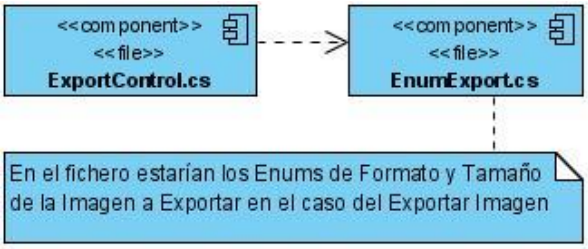

Figura 30: Realización del Componente Export_Diagram.dll

ClipBoard: Encapsula las funcionalidades de Copiar, Pegar y Cortar los elementos de un DFI a otro DFI o al mismo diagrama siempre que el núcleo de simulador permita la realización de la acción.

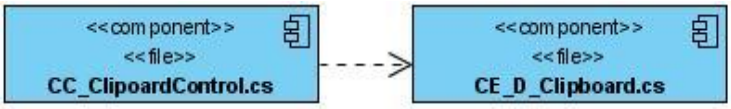

Figura 31: Realización del Componente ClipBoard.dll

Fundamentación del Estilo Arquitectónico

Como se mencionaba en la fase inicial del desarrollo de la aplicación, según sus características sería adecuado utilizar el estilo arquitectónico Tuberías y Filtros (ver epígrafe 1.13), combinado además con el estilo Llamadas y Retornos basado en la comunicación por métodos, procedimientos y subrutinas. La aplicación del Tubería y Filtros viene dada porque las imágenes a tratar son modificadas utilizando los componentes entidades como filtros y los mensajes y operaciones como tuberías, el estilo utilizado dentro de la aplicación es una modalidad del tuberías y filtros que combina el tuberías y filtros puro con el "tuberías y filtros degenerada" el primero se utilizará para mejorar en rendimiento a partir de la utilización de un buffer al que se le soliciten los datos y se vayan procesando y la segunda variante cuando se leen completamente los datos de entrada antes de producir algún dato en la salida. De esta manera se van modificando los datos de un procedimiento a otro encapsulados en los objetos que sufren modificaciones por los filtros para obtener finalmente el resultado deseado.

2.11. Vista de Despliegue

Los diagramas de despliegue muestran la configuración en funcionamiento del sistema, incluyendo su hardware y su software. Para cada componente de un diagrama de despliegue se deben documentar las

características técnicas requeridas, el tráfico de red esperado, el tiempo de respuesta requerido, entre otras, siempre teniendo en cuenta el tipo de aplicación que se desea desarrollar. En diferentes ocasiones el modelado de la vista de despliegue estática implica modelar la topología del hardware sobre el que se ejecuta el sistema. Los diagramas de despliegue son fundamentalmente diagramas de clases que se ocupan de modelar los nodos de un sistema. Aunque UML no es un lenguaje de especificación hardware de propósito general, se ha diseñado para modelar muchos de los aspectos hardware de un sistema a un nivel suficiente para que un ingeniero software pueda especificar la plataforma sobre la que se ejecuta el software del sistema y para que un ingeniero de sistemas pueda manejar la frontera entre el hardware y el software razonando sobre la topología de procesadores y dispositivos sobre los que se ejecuta el software.

Dentro de la aplicación objeto de la presente investigación no se consideró necesario realizar un diagrama de despliegue ya que el mismo constaría solo de un nodo procesador que constituiría una computadora personal (PC Cliente), es válido señalar que dentro de esta PC debe estar instalado el Framework 2.0 de la plataforma .Net ya que es necesario para el correcto funcionamiento de la aplicación que se desea desarrollar. Además esta PC deberá contener como periféricos mouse y teclado para el correcto funcionamiento de la aplicación. En esta PC deberá encontrarse instalado el sistema operativo Windows plataforma sobre la cual se desplegará la aplicación, debe poseer además tarjeta de memoria RAM 256 MB.

2.12. Resumen

En el presente capitulo se pudo observar cómo se distribuyó la aplicación en cuanto a sus funcionalidades, clases y componentes físicos. Se pudo especificar las vistas más importantes que caracterizan al subsistema que se desea desarrollar (Vista de CU, Vista Lógica, Vista de Implementación y Vista de Despliegue). En el capítulo se abordó detalladamente la propuesta arquitectónica describiendo en detalle cada una de sus vistas tratando de obtener o de cumplir con determinados elementos específicos conocidos como atributos de calidad. Dígase entonces que la arquitectura propuesta ha sido enfocada a obtener flexibilidad, reusabilidad del código y facilidad para el usuario final en el momento de realizar las operaciones que brinda el subsistema.

3. Capítulo 3: Evaluación de la Arquitectura

Introducción

En el presente capítulo se abordará la evaluación de la arquitectura propuesta como solución a la problemática existente en la presente investigación. Se describirán brevemente algunos de los métodos de evaluación de arquitectura, fundamentalmente se abordarán características y pasos de aplicación de los mismos. Se describirá además quienes son los implicados en estos métodos y las actividades que propone para los implicados cada uno de dichos métodos así como los atributos de calidad en que se centran los mismos. Finalmente se estimará el comportamiento del subsistema a desarrollar según los atributos de calidad que el mismo debe poseer.

3.1. Necesidad de Evaluar la Arquitectura

La arquitectura es un artefacto decisivo en la calidad de un software. La evaluación de la arquitectura permite mitigar 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. Desde etapas tempranas del proceso de desarrollo el arquitecto de software tiene la posibilidad de ir refinando la arquitectura, al definir la línea base de la misma se garantiza al menos el cumplimiento de los requerimientos funcionales básicos que debe poseer la aplicación al menos para una primera iteración, esta arquitectura será entonces refinada a través de todas las etapas de ciclo de vida del sistema, donde será sometida a prueba en varias ocasiones. Una arquitectura robusta debe cumplir con las funcionalidades del sistema y con los atributos de calidad esperados del mismo. La parte ligada a estos atributos de calidad son los requisitos no funcionales del sistema, los que poseen gran relevancia ya que especifican las características que se esperan del sistema, que sea robusto, fiable, que posea portabilidad, usabilidad, rendimiento, etc.

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 sí 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 que decisiones aplicar al proyecto que se desarrolla y de esta manera obtener un producto final sin fallas. [8]

3.2. Técnicas de Evaluación de Arquitectura

Las técnicas de evaluación de la arquitectura se clasifican en cualitativas (escenarios, cuestionarios y listas de chequeo) y cuantitativas (métricas, normas, máximos y mínimos teóricos, simulaciones, prototipos, etc.). Estos tipos de técnicas posibilitan evaluar una arquitectura y establecer comparaciones entre arquitecturas candidatas para determinar cuál satisface más un atributo de calidad específico. 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. Los resultados son obtenidos luego de comparar los valores de los atributos con los márgenes máximos y mínimos establecidos para evaluar la arquitectura candidata. Esta técnica tiene la desventaja de depender del conocimiento de los valores máximos y mínimos de la medición que se está empleando en la comparación, los cuales son difíciles de predecir.

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 [8]. Se plantean diferentes técnicas de evaluación tanto cualitativas como cuantitativas, entre estas se pueden encontrar: evaluación basada en escenarios, evaluación basada en simulación, evaluación basada en modelos matemáticos y evaluación basada en experiencia [3]. A continuación se muestra una breve descripción de las técnicas antes mencionadas:

3.2.1. Evaluación Basada en Escenarios

Un escenario es una breve descripción de la interacción de alguno de los involucrados en el desarrollo del sistema con el propio sistema. Por ejemplo, un usuario hará la descripción en términos de la ejecución de una tarea; un encargado de mantenimiento hará referencia a cambios que deban realizarse sobre el sistema; un desarrollador se enfocará en el uso de la arquitectura para efectos de su construcción o predicción de su desempeño. Un escenario consta de tres partes: el estímulo, el contexto y la respuesta. El estímulo es la parte del escenario que explica o describe lo que el involucrado en el desarrollo hace para iniciar la interacción con el sistema. Puede incluir ejecución de tareas, cambios en el sistema, ejecución de pruebas, configuración, etc. 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 permite establecer cuál es el atributo de calidad asociado. Los escenarios proveen un vehículo que permite concretar y entender atributos de calidad. Varios autores reconocidos en la industria de software a nivel mundial coinciden en la importancia del uso de los escenarios dada las ventajas de su uso, entre las cuales se encuentran:

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

Dentro de la evaluación por escenarios se utiliza lo que se conoce como el Utility Tree que no es más que 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. La intención del uso del Utility Tree es la identificación de los atributos de calidad más importantes para un proyecto particular. No existe un conjunto preestablecido de atributos, sino que son definidos por los involucrados en el desarrollo del sistema al momento de la construcción del árbol. El Utility Tree contiene como nodo raíz la utilidad general del sistema. Los atributos de calidad asociados al mismo conforman el segundo nivel del árbol los cuales se refinan hasta la obtención de un escenario lo suficientemente concreto para ser analizado y otorgarle prioridad a cada atributo considerado. [3] Cada atributo de calidad perteneciente al árbol contiene una serie de escenarios relacionados, y una escala de importancia y dificultad asociada, que será útil para efectos de la evaluación de la arquitectura.

También se utiliza lo que se conoce como Perfiles, que no es más que un conjunto de escenarios, generalmente con alguna importancia relativa asociada a cada uno de ellos. El uso de perfiles permite hacer especificaciones más precisas del requerimiento para un atributo de calidad. Para la definición de un perfil es necesario seguir tres pasos: definición de las categorías del escenario, selección y definición de los escenarios para la categoría y asignación del "peso" a los escenarios.

Cada atributo de calidad tiene un perfil asociado. Algunos perfiles pueden ser usados para evaluar más de un atributo de calidad. Esta técnica de evaluación basada en escenarios es dependiente de manera directa del perfil definido para el atributo de calidad que va a ser evaluado. La efectividad de la técnica es altamente dependiente de la representatividad de los escenarios.

3.2.2. 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. [3]

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. [3]

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. [3]

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. [3]

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. [3]

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 activaran los escenarios y se evaluaran en dependencia de los resultados los atributos de calidad esperados bajo ciertas condiciones impuestas básicamente para probar el desempeño de la aplicación.

3.2.3. 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.

3.2.4. Evaluación de Arquitectura Basada en Experiencia

Establece que en muchas ocasiones los arquitectos e ingenieros de software otorgan valiosas ideas que resultan de utilidad para la evaluación de decisiones erradas de diseño. Aunque todas estas experiencias se basan en evidencia anecdótica; es decir, basada en factores subjetivos como la intuición y la experiencia. Sin embargo, la mayoría de ellas puede ser justificada por una línea lógica de razonamiento, y pueden ser la base de otros enfoques de evaluación. Existen dos tipos de evaluación basada en experiencia: la evaluación informal, que es realizada por los arquitectos de software durante el proceso de diseño, y la realizada por equipos externos de evaluación de arquitecturas.

3.3. Métodos de Análisis de Arquitectura

Los métodos de análisis 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. Entre estos métodos fundamentalmente se encuentran:

3.3.1. Método de Análisis de Arquitectura de Software (SAAM)

El método de análisis de Arquitectura de Software (Software Architecture Analysis Method) fue el primero ampliamente documentado entre los métodos de evaluación. El mismo fue originalmente creado para el análisis de la modificabilidad de una arquitectura, pero en la práctica ha demostrado ser muy útil para evaluar de forma rápida distintos atributos de calidad, tales como modificabilidad, portabilidad, escalabilidad e integrabilidad.

La aplicación del método arroja resultados relacionados mayormente con la proyección sobre la arquitectura de los escenarios que representan los cambios posibles ante los que puede estar expuesto el sistema. Además evalúa el entendimiento de la funcionalidad del sistema, e incluso permite la comparación de múltiples arquitecturas con respecto al nivel de funcionalidad que cada una soporta sin modificación.

Con la aplicación de este método, si el objetivo de la evaluación es una sola arquitectura, se obtienen los lugares en los que la misma puede fallar, en términos de los requerimientos de modificabilidad. Para el caso en el que se cuenta con varias arquitecturas candidatas, el método produce una escala relativa que permite observar qué opción satisface mejor los requerimientos de calidad con la menor cantidad de modificaciones. Consta de seis pasos básicos, donde se desarrollan los escenarios, se describe la arquitectura, los escenarios son priorizados y evaluados y finalmente se emite una evaluación global de los mismos.

3.3.2. Método de Revisión Intermedio de Diseño (ARID)

El método de revisión intermedio de diseño (Active Reviews for Intermediate Design), es conveniente cuando se posee un diseño parcialmente completo que se desea evaluar. Es un método que permite en las fases tempranas del diseño realizar una evaluación cualitativa basada en escenarios que permite analizar las debilidades en cuanto a riesgos de la arquitectura. Es un híbrido que posee lo mejor del ATAM combinado con el método Active Design Review (ADR) este último utilizado para la evaluación de diseños detallados de unidades del software como los componentes o módulos. Las preguntas giran en torno a la calidad y completitud de la documentación y la suficiencia, el ajuste y la conveniencia de los servicios que provee el diseño propuesto.

Tanto ADR como ATAM proveen características útiles para el problema de la evaluación de diseños preliminares. En el caso de ADR, los involucrados reciben documentación detallada y completan cuestionarios, cada uno por separado. En el caso de ATAM, está orientado a la evaluación de toda una arquitectura. De ADR, resulta conveniente la fidelidad de las respuestas que se obtienes de los involucrados en el desarrollo. Así mismo, la idea del uso de escenarios generados por los involucrados con el sistema es tomada del ATAM. De la combinación de ambas filosofías surge ARID, para efecto de la evaluación temprana de los diseños de una arquitectura de software. El ARID propone dos etapas una de revisión:

Fase 1: Pre-Reunión o Actividades Previas	
Identificar los encargados de la revisión.	Los encargados de la revisión son los ingenieros de software que se espera que usen el diseño, y todos los involucrados en el diseño. En este punto, converge el concepto de encargado de revisión de ADR e involucrado del ATAM.
Preparar el informe del diseño	El diseñador prepara un informe que explica el diseño. Se incluyen ejemplos del uso del mismo para la resolución de problemas reales. Esto permite al facilitador anticipar el tipo de preguntas posibles, así como identificar áreas en las que la presentación puede ser mejorada.
Preparar los escenarios bases	El diseñador y el facilitador preparan un conjunto de escenarios base. De forma similar a los escenarios del ATAM y el SAAM, se diseñan para ilustrar el concepto de escenario, que pueden o no ser utilizados para efectos de la evaluación.
Preparar los materiales	Se reproducen los materiales preparados para ser presentados en la segunda fase. Se establece la reunión, y los involucrados son invitados.
Fase 2: Revisión	

Presentación de ARID	Se explica los pasos del ARID a los participantes.
Presentación del diseño	El líder del equipo de diseño realiza una presentación, con ejemplos incluidos. Se propone evitar preguntas que conciernen a la implementación o argumentación, así como alternativas de diseño. El objetivo es verificar que el diseño es conveniente.
Lluvia de ideas y establecimiento de imágenes	Se establece una sesión para la lluvia de ideas sobre los escenarios y el establecimiento de prioridad de escenarios. Los involucrados proponen escenarios a ser usados en el diseño para resolver problemas que esperan encontrar. Luego, los escenarios son sometidos a votación, y se utilizan los que resultan ganadores para hacer pruebas sobre el diseño.
Aplicación de los escenarios	Comenzando con el escenario que contó con más votos, el facilitador solicita pseudo-código que utiliza el diseño para proveer el servicio, y el diseñador no debe ayudar en esta tarea. Este paso continúa hasta que ocurra alguno de los siguientes eventos: Se agota el tiempo destinado a la revisión, Se han estudiado los escenarios de más alta prioridad, El grupo se siente satisfecho con la conclusión alcanzada. Puede suceder que el diseño presentado sea conveniente, con la exitosa aplicación de los escenarios, o por el contrario, no conveniente, cuando el grupo encuentra problemas o deficiencias.
Presentación de resultados	Al final, el facilitador recuenta la lista de puntos tratados, pide opiniones de los participantes sobre la eficiencia del ejercicio de revisión.

Tabla 3: Pasos del método ARID fuente propia

Entre los resultados arrojados por método pueden encontrarse fundamentalmente:

1. La arquitectura enfoques documentados,
2. El conjunto de escenarios y sus prioridades
3. El árbol de utilidad
4. Los riesgos descubiertos
5. Los no riesgos documentados
6. Los puntos de sensibilidad y los acuerdos de puntos encontrados

3.3.3. Método de Análisis de Acuerdos de Arquitectura de Software (ATAM)

El método de Análisis de Acuerdos de Arquitectura de Software está inspirado en tres áreas distintas: Los estilos arquitectónicos, el análisis de atributos de calidad y el método de evaluación SAAM, explicado anteriormente. El nombre del método ATAM (Architecture Trade-off Analysis Method) surge del hecho de que revela la forma en que una arquitectura específica satisface ciertos atributos de calidad, y provee una visión de cómo los atributos de calidad interactúan con otros; esto es, los tipos de acuerdos que se establecen entre ellos. El método se concentra en la identificación de los estilos arquitectónicos o enfoques arquitectónicos utilizados. Apoya a los involucrados con el proyecto a entender las consecuencias de las decisiones arquitectónicas tomadas con respecto a los atributos de calidad. Es un método que puede aplicarse en fases tempranas del desarrollo del software. Los implicados en la evaluación de la arquitectura a partir de este método son: Un equipo de evaluación, otro equipo que poseen el poder de poder decisiones sobre la arquitectura y por ultimo un equipo cuyos miembros registran el interés en la arquitectura porque su trabajo depende de la arquitectura del proyecto. Donde cada equipo juega un rol y realiza actividades con el fin evaluar los atributos de calidad. Consta de una serie de fases fundamentales donde se desarrollan las actividades. Los pasos de aplicación se muestran a continuación:

Fase 1: Presentación	
1. Presentación del	El líder de evaluación describe el método a los participantes, trata de establecer

ATAM.	las expectativas y responde las preguntas propuestas.
2. Presentación de las metas del negocio.	Se realiza la descripción de las metas del negocio que motivan el esfuerzo, y aclara que se persiguen objetivos de tipo arquitectónico.
3. Presentación de la arquitectura.	El arquitecto describe la arquitectura, enfocándose en cómo ésta cumple con los objetivos del negocio.
Fase 2: Investigación y análisis	
4. Identificación de los enfoques arquitectónicos	Estos elementos son detectados, pero no analizados.
5. Generación del Utility Tree.	Se elicitan los atributos de calidad que engloban la “utilidad” del sistema (desempeño, disponibilidad, seguridad, modificabilidad, usabilidad, etc.), especificados en forma de escenarios. Se anotan los estímulos y respuestas, así como se establece la prioridad entre ellos.
6. Análisis de los enfoques arquitectónicos	Con base en los resultados del establecimiento de prioridades del paso anterior, se analizan los elementos del paso 4. En este paso se identifican riesgos arquitectónicos, puntos de sensibilidad y puntos de balance.
Fase 3: Pruebas	
7. Lluvia de ideas y establecimiento de prioridad.	Con la colaboración de todos los involucrados, se complementa el conjunto de escenarios.

8. Análisis de los enfoques arquitectónicos	Este paso repite las actividades del paso 6, haciendo uso de los resultados del paso 7. Los escenarios son considerados como casos de prueba para confirmar el análisis realizado hasta el momento.
Fase 4: Reporte	
9. Presentación de los resultados	Basado en la información recolectada a lo largo de la evaluación del ATAM, se presentan los hallazgos a los participantes.

Tabla 4 : Pasos del método ATAM fuente [9] y propia.

3.4. Evaluación de la Arquitectura Propuesta:

Dado que dentro del proyecto de desarrollo de la aplicación no existe o no se dispone del personal calificado para evaluar la arquitectura propuesta a partir de los roles que define el método ATAM seleccionado para esta operación por la necesidad existente de evaluar la arquitectura y poseer el ATAM las características medibles para evaluar la arquitectura. Se consideró que el arquitecto de software 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. Se consideró además omitir los pasos de la primera fase del método dada la realidad de que el arquitecto de software entiende los procesos descritos en la misma ya que ha sido parte del equipo de desarrollo de la aplicación así como la omisión de las presentaciones del método a utilizar por las mismas razones antes expuestas. Dando paso entonces a las partes más importantes del método que serían las definiciones de diferentes escenarios donde se evidenciará el comportamiento de los atributos de calidad así como la utilización del Utility Tree para dar prioridad a los atributos de calidad evaluados.

Se consideró conveniente antes de pasar a la definición de los escenarios establecer cuales atributos de calidad se requieren satisfacer teniendo en cuenta que los mismos se basan fundamentalmente en los requerimientos no funcionales del software requerido.

3.4.1. Atributos de Calidad

Funcionalidad:

Según la industria arquitectónica predominante en el mundo y el SEI no existe una formalidad de que la funcionalidad sea un requisito de calidad, según el libro Software Architecture in Practice Second Edition una de las bibliografías más consultadas dentro de la industria arquitectónica de software, los requisitos de calidad están más ligados a los requerimientos no funcionales. La funcionalidad es “ortogonal” a estos requerimientos no funcionales de ahí que su mayor influencia deberá estar concentrada en la seguridad, el rendimiento de la aplicación, usabilidad ya que en dependencia del tipo de aplicación y las condiciones del mismo será la respuesta de esa funcionalidad. Entonces se podría estructurar la pregunta:

¿Qué es la funcionalidad?: Es la capacidad del sistema para hacer el trabajo para los que fue concebido. Una tarea que requiere de muchos o la mayoría de los elementos del sistema de manera coordinada para completar el trabajo [9]. Por lo tanto, si a los elementos no han sido asignados responsabilidades de manera correcta o no se han dotado de instalaciones para la correcta coordinación con otros elementos, el sistema no será capaz de ofrecer la funcionalidad requerida.

Rendimiento (Performance):

El rendimiento de una aplicación en general implica tanto las dependencias arquitectónicas como las dependencias no arquitectónicas [9]. En el primer caso depende de la comunicación establecida entre los diferentes componentes del sistema, lo que se ha asignado en cuanto a recursos a cada uno así como la distribución de los recursos compartidos. Y en el caso de las dependencias no arquitectónicas se refiere a lo que concierne al código de la aplicación, los algoritmos de realización de determinadas funcionalidades.

Modificalidad (Modifiability):

Un sistema es modificable en caso de que los cambios afecten el menor número posible de diferentes elementos.

Está relacionado principalmente con los siguientes atributos:

Maintainability: Relacionado con la resolución de problemas y mantenimiento al software que se desea desarrollar.

Extensibility: Relacionado con la posibilidad de poder extender el software con nuevas funcionalidades.

Restructuring: Relacionado con la reorganización de módulos.

Interoperability: Relacionado con la integración que posea el sistema con otros sistemas.

Usabilidad:

La usabilidad de un sistema consta de dos partes fundamentales una que no se relaciona directamente con la arquitectura que es lo referente a la interfaz de usuario, que la misma sea intuitiva, fácil de utilizar que debe presentar una botón en una posición determinada una pantalla de un tipo determinado, son cuestiones que se relacionan más con el diseño de la aplicación. La arquitectura comienza a jugar su papel en la usabilidad cuando e brindan operaciones donde a partir de los datos cargados se puede cancelar una operación determinada inclusive comenzada la misma sin riesgo de perder los datos ya procesados, cuando se brindan operaciones de Deshacer y Rehacer acciones, acciones de este tipo que reutilizan datos se pueden considerar de arquitectura ya que dentro de las mismas interactúan mayormente varios elementos.

3.4.2. Atributos de Calidad seleccionados como críticos para la evaluación del subsistema de modelado de DFI:

Rendimiento:

Se necesita que el editor gráfico no exceda en tiempo de respuesta 0,3 segundos para adicionar un componente a partir de la paleta de componentes, sobre el panel de edición de DFI. Además el consumo de la RAM de la PC cliente donde será desplegado el sistema deberá ser menos de 80 MB debido al consumo de los recursos en el almacenamiento de las instancias en memoria dado que el tratamiento de imágenes es costoso en este sentido.

Usabilidad:

Se requiere de sistema que brinde la posibilidad de cancelar operaciones a partir de datos cargados o de datos de entrada, además que brinde operaciones de hacer y deshacer acciones así como que brinde información.

Modificabilidad:

El sistema debe ser modificable, permitir la agregación de nuevas funcionalidades, reutilización de los métodos definidos por las clases controladoras así como la integración con sistemas heredados que puedan interactuar con el subsistema de modelado de DFI.

Presentación de la Arquitectura: La arquitectura está basada en el modelo Tuberías y Filtros Degenerada, posee una estructura en paquetes y subsistemas que brindan las operaciones necesarias para la realización de los requerimientos no funcionales y funcionales de la aplicación.

Árbol de Utilidad del subsistema de modelado de DFI:

Atributo de Calidad	Refinamiento del Atributo Evaluado	Escenarios
Rendimiento	Tiempo de respuesta de adicionar un componente	El Especialista acciona el click del mouse en la paleta de componentes y luego sobre el panel bajo un entorno normal el sistema dibuja el componente antes de los 0,2 seg. (A,B)
	Tiempo de respuesta bajo sobrecarga	El Especialista Adiciona un componente. Bajo las condiciones de estado de sobrecarga, uso de la memoria RAM con alrededor de 150 instancias de los objetos que poseen las imágenes a cargar en el editor con demanda de 1 solo editor realizando peticiones, PC 256 MB RAM. El editor dibuja el componente en un tiempo menor a 1,0 segundo. (A, A).
Usabilidad	Ayuda al Usuario	Un usuario desea realizar una conexión y no sabe cómo, busca ayuda en el manual de usuario y el sistema brinda la información requerida. (A, B).
	Operaciones Hacer y Deshacer	El especialista accidentalmente crea un componente bajo la ejecución normal del sistema, el sistema brinda la operación de

		Deshacer para corregir dicho error cometido. (A, A).
	Utilización de datos cargados y posibilidad de cancelar operaciones	El especialista accidentalmente acciona la operación cerrar editor gráfico, el editor pregunta si realmente desea cerrar dando la posibilidad de salvar los datos o cancelar en este momento. (A, L).
Modificabilidad	Extensibilidad	Un desarrollador desea adicionar nuevas operaciones a un paquete o modulo del sistema. La operación debe estar adicionada un tiempo menor a las 3 h. (A,M)
		Un desarrollador desea adicionar un nuevo tipo de componente con el que desea trabajar dentro del editor. (A, M).
	Interoperabilidad	Un desarrollador desea utilizar el sistema para la edición de diagramas dentro de otro sistema que posee el mismo. La integración de los sistemas debe quedar realizada en un tiempo menor de 5 semanas. (A, M).
Portabilidad	Multiplataforma	Existe necesidad de cambiar la plataforma sobre la cual corre el sistema, se desea correr sobre otro sistema operativo manteniendo la arquitectura X86 del microprocesador. (A, A).

Definición de los principales escenarios.

Los principales escenarios serán aquellos que tengan según el arquitecto una prioridad alta y además poseen dificultad alta de implementación.

Fuente del estímulo	Se trata de alguna entidad, ser humano, sistema u otra entidad que interactué con el sistema.
Estímulo	Condición que necesita ser considerada cuando arriba al sistema.
Entorno	El estimulo se produce cuando el sistema se encuentra bajo ciertas condiciones que se toman en cuenta para valorar el atributo de calidad evaluado.
Componentes o Artefactos	Son aquellos componentes que son afectados bajo el escenario representado.
Respuesta	Actividad realizada después de que el sistema es estimulado.
Medida de la Respuesta	Cuando el sistema emita alguna respuesta esta debe ser medida de alguna manera para que el requerimiento pueda ser aprobado.

Tabla 5: Definición General de un Escenario

Escenario # 1 Rendimiento en estado sobrecargado.

Fuente del estímulo	El Especialista acciona el click del mouse en la paleta de componentes y luego sobre el panel.
Estímulo	Evento click del mouse sobre el panel con una demanda períodica de arribo de eventos.
Entorno	Ejecución en estado de sobrecarga de uso de la memoria RAM con alrededor de 150 instancias de los objetos que poseen las imágenes a cargar en el editor con demanda de 1 solo editor realizando peticiones, PC con 256 MB RAM.

Componentes o Artefactos	Todo el sistema.
Respuesta	El editor dibuja el componente seleccionado.
Medida de la Respuesta	El componente es dibujado en el panel en un tiempo en el rango de 0,2 a 0,3 segundos.

Tabla 6: Escenario de Rendimiento en estado de Sobrecarga.

Escenario # 2 Usabilidad, Operaciones de Hacer y Deshacer.

Fuente del estímulo	El Especialista accidentalmente elimina unos de los componentes del diagrama y los desea recuperar.
Estímulo	Evento click del mouse sobre la acción Deshacer acción.
Entorno	Ejecución Normal.
Componentes o Artefactos	Componente de Entidades y Componente de Undo-Redo.
Respuesta	El sistema restablece el estado del editor una vez requerida la acción Deshacer por el Especialista.
Medida de la Respuesta	El componente será restaurado en el panel del DFI antes de transcurridos 0,5 seg.

Tabla 7: Escenario Usabilidad

Escenario # 3 Portabilidad.

Fuente del estímulo	Se desarrolla el núcleo del simulador en un sistema operativo diferente de Windows dígase Linux.
Estímulo	Se desea utilizar los servicios del subsistema de modelado de DFI para modelar los diagramas del simulador con el nuevo núcleo desarrollado.
Entorno	Ejecución sobre el sistema operativo Linux.
Componentes o Artefactos	Sistema
Respuesta	El sistema no funciona correctamente debido a que no ha sido concebido para funcionar en varias plataformas.
Medida de la Respuesta	

Tabla 8: Escenario Portabilidad

Escenario # 3 Extensibilidad.

Fuente del estímulo	Se desarrolla el Reportador del Simulador y se desea adicionar funcionalidades al Editor Gráfico para Editar los DFI y además personalizar los reportes.
Estímulo	Se intenta establecer comunicación entre un nuevo módulo de componentes, Reportador.
Entorno	Ejecución normal.

Componentes o Artefactos	Sistema
Respuesta	La arquitectura permite adicionar el módulo Reportador y las operaciones requeridas sin necesidad de realizar grandes cambios en la Arquitectura.
Medida de la Respuesta	Los desarrolladores deben terminar de integrar los módulos en un tiempo de 3h.

Tabla 9: Escenario Extensibilidad.

Evaluación:

Riesgos: Constituye un riesgo y una desventaja en hecho de que la aplicación no sea multiplataforma. A pesar de que es poco probable que la plataforma de la aplicación deje de ser Windows es importante destacar que si en algún momento dado el avance del software libre se decidiera emigrar la aplicación se convertiría esto en un problema que ya es un riesgo y que se ven afectados los atributos de calidad como la portabilidad del sistema desarrollado.

Táctica y recomendación: Investigación sobre el potencial de Mono tomando en cuentas las características de la aplicación a desarrollar, valoración futura de migrar a dicha plataforma compatible con .Net donde se pueden desarrollar aplicaciones multiplataforma.

No Riegos: El rendimiento de la aplicación a partir de las restricciones de hardware impuestas para la aplicación no constituyen riesgo, la aplicación se mantiene dentro de los rangos aceptables de tiempo de respuesta bajo sobrecarga, a medida que aumente el número de editores pues mayor será el consumo de los recursos de la PC. Según las pruebas realizadas bajo sobrecarga el sistema responde en un intervalo de tiempo entre los 0,2 y 0,3 segundos.

Dado que el sistema deberá correr en una PC con 256 MB de memoria según las pruebas realizadas en estado de sobrecarga con un mínimo de 100 instancias en un editor activo, el consumo de memoria RAM promedió alrededor de los 23 MB con imágenes de tamaño máximo de 400KB, tomando en cuenta la cantidad de procesos, se considera que el consumo de memoria RAM no constituye un riesgo al no ser

que se instancien un promedio de 80 editores activos con al menos 100 instancias donde deberá lógicamente disminuir el tiempo de respuesta y sobrecargarse el uso del CPU y la memorias RAM del PC cliente.

La modificabilidad de la aplicación cumple con las expectativas prevista ya que se brinda un paquete que contiene las interfaces e comunicación con otros subsistemas que puedan comunicarse con el subsistema desarrollado. Además existen clases abstractas propiciadas para la herencia de los métodos y atributos necesarios para utilizar el subsistema de modelado de DFI.

La extensibilidad de la aplicación demuestra que la necesidad de integrar el subsistema a una interfaz gráfica independiente queda solventada. Se brinda un mecanismo de interfaz utilizada por dicha interfaz gráfica que permite el conocimiento de los componentes que se encuentran en la paleta lo que permite desde el subsistema de modelado de DFI conocer el tipo de componente que se desea adicionar al DFI.

Puntos de Sensibilidad:

Dentro de la aplicación desarrollada existen puntos de sensibilidad que traen como consecuencia el que no exista un equilibrio en los atributos de calidad en este caso Usabilidad y Rendimiento. Se brindan posibilidades de operaciones Hacer y Deshace pero las mismas afectan el rendimiento de manera considerable ya que se hace una salva de la información luego de cada acción que realizó el usuario final sin poder eliminar los cambios producidos por estas acciones. Existe necesidad de llevar una traza de las acciones sin afectar el rendimiento de la aplicación en cuanto a consumo de la memoria RAM.

Táctica: Se almacenará en memoria solo los datos que hayan sido modificados de un estado a otro estado dentro del editor y no un nuevo estado del mismo conforme se encontraba a hora de realizar determinada operación. Se utilizaran iteradores del lenguaje C# dadas las ventajas de rendimiento que los mismos brindan a la hora de seleccionar los estados en una lista los mismo no necesitan iterar toda las lista de estados sino que los mismo se ejecutan para una llamada desde el mismo estado o posición donde quedaron como resultado de la llamada anterior utilizando las ventajas del **yield** dentro del lenguaje de programación. De esta manera resulta más óptimo solicitar los estados siguiente o anterior del editor gráfico.

3.5. Utilización de elementos de la arquitectura propuesta en otras soluciones conocidas:

Existen muchas aplicaciones desarrolladas con el estilo de arquitectura tuberías y filtros que es la base de la propuesta de arquitectura, es un estilo muy conocido de ahí que no sea necesario como se dice dentro de la industria de la ingeniería de software “inventar la rueda” sino que es un estilo simplista que fue de los primeros que se utilizaron dentro de la industria de la arquitectura y que tiene sus ventajas y desventajas las cuales son también ampliamente conocidas. Dentro de las aplicaciones de mayor peso que se pueden encontrar donde se utilice dicho estilo arquitectónico está lo que se conoce como KHOROS, un entorno de desarrollo científico que entre otras cosas posee un potente lenguaje de Programación Visual Orientada a Flujo de Datos [10]. Desarrollado inicialmente por el Departamento de Ingeniería de Computación y Electrónica de la Universidad de Nuevo México, en el año 1987; su primera versión estaba orientada únicamente al Tratamiento Digital de Imágenes. KHOROS es un entorno científico donde se ha empleado el estilo arquitectónico Tuberías y Filtros para la confección de su primera versión de ahí que se demuestre la idoneidad del mismo para el tratamiento de imágenes. KHOROS posee ventajas como:

Rutinas de procesamiento de imágenes y cálculo matricial.

Utilidades de visualización interactiva de imágenes, animaciones, gráficas 2D y 3D, etc.

Herramientas de clasificación interactiva de señales e imágenes.

Rutinas de visualización 3D y `renderizado de imágenes'.

La idea de citar KHOROS no es establecer una comparación irreal entre las arquitecturas sino resaltar que el estilo arquitectónico de Tuberías y Filtros adoptado en la propuesta de la arquitectura para el subsistema de modelado de DFI, ha sido utilizado en entornos de tratamiento de imágenes que han alcanzado un grado alto de calidad y aceptación en el mercado mundial. Las características de este estilo son ventajosas para el tipo de aplicación que se desea desarrollar.

Graph Editing Framework (GEF) es otro de los productos que se utiliza en la representación de elementos gráficos. Es un Framework que permite realizar modificaciones a los gráficos, cambiarle el tamaño, moverlos y conectarlos utilizando además puertos y curvas o líneas para realizar esta conexión [11]. Este Framework esta desarrollado en java potente lenguaje para el tratamiento de los gráficos. Si es cierto que

el mismo utiliza como estilo arquitectónico patrón Modelo Vista Controlador y existe la vista de casos de uso del proyecto GEF es muy similar a la propuesta dentro de la arquitectura del subsistema de modelado de DFI. GEF es una librería desarrollada en java que puede ser utilizada dentro de un programa para un dominio específico. Los CU de GEF concuerdan en cuanto al modelado de la parte gráfica es decir conectan gráficos a partir de puertos entre otras cosas seleccionan elementos, dibujan elementos y ambos se pueden definir para un sistema concreto, dentro de la vista lógica poseen clases y controladoras y entidades similares a las clases propuestas en la presente investigación. Dígase que el proyecto GEF posee una clase Editor, una clase Capa además existen clases que gestionan los datos de esas clases interactuando según el MVC. En general la arquitectura del subsistema de modelado de DFI de una forma u otra posee similitud y diferencias con otras similares en el resto del mundo.

3.6. Conclusiones:

En el presente capítulo se analizaron brevemente los atributos de calidad de la arquitectura propuesta. Se arrojaron los resultados y se propusieron tácticas para mitigar los riesgos encontrados en la evaluación de la arquitectura. Se explicaron algunos de los métodos de evaluación de arquitectura y se valoró la aplicación de uno de ellos a la solución propuesta. El método seleccionado fue ATAM sufriendo modificaciones que favorecen el arribo temprano a resultados acerca del comportamiento de los atributos de calidad en la propuesta arquitectónica del subsistema de modelado de DFI. Es válido destacar que los atributos de calidad que significan riesgos dentro de la aplicación no son de gran importancia ya que son atributos que dados los escenarios definidos arrojan resultados negativos pero que no afectan la finalidad con la que fue concebida la aplicación desde el planteamiento del objetivo general de la investigación. Pudiéndose concluir planteando que la arquitectura propuesta cumple de manera satisfactoria los requerimientos de calidad críticos del subsistema de modelado de Diagramas de Flujo de Información.

CONCLUSIONES DE LA INVESTIGACIÓN

Conclusiones Generales

Al finalizar la investigación se pudo arribar a las siguientes conclusiones:

La definición de la línea base de la arquitectura de manera correcta, permitió adicionar a través del ciclo de desarrollo de la aplicación, las funcionalidades del subsistema de modelado de DFI para el simulador DFISim sin necesidad de realizar grandes cambios en la arquitectura.

La validación de la arquitectura permitió conocer a grandes rasgos como se comportará el sistema ante las acciones del cliente final, permitiendo afirmar la arquitectura propuesta cumple con los requerimientos funcionales y no funcionales que debe poseer el subsistema de modelado de DFI.

La idea a defender planteada como propuesta de solución inmediata al problema científico de la investigación, ha quedado validada de manera satisfactoria dados los resultados de la descripción de la arquitectura y el comportamiento de los atributos de calidad de la solución propuesta, dándole cumplimiento de esta manera al objetivo general de la investigación.

Recomendaciones

Se recomienda:

Como parte de la estrategia desarrollada por la Universidad de las Ciencias Informáticas y por el país para lograr la mayor utilización de herramientas libres a modo de contrarrestar el monopolio de las compañías comercializadoras de software propietario y de las licencias del software privativo así como para logar una mayor calidad del producto en cuanto a Portabilidad, se recomienda:

Investigar profundamente sobre las potencialidades de la plataforma Mono en cuanto al tratamiento de imágenes, con el objetivo de desarrollar la aplicación sobre la misma con el fin de lograr que la aplicación pueda ser utilizada bajo la licencia de software libre y ejecutarse sobre entornos Linux.

Desarrollar la aplicación utilizando una arquitectura orientada a eventos completamente ya que permitiría un mayor desacoplamiento las clases y paquetes del diseño propuesto y en general entre los componentes funcionales de la de arquitectura.

Referencias:

1. Vitoria, H.E.P.d.A. and R.A.H. León (2005) Informatización de la ingeniería de proceso en la industria química cubana. Volume, 5-7

2. Reynoso, C.B. (2004) Introducción a la Arquitectura de Software. Volume, 1-10

3. Camacho, E., F. Cordeso, and G. Nuñez (2004) Arquitectura de Software. Volume, 1-54

4. Buschmann, F., et al., PATTERN-ORIENTED SOFTWARE ARCHITECTURE, A system of Patterns, WILEY, Editor. 2000.

5. Larman, C., UML y PATONES. Introducción al análisis y diseño orientado a objetos. , P. Hall, Editor. 1999.

6. Ferg, S. (2006) Event-Driven Programming. Volume,

7. Center, I.T.J.W.R., Introduction to Design Patterns in C#, ed. J.W. Coper. 2002.

8. Armas, E.M.d. and E.O. Fonseca, Sistema Informático para la Red Nacional de Genética Médica: Definición de la Arquitectura de Software, in Facultad 6 2008, Universidad de las Ciencias Informáticas: Ciudad de la Habana. p. 105.

9. Bass, L., P. Clements, and R. Kazman, Software Architecture in Practice, Second Edition. 2003, Addison Wesley.

10. López, M.J.L., J.M.F. García, and N.R. Reyes. KHOROS 2.1: UN SISTEMA DE DESARROLLO DE SOFTWARE CIENTÍFICO. 1997 [citado 2009 5/5]; disponible en: http://www.mappinginteractivo.com/plantilla-ante.asp?id_articulo=851.

11. Tools, T.o.O.S.S.E. Graph Editor Framework. 2001 [citado 2009 5/5]; disponible en: http://gef.tigris.org/.

Bibliografía Consultada:

- León, R.A.H. and S.C. Hernández, EL PARADIGMA CUANTITATIVO DE LA INVESTIGACIÓN CIENTÍFICA, EDUNIV, Editor. 2002.
- Sampiery, R.H., C.F. Collado, and P.B. Lucio, Metodología de la Investigación 1998.
- Larman, C., UML y PATONES. Introducción al análisis y diseño orientado a objetos. , P. Hall, Editor. 1999.
- Rumbaugth, J., I. Jacobson, and G. Booch, El lenguaje Unificado de Modelado. Manual de Referencia, A. Wesley, Editor. 2000.
- Sierra, A.A. Las Metodologías Ágiles en la enseñanza de la Ingeniería de software. 2003 [citado 2008 17/12].
- Bass, L., P. Clements, and R. Kazman, Software Architecture in Practice, Second Edition. 2003, Addison Wesley.
- Krutchten, P., et al., Integrating Software-Architecture-Centric Methods into the Rational Unified Process 2004, Carnegie Mellon Software Engineering Institute.
- Flower, M., et al., PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE. 2002.
- Kazman, R and Nord, R.L., Integrating Architecture Methods: The Case of the Rational Unified Process, 2004.
- Som, G., U. Zorrilla, and J. Serrano, Curso de Introducción a .Net con C#. 2006.
- Buardes, J.M. (2003) Informática Gráfica. Volume, 2-6.
- Kasman, R. Klein, M. Clements, P. ATAM: Method for Architecture Evaluation, Carnegie Mellon University, 2000. Páginas 13-23.

- Passeriny, N. Brey, G.A, Coco, G. Atributos de Calidad/ Tácticas. Universidad Tecnológica Nacional, Facultad Regional Departamento de Sistemas, 2006.
- Colectivo de Autores, Grupo de Investigación en Ingeniería de Software, La Importancia de la Arquitectura en el desarrollo de Software de Calidad, 2005.
- Kasman, R. Klein, M. Clements, P., Bass, L., Models for Evaluating and Improving Architecture Competence, Carnegie Mellon, 2008.