UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

Título: Informe del rol de arquitecto del subsistema de modelos y propiedades de un simulador para la industria química.

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

Autor: Felix Duviel Berta Hernández

Tutor: Ing. Juan Carlos Suarez López

Ciudad de la Habana, 2009.

"Año 50 de la Revolución"

"El aspecto fundamental en el cual la juventud debe señalar el camino es precisamente en el aspecto de ser vanguardia en cada uno de los trabajos que le compete."

Ernesto Guevara

DATOS DE CONTACTO

Tutor:

Ing. Juan Carlos Suarez López

E-mail: jslopez@uci.cu

Universidad de las Ciencias Informáticas.

AGRADECIMIENTOS

A mi madre por estar siempre pendiente y apoyarme en todas mis decisiones.

A mi abuela Cuca que se lo debo todo en esta vida.

A mis tíos Leidys y Luis, por ser como unos padres para mí.

A Nereida, Olivia y Andrés por todos sus cuidados, algo que nunca olvidaré.

A mis abuelos Jesús, Mima y Pipo y a todo el familión del condado.

A mi familia en general.

Agradecer a Adnelis por estar siempre ahí cuando lo necesité, por soportarme a pesar de mi carácter y permanecer a mi lado en las buenas y en las malas.

A todos los amigos, los de siempre y los de la universidad, a los que están y los que no, por todos los momentos compartidos.

Agradecimientos a todos por hacer realidad este sueño.

DEDICATORIA

En la vida, muchas veces nos encontramos desorientados y sin saber qué hacer, puede que te encuentres en esta situación, pero siempre que puedas contar tu familia nunca te sentirás solo. Por eso, por todo el apoyo y amor que me han brindado, quisiera dedicarle este logro, especialmente a cuatro mujeres y un hombre de mi familia:

A mi madre por brindarme su apoyo y amor incondicional, ella es la amiga en la cual siempre podré confiar.

A "Cuqui la Mora" por ser más que una madre para mí y sufrir mis éxitos y fracasos. Es esa mujer sin la cual no sé donde estaría en estos momentos.

A mi tía Leidy por ser mi otra madre y diseñadora personal.

A mi amiga, mi confidente, mi otra mitad: Adnelis, por darme su cariño y estar siempre "ahí" cuando lo necesité.

A mi abuelo Enrique por ser un ejemplo para mí en todos los sentidos. Un hombre en el que todo era bueno y que la vida no le permitió verme graduado.

A todos gracias por hacer de mí una persona mejor.

Duviel

RESUMEN

En los últimos años, la simulación de procesos ha llegado a ser un medio adecuado, oportuno y de apoyo para el diseño, caracterización, optimización y monitoreo del funcionamiento de procesos industriales y además su uso se está extendiendo en las instituciones de formación de ingenieros.

Cuba no se ha quedado ajena al resto del mundo en cuanto al empleo de estas potentes herramientas. En la mayor de las Antillas, como a nivel mundial, la simulación está siendo utilizada con fines económicos, por ejemplo, en la industria química se simula los procesos de las producciones de azúcar crudo y refino, alcohol, ácido sulfúrico, sosa cáustica, refinación de petróleo, níquel y cobalto, entre otros.

En la actualidad la competitividad en las empresas ha hecho que el dinamismo presente en los simuladores sea un factor determinante y mientras más procesos permitan simular la herramienta, más potente será. Detrás de todo este desarrollo se encuentra la arquitectura de software y es en este punto hacia donde estará enfocado el presente trabajo.

Para ello se tratarán aspectos teóricos tales como arquitectura de software, los arquitectos y los diferentes estilos y patrones arquitectónicos. Todo ello con el fin de lograr una arquitectura de software lo suficientemente robusta y que permita el funcionamiento óptimo del subsistema de: modelos matemáticos, componentes químicos y propiedades físicos químicas de un simulador de procesos para la industria química.

Palabras Claves: simuladores, arquitectura, estilos, patrones, modelos matemáticos, componentes químicos y propiedades físico-químicas

Contenido

FIGURAS

TABLAS

INTRODUCCIÓN

¿Qué se entiende por simulación?

"La simulación es reproducir el ambiente, las variables (rasgos, apariencia, características, contexto) de un sistema real. Es imitar una situación del mundo real en forma matemática. La simulación constituye una técnica económica que nos permite ofrecer varios escenarios posibles de una situación y nos permite equivocarnos sin provocar efectos sobre el mundo real (por ejemplo un simulador de vuelo o conducción)."(PARRA, 2008)

Definición Estricta (H.Maisel y G.Gnugnoli)

"Simulación es una técnica numérica para realizar experimentos en una computadora digital. Estos experimentos involucran ciertos tipos de modelos matemáticos y lógicos que describen el comportamiento de sistemas de negocios, económicos, sociales, biológicos, físicos o químicos a través de largos periodos de tiempo."(GNUGNOLI, 2004)

Otra Definición (Robert E. Shannon)

"Simulación es el proceso de diseñar y desarrollar un modelo computarizado de un sistema o proceso y conducir experimentos con este modelo con el propósito de entender el comportamiento del sistema o evaluar varias estrategias con las cuales se puede operar el sistema."(SHANNON, 2006)

Simulación de Procesos

"Se describe la simulación de procesos de una empresa mediante la generación de modelos matemáticos y estadísticos, como una herramienta para la adecuada gestión y administración de la organización, puesto que entre sus principales ventajas, está el que evite incurrir en los costos que implica una práctica real para conocer el funcionamiento y los posibles problemas de dichos procesos."(GARAVITO, 2008)

Enunciados algunos conceptos de la simulación se pueden destacar aspectos importantes que se evidencian en estas definiciones y que han dejado claro que simular consta de modelos matemáticos, para describir y evaluar el comportamiento de situaciones o sistemas del mundo real, que nos eviten incurrir en equivocaciones que puedan traducirse en pérdidas para la sociedad, ya sean económicas o de otro tipo.

Variadas han sido las propuestas de aplicaciones informáticas a través de los años. En este sentido, las empresas han entrado al mercado con productos cada vez más novedosos y llamativos, ya sea por su interfaz gráfica o facilidad de uso, pero sobre todo por sus modelos matemáticos y el análisis de resultados, además de contar con una sólida y bien estructurada arquitectura que soporte toda la lógica de cálculo que implica un simulador. En 1974 aparece el primer simulador de procesos químicos, el FLOWTRAN. A partir de allí se vienen presentando simuladores como el POWERFACTS de la Dow Chemical, GPSS II, CSL y CHIPS de IBM; GASP y GPS de la Corporación del Acero de USA; CHEOPS de la Compañía Petrolera Shell, entre otros.

En la actualidad con la aparición de Windows se han desarrollado Simuladores de Procesos Químicos de propósito general muy potentes, siendo los más conocidos y populares los siguientes: ASPEN PLUS, CHEMCAD, HYSYS, SUPERPRO DESIGN y otros cuyo costo oscila entre los cinco mil y treinta mil dólares por lo que la adquisición de ellos es muy limitada.

Cuba tampoco se ha quedado exenta del resto del mundo en este sentido. La simulación de procesos químicos en la mayor de las Antillas se ha centrado, en la industria azucarera, la producción de etanol y la refinación de petróleo. Al MINAZ por ejemplo, se han llevado simuladores como el AGE, pero este se basa en métodos de cálculo simplificado ó rápido que no permiten obtener todos los resultados necesarios que pueden dar los simuladores que usan modelos más precisos y sin simplificaciones innecesarias. Otro ejemplo es el SIMFAD, pero el mismo está orientado a ecuaciones, por lo que se necesita una mejor inicialización de las variables y al presentar una mayor complejidad, existe una menor confiabilidad en los resultados y más problemas de convergencia, además de hacerse más difícil su empleo por los “no especialistas”.

Todo lo logrado en este sentido está limitado fuertemente porque el software cubano carece de varias facilidades, sobre todo de las necesarias para el análisis y la síntesis de los resultados de las simulaciones

de procesos complejos. Además de que el software extranjero solo puede usarse en Cuba e internacionalmente para fines académicos.

En cuanto al desarrollo de la simulación en las universidades del país, se encuentran el simulador de procesos tecnológicos concebido en la Universidad Central de Las Villas (UCLV), pero al igual que el AGE sus métodos eran demasiado simplificados. Por su parte en la Ciudad Universitaria José Antonio Echeverría (CUJAE) se lograron avances con el Termo Azúcar el mismo contaba con un poco menos de deficiencias que los anteriores realizados en esta rama, pero todavía se encontraba por debajo de sus similares extranjeros. Este producto fue implementado para su uso en Windows en la Universidad de las Ciencias Informáticas (UCI), pero el simulador sigue estando muy orientado a la industria azucarera, por lo que se hace muy difícil encontrar componentes de otro tipo de empresas como son los sulfatos, óxidos y demás componentes inorgánicos, debido a que no cuenta con una base de datos de compuestos, sino que por el contrario, están predefinidas las sustancias que se van a emplear específicamente para este tipo de simulación.

De ahí que surge la **situación problemática** para la presente investigación porque no existe una arquitectura de software para un simulador de procesos de la industria química, que erradique las deficiencias de: solo estar desarrollado para propósitos específicos y no posibilitar un soporte general a la simulación de procesos, que hagan más extensible el simulador al dar la opción de personalizar los compuestos, para formar nuevas mezclas y así ofrecerle al usuario la posibilidad de tener una gran variedad de bibliotecas para la simulación.

Por todo lo anterior se plantea como **problema científico** la inexistencia de la arquitectura de los componentes que garanticen el funcionamiento del subsistema de: modelos matemáticos, componentes químicos y propiedades físico-químicas en el simulador de procesos para la industria química.

Por lo que se define como **objeto de estudio** el proceso de desarrollo de una arquitectura de software para un simulador de procesos de la industria química.

Delimitando así el **campo de acción**, a la arquitectura del paquete de modelos matemáticos, componentes químicos y propiedades físico-químicos de un simulador de procesos para la industria química.

Este trabajo plantea como **objetivo general** desarrollar la arquitectura del paquete de modelos matemáticos, componentes químicos y propiedades físico-químico de un simulador de procesos para la industria química. De acuerdo con esta propuesta, se trazaron los siguientes **objetivos específicos**:

- Definir el estilo arquitectónico a emplear.
- Describir la arquitectura del subsistema de modelos matemáticos y propiedades.
- Establecer una estrategia para lograr la calidad de la arquitectura diseñada.

Se ha propuesto como **idea a defender** que si se implementa una correcta arquitectura de software, se podrá garantizar el funcionamiento del subsistema de modelos matemáticos, componentes químicos y propiedades físico-químicas en el desarrollo de un simulador para la industria química.

Para cumplir con los objetivos propuestos y resolver la situación problemática se tienen las siguientes **tareas.**

- Realizar un estudio del estado del arte de los simuladores de procesos existentes para sentar las bases de la investigación.
- Caracterizar las tendencias arquitectónicas actuales sobre simuladores, para seleccionar la que más se adecue al subsistema de modelos y propiedades.
- Analizar los requerimientos del simulador.
- Seleccionar los patrones arquitectónicos a emplear.
- Confeccionar la propuesta de arquitectura para el simulador.
- Evaluar la arquitectura diseñada.

Como **métodos de investigación científica se han propuesto:**

-Teóricos

- **Análisis Histórico-Lógico:** El método al que se hace referencia es empleado para analizar la evolución de la arquitectura de software para simuladores en la universidad y comenzar trabajar en base a errores cometidos anteriormente en este sentido.

- **Analítico-Sintético:** Es empleado para estudiar toda la información referente a la arquitectura de software para simuladores y así analizar los documentos y la teoría existente con el fin de extraer los elementos más importantes relacionados con el tema y constatar su relación.

- **Modelación**: Permite la creación de modelos, los cuales representan una vista simplificada de los procesos, posibilitando con esto describir el objeto de estudio.

La estructuración del documento viene dada por un **primer capítulo** donde se abordan elementos relacionados con la fundamentación teórica y el estado del arte. En el mismo se hace un estudio de los simuladores de procesos para la industria química, además de tratar conceptos claves, como arquitectura de software, arquitecto de software, las principales tendencias arquitectónicas y los patrones arquitectónicos. Teniendo presente que la metodología de software seleccionada, cumpla con las características necesarias y suficientes para lograr una robusta arquitectura que se adecue al subsistema de modelos y propiedades. En el **segundo capítulo** se realiza la descripción de la arquitectura propuesta a través de las vistas arquitectónicas, considerando para ello los requerimientos que debe cumplir el sistema. En el **tercer capítulo** se lleva a cabo un análisis de los resultados de acuerdo a las normas de calidad.

CAPÍTULO I: ESTADO DEL ARTE Y FUNDAMENTACIÓN TEÓRICA

En este capítulo la investigación se ha centrado en el análisis de los simuladores de procesos químicos y la arquitectura de software, así como de las diversas tendencias arquitectónicas y las principales funciones del arquitecto, partiendo de conceptos esenciales. Haciendo referencia a las metodologías existentes para el desarrollo de software, caracterizando la candidata a emplearse.

El mundo en que nacemos, vivimos y morimos es sin lugar a dudas cada vez más competitivo tecnológicamente hablando, esto se ha extendido a las diferentes esferas de producción y a la industria química moderna, la cual no se ha quedado detrás en este sentido. Es por ello que la búsqueda de una mayor competitividad hace indispensable el constante perfeccionamiento de los procesos, en especial los aspectos energéticos; componente determinante en los costos de operación. Una de las vías más exitosas para lograr lo anterior es el desarrollo de software para la simulación, análisis, síntesis y control de procesos apoyado en técnicas de inteligencia artificial, de integración de procesos y minería de datos.

1.1. DEFINICIONES ASOCIADAS

1.1.1. Simulación

La simulación de procesos se ha convertido en una de las herramientas más importantes dentro de muchas empresas. En líneas básicas, permite representar un proceso mediante otro que lo hace mucho más simple o entendible. Según Eric de Perthius, director general para España y Portugal de Dassault Systems, esta tecnología “es una representación de la realidad a través de las herramientas informáticas que permite además, hacer una predicción de movimientos, cargas, comportamientos, etc. Para Perthius, estas soluciones incluyen aplicaciones para el análisis de elementos finitos y soluciones metafísicas, que permiten dotar a los ingenieros de métodos para facilitar la comprensión de los retos que se les plantean y sus posibles soluciones.” (CORDÓN, 2008)

“Simulación es una técnica numérica para realizar experimentos en una computadora digital. Estos experimentos involucran ciertos tipos de modelos matemáticos y lógicos que describen el comportamiento de sistemas de negocios, económicos, sociales, biológicos, físicos o químicos a través de largos periodos de tiempo.” (GNUGNOLI, 2004)

Pudiera resumirse entonces que la simulación es imitar una situación del mundo real en forma matemática.

1.1.2. Proceso

Proceso: “Someter algo a una serie de transformaciones físicas, químicas o biológicas; tratar los datos o materiales que se poseen para obtener nuevos datos o conclusiones.” (SOLUTIONS, 2008)

“Un Proceso de Software consiste en una serie de actividades que garantizan, técnica y administrativamente, que un software pueda ser desarrollado de manera organizada, disciplinada y previsible.” (INFANTE, 2003)

1.1.3. Simulación de Procesos

“Se describe la simulación de procesos de una empresa mediante la generación de modelos matemáticos y estadísticos, como una herramienta para la adecuada gestión y administración de la organización, puesto que entre sus principales ventajas, está el que evite incurrir en los costos que implica una práctica real para conocer el funcionamiento y los posibles problemas de dichos procesos.”(GARAVITO, 2008)

“La simulación de procesos es definida como una técnica para evaluar en forma rápida un proceso con base a una representación del mismo, mediante modelos matemáticos.” (VERA MESTANZA 2009)

Simular consiste en el diseño del modelo matemático de un sistema, y la posterior ejecución de una serie de experimentos con la intención de entender su comportamiento bajo ciertas condiciones. El modelo debe ser capaz de reproducir el comportamiento del proceso real con la mayor exactitud posible.

La **simulación de procesos** se ha convertido en una herramienta indispensable para el desarrollo de la industria, y además de poder hacer el trabajo de ingeniería más rápidamente y con mayor calidad ofrece ventajas tales como:

a) Incrementar la eficiencia y eficacia del proceso.

b) Reconciliar los datos del proceso obtenidos en el laboratorio, especialmente cuando el muestreo no es en línea, con los resultados de las simulaciones.

c) Suministrar información sobre corrientes que no son medidas.

d) Organizar y ejecutar de forma más eficiente las evaluaciones de equipos, subprocesos y del proceso en su totalidad.

e) Evaluar propuestas de inversiones ahorrando el dinero para financiarlas y a la vez crear mayores ganancias.

f) Analizar la factibilidad de los proyectos de investigación e innovación.

g) Aumentar de forma significativa el conocimiento de los ingenieros de fábrica sobre el proceso, aspecto especialmente importante cuando estos son jóvenes.

1.1.4. Estructura de una simulación

Metas y límites del sistema: Como la simulación es un proceso condicionado y generalmente visual, es muy importante establecer las metas de forma clara y específica desde un inicio. Deben también determinarse los límites del sistema, recordando que generalmente tanto en arquitectura como en construcción se manejan sistemas complejos y es aquí cuando estos límites toman sentido.

Hipótesis y condiciones límite: Cada simulación se basa en hipótesis del comportamiento posible de un sistema, por ello es importante plantear claramente dichas hipótesis. Las condiciones límite se determinan mediante preguntas de investigación que definirán de alguna manera el curso que se seguirá en la simulación.

Modelos físicos y lógicos: Esta etapa de la representación de la realidad es quizá el centro de la simulación. El modelo debe corresponder con los requerimientos y expectativas de aquellos que diseñan o construyen.

Solución matemática, algoritmos: Como los modelos se convierten generalmente en forma de grandes cantidades de cálculos matemáticos, la selección de los procesos de solución tiene un cierto sentido.

Implementación de cómputo: Esta implementación, la programación propiamente dicha, depende del lenguaje.

Estructura de los datos, importación / exportación: Las estructuras de datos van tomando cada vez una mayor importancia, por un lado permiten representar fácilmente hechos y por otro lado facilitan la sustitución de datos con otras aplicaciones.

Interfaz de entrada / salida: La entrada de datos es muy laboriosa ya que la consistencia de los datos de entrada debe ser comprobada.

Validación, interpretación de resultados: Los resultados del modelo de simulación deben ser comprobados para la confiabilidad del modelo y la posibilidad de perfeccionar su implementación. Existen procedimientos para validar un modelo de simulación ó un programa de simulación.

La simulación como instrumento de diseño tiene muchas alternativas, desde los cálculos simples, pasando por los modelos a escala, hasta los modelos computarizados. Su uso es una forma de representar la realidad antes de enfrentarla propiamente, con las ventajas de ajustarla a escenarios límite que simulen las condiciones de diseño propuestas, variándolas cuantas veces sea necesario hasta obtener los resultados deseados.

Quizá los modelos de simulación que más se están desarrollando son los modelos computarizados por su versatilidad, exactitud, facilidad de implementación y manejo visual. Los modelos a escala implican gran cantidad de mano de obra y costo y los modelos matemáticos son muy abstractos y especializados.

1.1.5. EXPERIENCIAS PRECEDENTES

1.1.5.1. Simuladores

Puede decirse que desde la década de los años 70 se desarrollaron y aplicaron simuladores como el POWERFACTS de la Dow Chemical, GPSS II, CSL y CHIPS de IBM; GASP y GPS de la Corporación del Acero de USA; CHEOPS de la Compañía Petrolera Shell, Flexible FLOWSHEET de la Corporación Kellogg, PEDLAN de la Compañía Petrolera MOBIL. También en esos años en el sector académico de Canadá y USA se crearon el PACER y el SPEEDUP. Algunos de los simuladores mencionados dieron lugar a nuevas versiones o nuevos desarrollos, que los implicados en el tema de la simulación del Instituto Superior Pedagógico José Antonio Echeverría (ISPJAE) han utilizado en distintos momentos, como el

GEMCS, GASP II, CHEMCAD, HYSYS, SUGARS, ASPEN PLUS y otros. Los primeros libros dedicados a la simulación de procesos fueron publicados por la editorial Wiley & Sons a partir de 1967 y como su contenido fundamental era la modelación y estrategia de simulación mantienen actualmente su vigencia.

Los simuladores más importantes, son sobre todo productos de transnacionales vinculadas a la industria petroquímica y la IBM, primer monopolio de la producción de "hard" y "software"; así como de universidades de primer nivel mundial.

La aplicación de simuladores para la industria química en Cuba comenzó en el ISPJAE en el año 73 bajo la tutoría de uno de los autores del libro, ya mencionado, "Chemical Plant Simulation" y de los creadores del sistema de programas, denominado "General Engineering and Management Computation System (GEMCS)" desarrollado en la Universidad de McMaster, Canadá.

El GEMCS y el Termo azúcar

En el periodo 1973 - 74 el GEMCS se modificó para poder aplicarlo al proceso azucarero y las computadoras disponibles en Cuba, lo que se logró inicialmente con docentes del ISPJAE.

También existen otros simuladores y aplicaciones a procesos completos o subprocesos en Cuba como los relacionados con la producción de etanol, refinación de petróleo y de sulfato ferroso. En el ICIDCA se dispone de un Simulador, orientado a ecuaciones, del proceso azucarero y de alcohol SIMFAD 3.0, resultado de todo un amplio trabajo anterior de modelación matemática y simulación de equipos y subprocesos de la industria azucarera. En la Universidad Central de las Villas (UCLV) se desarrolló un simulador de procesos tecnológicos. Otro simulador desarrollado en el MINAZ (Villa clara) es el denominado AGE de carácter determinístico y que contiene recomendaciones de cómo actuar en función de los resultados. Pero al igual que otro paquete de simulación de la UCLV, está basado en métodos de cálculo simplificados ó rápidos que, en opinión de los participantes del ISPJAE en este proyecto, no permiten obtener todos los resultados importantes y necesarios que pueden dar los simuladores que usan modelos más precisos y sin simplificaciones innecesarias si se dispone de computadoras.

En el caso de la Universidad de las Ciencias Informáticas se han dado pasos en cuanto a simulación se refiere, pero está más bien encaminada a la realidad virtual, tal es el caso de SIMPRO y el simulador quirúrgico. No ha sido tanto así en cuanto a simulación de procesos, como experiencia precedente se

encuentra el personal que trabajó como programadores en el proyecto para la simulación de los procesos industriales de azúcar, alcohol, refinación de petróleo, níquel-cobalto y cerveza, los cuales estaban asesorados por la Facultad de Ingeniería Química de la Ciudad Universitaria José Antonio Echeverría (CUJAE), que era la encargada de prepararlos, por lo que este proyecto que implementó el simulador actual, será tomado como base para el futuro desarrollo de la arquitectura de software para el subsistema de modelos y propiedades del simulador de procesos que se quiere desarrollar.

1.2. SITUACIÓN ACTUAL DE LA SIMULACIÓN

Hoy en día, la simulación va más allá del mero modelado de sistemas de alto nivel y se centra más en la creación de prototipos de software, consiguiendo un ahorro de tiempo y dinero a las empresas en sus desarrollos. El sector energético y petroquímico también ha descubierto los beneficios que pueden aportar a la hora de desarrollar nuevos productos y soluciones innovadoras. Con este epígrafe se pretende analizar las características relevantes de algunos de los simuladores comerciales actuales, para desarrollar una arquitectura con calidad.

Algunos de los simuladores más importantes:

*** ASCEND * Aspen Plus * CHEMCAD * COCO simulator * Design II * Dymola * SimSci-Esscor DYNSIM * EMSO * Hysys * GIBBSim * gPROMS * Open Modélica * Petro-SIM * SimSci-Esscor PRO/II * Pro Max * SysCAD * VMGSim * UniSim Design**

1.2.1. Características de los simuladores comerciales.

El estudio de los simuladores actuales es necesario en la realización de la arquitectura del subsistema de modelos matemáticos y propiedades para un simulador de la industria química. Por lo que la misma debe estar preparada para dar soporte a varios de los aspectos que a continuación se mencionan en los diferentes productos de software descritos en este epígrafe.

CHemCAD: Aspectos importantes: Incluye bases de datos de componentes químicos, métodos termodinámicos y unidades de operación que permiten la simulación en estado estacionario de procesos químicos continuos desde escala laboratorio a escala industrial. Permite la simulación en régimen dinámico y de procesos discontinuos. Funcionamiento bajo Windows y brinda la posibilidad de incluir

código de programación en VISUAL BASIC. Puede ser utilizado en: petróleo, refinería, petroquímica, polímeros, química fina y farmacéutica, construcciones, ingeniería, etc. Actualmente está disponible la versión 6.0 con nuevas características y unidades de operación: hornos, cambiadores de calor de aire, separaciones por membrana, electrolitos para polímeros y aplicaciones para simular columnas.

ASCEND: Es un medio de modelos flexibles para la solución de difíciles problemas de ingeniería y la ciencia. Ofrece un modelo de descripción con un lenguaje orientado a objetos para la descripción de su sistema y un entorno de secuencias de comandos que le permite automatizar sus problemas más complejos de simulación. Incluye un potente y fiable solucionador de rutinas que analizan la estructura de su modelo y puede resolver miles de ecuaciones no lineales simultáneas en unos segundos en la vida cotidiana de hardware. Se encuentra bajo desarrollo activo y está licenciado bajo la GNU General Public License, por lo que sería un buen ejemplo para el trabajo a desarrollar, si se quiere que sea multiplataforma.

ASPEN PLUS: Es un simulador de procesos ampliamente utilizado en las industrias químicas y petroquímicas. Posee una librería de modelos que incluye la gran mayoría de los equipos típicos de estas industrias así como los modelos termodinámicos y las bases de propiedades físico-químicas. Presenta un algoritmo de cálculo basado en el método secuencial modular, desarrollando además el método orientado a ecuaciones. Posibilitando también un entorno de simulación modular tanto para estado estacionario, como para régimen dinámico.
Es un simulador bidireccional, ya que el flujo de información va en dos direcciones (hacia delante y hacia atrás). De esta forma, puede calcular las condiciones de una corriente de entrada a una operación a partir de las correspondientes a la corriente de salida sin necesidad de cálculos iterativos. Permite la regresión de datos experimentales y el diseño preliminar de los diagramas de flujo usando modelos de equipos simplificados. Realiza balance de materia y energía rigurosas usando modelos de equipos detallados. Dimensiona piezas claves de los equipos.

HYSYS: Es Interactivo ya que permite realizar cálculos automáticos al aportar nueva información. Es abierto y extensible debido a sus unidades de operación en estado estacionario o dinámico. Permite especificar expresiones para reacciones cinéticas. Tiene paquetes de propiedades especiales. Presentando también extensión de la funcionalidad de la simulación en respuesta a posibles cambios.

COCO simulator: COCO es software libre, pero para uso en Windows. Cumple con el estándar CAPE-OPEN. Es empleado en simulaciones modulares secuenciales, por lo que reviste gran importancia para el desarrollo del presente trabajo. Ofrece un proceso químico de simulación para los estudiantes. Presenta un diagrama esquemático de entorno de modelado que permite que todos puedan añadir nuevas unidades de operación y paquetes termodinámicos. Cuenta con un banco de datos de compuestos químicos y con sus métodos de cálculo analítico o derivados permite la exportación de más de cien propiedades a su biblioteca termodinámica. El simulador supone un conjunto de unidades de operaciones tales como el flujo de divisores, mezcladores, intercambiadores de calor, compresores, bombas reactores, etc.

1.3. ARQUITECTURA DE SOTWARE

En los inicios de la informática, la programación se consideraba un arte y se desarrollaba como tal, debido a la dificultad que entrañaba para la mayoría de las personas, pero con el tiempo se han ido descubriendo y desarrollando formas y guías generales, en base a las cuales se puedan resolver los problemas. A estas, se les ha denominado Arquitectura de Software. En el libro "An introduction to Software Architecture", David Garlan y Mary Shaw definen que la Arquitectura es un nivel de diseño que hace foco en aspectos "más allá de los algoritmos y estructuras de datos de la computación; el diseño y especificación de la estructura global del sistema es un nuevo tipo de problema".(DAVID GARLAN 1994)

El concepto de arquitectura software incluye los aspectos estáticos y dinámicos más significativos del sistema. La arquitectura surge de las necesidades de la empresa, como las perciben los usuarios y los inversores, y se refleja en los casos de uso. Sin embargo, también se ve influida por muchos otros factores, como la plataforma en la que tiene que funcionar el software (arquitectura hardware, sistema operativo, sistema de gestión de base de datos, protocolos para comunicaciones en red).

La función corresponde a los casos de uso y la forma a la arquitectura. Debe haber interacción entre los casos de uso y la arquitectura. Por un lado, los casos de uso deben encajar en la arquitectura cuando se llevan a cabo. Por otro, la arquitectura debe permitir el desarrollo de todos los casos de uso requeridos, ahora y en el futuro. En realidad, tanto la arquitectura como los casos de uso deben evolucionar en paralelo.

1.3.1. ¿Qué es la arquitectura de software?

Han sido diversas las definiciones de arquitectura, y se le ha catalogado en ocasiones como “el diseño de más alto nivel de la estructura de un sistema.” (CASANOVAS, 2004)

Pressman plantea que “… es una descripción de los subsistemas y los componentes de un sistema informático y las relaciones entre ellos (...)” (PRESSMAN, 2006)

“La arquitectura de software tiene que ver con el diseño y la implementación de estructuras de alto nivel. Es el resultado de ensamblar un cierto número de elementos arquitectónicos de forma adecuada para satisfacer la mayor funcionalidad y requerimientos de desempeño de un sistema, así como requerimientos no funcionales como la confiabilidad, escalabilidad, portabilidad y disponibilidad. (KRUCHTEN, 1995)

“La definición oficial de Arquitectura del Software es la IEEE Std 1471-2000 que reza así: La Arquitectura del Software es la organización fundamental de un sistema formada por sus componentes, las relaciones entre ellos y el contexto en el que se implantarán, y los principios que orientan su diseño y evolución.” (CASANOVAS, 2004)

Una arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, trabajen en una línea común que permita alcanzar los objetivos del sistema de información, cubriendo todas las necesidades. Se selecciona y diseña con base en requerimientos y restricciones. Unas arquitecturas son más recomendables de implementar con ciertas tecnologías mientras que otras tecnologías no son aptas para determinadas arquitecturas. La arquitectura de software define, de manera abstracta, los componentes que llevan a cabo alguna tarea de computación, sus interfaces y la comunicación ente ellos.

La arquitectura por pasos.

Como un edificio, un sistema de software es una única entidad, pero al arquitecto del software y a los desarrolladores les resulta útil presentar el sistema desde diferentes perspectivas para comprender mejor el diseño. Estas perspectivas son vistas del modelo del sistema. Todas las vistas juntas representan la arquitectura. Incluyéndose decisiones importantes sobre:

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

Sin embargo, la arquitectura de software está afectada no sólo por la estructura y el comportamiento, sino también por el uso, la funcionalidad, el rendimiento, la flexibilidad, la reutilización, la facilidad de comprensión, las restricciones y compromisos económicos y tecnológicos, y la estética.

Se necesita una arquitectura para: Comprender el sistema. Organizar el desarrollo. Fomentar la reutilización. Hacer evolucionar el sistema. La relación siguiente hace una recopilación de los objetivos más importantes que debe cubrir un diseño de arquitectura:

• Dar cobertura a la funcionalidad: el diseño del software debe en primer lugar centrarse en dar cabida a todas las funcionalidades que se especifican para el simulador.

• Facilitar el desarrollo del proyecto de software: el software de un simulador puede ser un proyecto complejo donde intervengan docenas de personas y se extienda durante años. Es un objetivo crucial que el reparto de funciones entre los distintos módulos permita un desarrollo independiente, pero coordinado, de los mismos. La programación orientada a objetos y a componentes permite, por su filosofía de origen, una programación modular a la vez independiente pero coordinada.

• Optimizar el uso de los recursos del hardware: los recursos se refieren a la capacidad de cálculo de los procesadores, al uso de la memoria, al almacenamiento y acceso a los discos, a la capacidad de generación gráfica y a la capacidad de comunicaciones. Permitir la ínter confiabilidad de módulos para mejorar y ampliar las capacidades del software del simulador en el futuro es preciso que la arquitectura sea lo suficientemente abierta para poder sustituir un módulo sin que se vean afectados demasiados los demás.

• Facilitar el flujo de información para la simulación en tiempo real: esta es una de los retos más difíciles de una buena arquitectura. Requiere un gran esfuerzo minimizar el traspaso de datos entre las diferentes

partes del software sin que ninguna tenga déficit o exceso, que no haya problemas de sincronismo y evitar la saturación del hardware de comunicaciones.
Los aspectos que condicionan principalmente la arquitectura se pueden relacionar como sigue: Funcionalidad, Recursos de hardware, Capacidad del software y Capacidad presupuestaria.

1.4. ARQUITECTO DE SOTWARE. SUS RESPONSABILIDADES

El rol del arquitecto de software implica articular la visión arquitectónica, conceptualizando y experimentando con enfoques de arquitecturas alternativas, creando modelos, componentes y documentos de especificación de interfaces, validando la arquitectura contra los requerimientos.

“Un arquitecto de software es aquel que está involucrado con uno o varios desarrollos de software, pero desde un punto de vista macro. Es decir, comenzando por los requisitos (funcionales y no funcionales) y hasta llegar a conocer el impacto que tendrá en el hardware del cliente. El arquitecto de software tiene tres variables principales sobre cuales moverse: tiempo, satisfacción del cliente y, principalmente, costo de los desarrollos.” (PIZARRO, 2006)

El arquitecto tiene la visión global de los proyectos. Se desempeña gestionando los proyectos que se llevan a cabo y la tecnología donde deberá aplicarse. Debe ayudar a la implementación de la Metodología. Conocer a la perfección los requerimientos y restricciones. Alinearse con la Visión de la Organización. Asesorar en la Planificación y Estimación del proyecto. Definición de Estándares y políticas de reusabilidad de Componentes. Proveer una guía técnica clara y consistente.

Entre sus responsabilidades están:

Arquitectura: Definición de arquitectura de los sistemas, vista física, vista lógica, principios de arquitectura, seguridad, etc.

Selección de software: Pilas de aplicaciones, bases de datos, librerías, frameworks, estándares tecnológicos, etc.

Selección de Infraestructura: Sistemas Operativos, hardware, redes, sistemas de recuperación, etc.

Requisitos no Funcionales: Rendimiento, escalabilidad, seguridad.

Liderazgo: Liderazgo Técnico, responsabilidad y autoridad, dirección de equipos.

Coaching y Mentoring: Ayuda sobre problemas técnicos y en la evolución profesional.

Metodología de Proyectos: Estructura de Proyectos, Metodologías (Waterfall, Scrum, RUP, XP...).

Procesos de Desarrollo: Control de versiones de código fuente, procesos de construcción, integración continua, automatización de pruebas y otros procesos y herramientas de desarrollo.

Prácticas y Estándares: Estándares de codificación y libros blancos, selección de herramientas, entre otros.

Diseño, Desarrollo y Pruebas: Diagramas UML, codificación, pruebas unitarias.

Experiencia: Conocimiento sobre tecnologías y arquitecturas.

Estar al día en cuanto a tendencias tecnológicas: Agiles, Web 2.0, SOA, Lightweight Java EE.

Ocho hábitos presentes en Arquitectos exitosos:

1. Mantener las cosas simples
2. Dejar a otros defender la arquitectura
3. Actuar, no discutir
4. Mantener la concentración en la visión
5. Estar predispuesto al cambio, pero cuidado con los cambios fuertes de una sola vez.
6. Aprender cuando parar.
7. Saber cómo seguir.
8. Ser consciente de su papel de mentor.

Principales errores de los Arquitectos: Creer que los requerimientos están claros y no cambian o refinan. Dejarse seducir por tecnologías. Comunicación errónea de la arquitectura. Creer que sabe todo y

demostrarlo. Todo reusable (sobre diseño). Perder el contacto con la tecnología. No confiar en los desarrolladores.

Considerando lo anterior, se puede observar que el rol del arquitecto de software es crítico y sumamente importante, puesto que requiere de una gran variedad de conocimientos, tales como: ingeniería de requerimientos, teoría de arquitecturas de software, codificación, tecnologías de desarrollo, plataformas de hardware y software.

1.5. ESTILOS Y PATRONES ARQUITECTÓNICOS

De manera concreta, al diseñar una arquitectura de software debemos crear y representar componentes que interactúen entre ellos y tengan asignadas tareas específicas, además de organizarlos de forma tal que se logren los requerimientos establecidos. Podemos partir con patrones de soluciones ya probadas, con la intención de no comenzar de cero las propuestas y utilizar modelos que han funcionado. Estas soluciones probadas se conocen como estilos arquitectónicos, patrones arquitectónicos y patrones de diseño, que van de lo general a lo particular. “Un estilo arquitectónico consiste en una colección de tipos de componentes con una descripción del patrón o interacción a través de ellos “(P. CLEMENTS, 2003).

Para Pressman, un estilo describe “una categoría de sistema que abarca un conjunto de componentes que realizan una función requerida por el sistema, un conjunto de conectores que posibilitan la comunicación, la coordinación y cooperación entre los componentes, las restricciones que definen como se integran los componentes para conformar el sistema, y los modelos semánticos que facilitan al diseñador el entendimiento de todas las partes del sistema. El estilo arquitectónico es también un patrón de construcción.” (PRESSMAN, 2002)

“El estilo afecta a toda la arquitectura de software y puede combinarse en la propuesta de solución. Un patrón arquitectónico se enfoca a dar solución a un problema en específico, de un atributo de calidad, y abarca solo parte de la arquitectura”(P. CLEMENTS, 2003) .

Un patrón de diseño ayuda a diseñar la estructura interna de un componente específico, es decir, su detalle. Aunque estos estilos y patrones se pueden adoptar, también pueden adaptarse con objeto de lograr alguna funcionalidad concreta esperada.

1.5.1. ESTILOS

Los estilos sirven para sintetizar estructuras de soluciones. Pocos estilos abstractos encapsulan una enorme variedad de configuraciones concretas. Definen los patrones posibles de las aplicaciones. Permiten evaluar arquitecturas alternativas con ventajas y desventajas conocidas ante diferentes conjuntos de requerimientos no funcionales.

1.5.1.1. Clasificación General de los Estilos

- **Sistemas Basados en Flujos de Datos: (**Filtro-Tubería) (Secuencial por Lotes)
- **Sistemas de Llamada/Retorno (Call/Return):** (Sistema Modular) (Arquitectura Orientada a Objetos) (Jerarquía de Capas) (Model-View-Controller (MVC))
- **Componentes Independientes:** (Procesos en Comunicación) (Sistemas de Eventos)
- **Máquinas Virtuales:** (Intérpretes) (Sistemas basados en conocimiento (KBS))
- **Sistemas de Control:** (Lazo Abierto o Cerrado) (Sistemas Centrados en Datos (repositorios)) (Bases de Datos) (Sistemas de Hipertexto) (Modelo REST) (Arquitectura de Pizarra)
- **Estilos Peer-to-Peer:** (Arquitecturas Basadas en Eventos) (Arquitecturas Orientadas a Servicios (SOA)) (Arquitecturas Basadas en Recursos)

1.5.1.2. Características de algunos de los estilos más importantes.

Arquitectura basada en eventos: Impide incurrir en el modelo de aplicaciones que preguntan si sucedió algo. Generan la ejecución apenas ocurre el evento o el usuario se conecta.

Ventajas: Simplicidad: Fácil programación. Evolución: se pueden reemplazar componentes suscriptores. Modularidad: una sola modalidad para eventos diversos. Puede mejorar eficiencia, eliminando la necesidad de polling por ocurrencia de evento.

Desventajas: Posibilidad de desborde. Potencial imprevisión de escalabilidad. Pobre comprensibilidad: Puede ser difícil prever qué pasará en respuesta a una acción. No hay garantía del lado del publisher que el suscriptor responderá al evento. No hay mucho soporte de recuperación en caso de falla parcial.

Tuberías 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.

Ejemplos: UNIX, compiladores, ambiente Khoros de procesamiento de imágenes, procesos de conversión de formatos, XSLT, pipeline de Commerce Server.

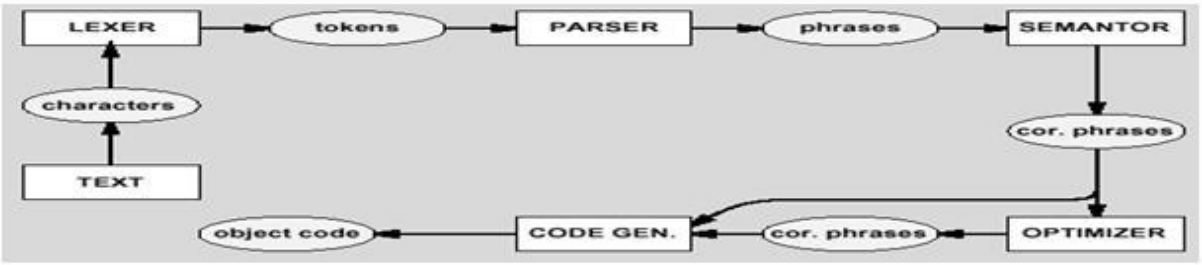

Ilustración 1: Estilo Tuberías y Filtros

Ventajas: Es simple de entender e implementar. Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea. Por ejemplo, transformaciones mediante Mapper de XSLT (BizTalk). Lineal: la conducta del sistema es la suma de la conducta de los sucesivos filtros. Fuerza un procesamiento secuencial. Es fácil de envolver (wrap) en una transacción atómica. Los filtros se pueden empaquetar, y hacer paralelos o distribuidos.

Desventajas: El patrón puede resultar demasiado simplista, especialmente para orquestación de servicios que podrían ramificar la ejecución de la lógica de negocios de formas complicadas. No maneja con demasiada eficiencia construcciones condicionales, bucles y otras lógicas de control de flujo. No es posible que dos o más filtros cooperen. Eventualmente pueden llegar a requerirse buffers de tamaño indefinido, por ejemplo en las tuberías de clasificación de datos. La independencia de los filtros implica que es muy posible la duplicación de funciones de preparación que son efectuadas por otros filtros (p. ej. el control de corrección de un objeto de fecha). Es complicado manejar uniformemente gestión de errores en filtros diferentes.

Modelo-Vista-Controlador

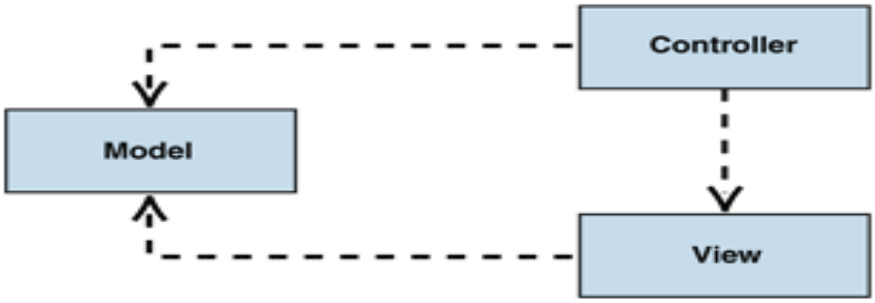

Ilustración 2: Estilo-Modelo-Vista-Controlador

Se emplea para modularizar la interface de usuario, las reglas de negocios y el control de eventos. P. ej. Modelo de programación de Visual Basic, aplicación cliente con interfaz gráfica, etc.

Modelo: Administra el comportamiento y los datos del dominio de aplicación, responde a requerimientos de información sobre su estado (usualmente formulados desde la vista) y responde a instrucciones de cambiar el estado (habitualmente desde el controlador). Mantiene el conocimiento del sistema. No depende de ninguna vista y de ningún controlador.

Vista: Maneja la visualización de la información.

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

Ventajas: Se pueden mostrar distintas variantes de display simultáneamente, porque la vista no depende del modelo (ni el modelo de la vista). La interface tiende a cambiar más rápido que las reglas de negocios. Agregar nuevos tipos de vista no afecta al modelo.

Desventajas: Puede aumentar un poco la complejidad de la solución. Como está guiado por eventos, puede ser algo más difícil de depurar. Si hay demasiados cambios en el modelo, la vista puede caer bajo un diluvio de refrescos, a menos que se prevea programáticamente (por ejemplo, actualizando varias modificaciones en lotes).

Arquitectura de Pizarra

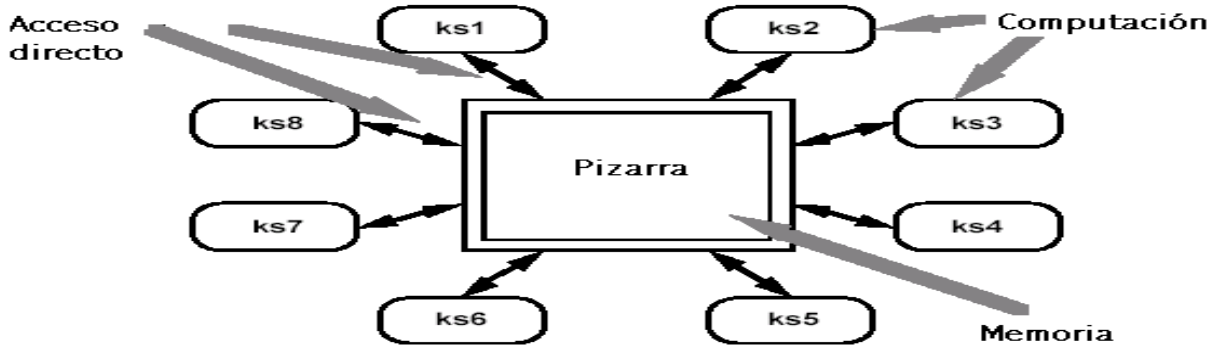

Ilustración 3: Arquitectura de Pizarra

Se utiliza cuando existen problemas no susceptibles de tratarse analíticamente.

Reconocimiento de patrones, aprendizaje de máquina, data mining.

Ejemplos: Procesamiento de señales. Reconocimiento de habla. Redes neuronales. Agentes autónomos (débilmente acoplados).

- Variante repositorio: Base de datos o almacenamiento persistente. Es pasivo. Los clientes acceden a datos compartidos cuando lo requieren.
- Variante pizarra propiamente dicha: Es activo. Pueden enviar notificación a suscriptores cuando la información cambia. El estado es el disparador de la acción a ejecutar.

Ventajas: Posibilita una forma adecuada de manejar grandes volúmenes de datos. Además de que permite lograr una administración centralizada.

Desventajas: Los subsistemas tienen que estar de acuerdo en el modelo de datos del repositorio. La evolución de los datos es difícil y cara. Siendo difícil de distribuir con facilidad.

Máquina Virtual

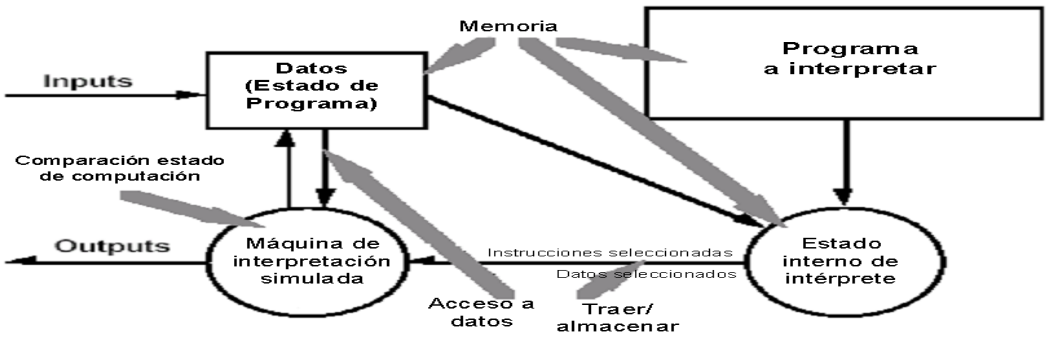

Ilustración 4: Máquina Virtual

Simula funcionalidades no nativas a hardware o software. P. ej. Implementar scripting de alto nivel en aplicación de negocios. Un intérprete posee por lo general cuatro componentes:

Una máquina de interpretación que lleva a cabo la tarea. Una memoria que contiene el pseudo-código a interpretar. Una representación del estado de control de la máquina de interpretación, y una representación del estado actual del programa que se simula.

Ventaja: La principal facilidad que ofrece es la simulación

Desventaja: Su debilidad radica en la especificidad, ya que este estilo es específico para este tipo de simulación.

Arquitectura basada en objetos

Se ha discutido si es un estilo abstracto de arquitectura o más bien una tecnología concreta de implementación. Los componentes son objetos. Los conectores son procesos de invocación de procedimientos y funciones.

Ventajas: Todas las facilidades que brinda la programación orientada a objetos, tales como: encapsulamiento, reutilización, fácil descomposición en una colección de agentes interactivos.

Desventajas: Propagación de cambios (p. ej. nombres, data types de argumentos). Dependencias en cascada. Si se cambia un objeto, se deben modificar todos los que lo invocan. Granularidad muy fina para sistemas grandes. Usualmente, un solo valor de retorno.

Arquitectura en capas

Resulta útil cuando se pueden identificar distintas clases de servicios que pueden ser articulados jerárquicamente. Puede verse como una jerarquía de máquinas virtuales cada vez más poderosas y abstractas. Los componentes de cada capa consisten en conjuntos de procedimientos. Las interacciones entre capas usualmente proceden por invocación de procedimientos. Por definición, los niveles de abajo no pueden usar funcionalidades ofrecidas por los de arriba.

Ventajas: La estructuración en capas posibilita la modularidad. Además de que mantener diferentes tipos de códigos lo más separado posible es una buena práctica en cualquier programación. Facilita el re-uso de código, reduce notablemente la duplicidad de este último y hace que la búsqueda y corrección de problemas sea más fácil en un futuro.

Desventajas: Es difícil razonar a priori sobre la separación en capas requerida. Capas disjuntas requieren drivers de otras capas. Formatos, protocolos y transportes de la comunicación entre capas suelen ser específicos y propietarios. Algunos componentes pueden estar lógicamente vinculados con más de una capa. En ocasiones, hay que pasar por encima de capas intermedias. Otras veces, razones de performance hace que se sitúen funciones en las capas indebidas (lógica de negocios en procedimientos almacenados). La multiplicación de capas genera procesos adicionales de comunicación y coordinación.

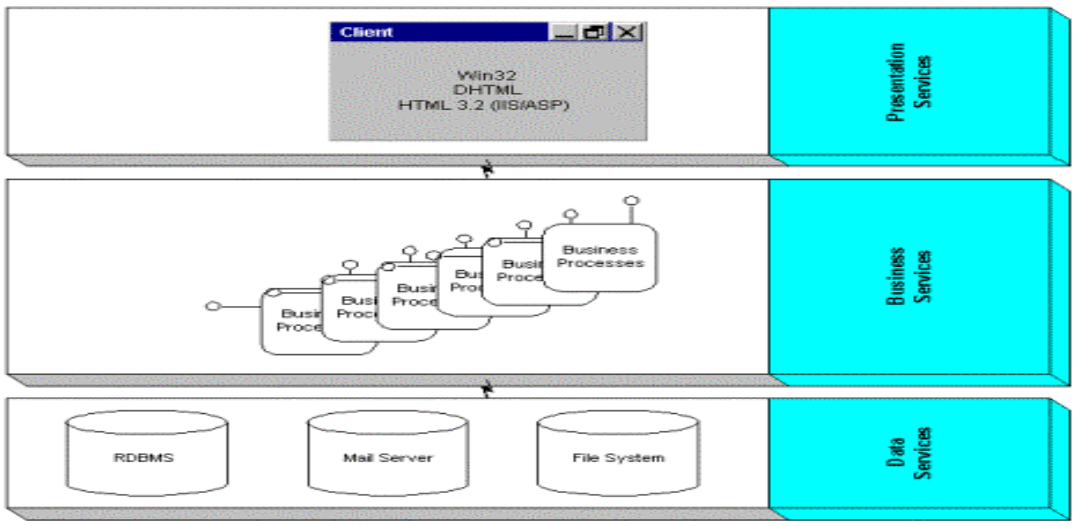

Ilustración 5: Arquitectura en capas

Estilos heterogéneos

Estilos naturalmente heterogéneos (Len Bass)

- ✓ Ocasionalmente heterogéneos: distintas partes son de diferentes estilos.
- ✓ Jerárquicamente heterogéneos: Diferentes partes de un sistema de un estilo corresponden a otro estilo: cliente/servidor
- ✓ Simultáneamente heterogéneos: Diversos estilos pueden describir el mismo sistema (una base de datos multiusuario → repositorio o cliente/servidor).

Estilos compuestos

☻C2 ☻GenVoca ☻REST

Arquitectura C2

Estilo basado en componentes y mensajes para sistemas altamente distribuidos, heterogéneos, con fuerte participación de interface usuario. Las arquitecturas C2 son redes de componentes concurrentes enganchados por conectores.

- No hay conexiones componente a componente
- Todas las comunicaciones por intercambio de mensajes a través de buses

Su objetivo es la reutilización de componentes y conectores. La literatura no señala desventajas. Combina estilo basado en eventos con arquitectura en capas. Cada componente tiene dos puntos de conexión (interfaces), top & bottom. El top (bottom) de un componente sólo se puede ligar al bottom (top) de sólo un bus. No es posible tener ciclos. Un componente no puede recibir un mensaje generado por él mismo.

Estilo Rest

Arquitectura con modelo de datos (recursos, URIs y representaciones XML). Composición de diversos estilos: repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda e interfaz uniforme.

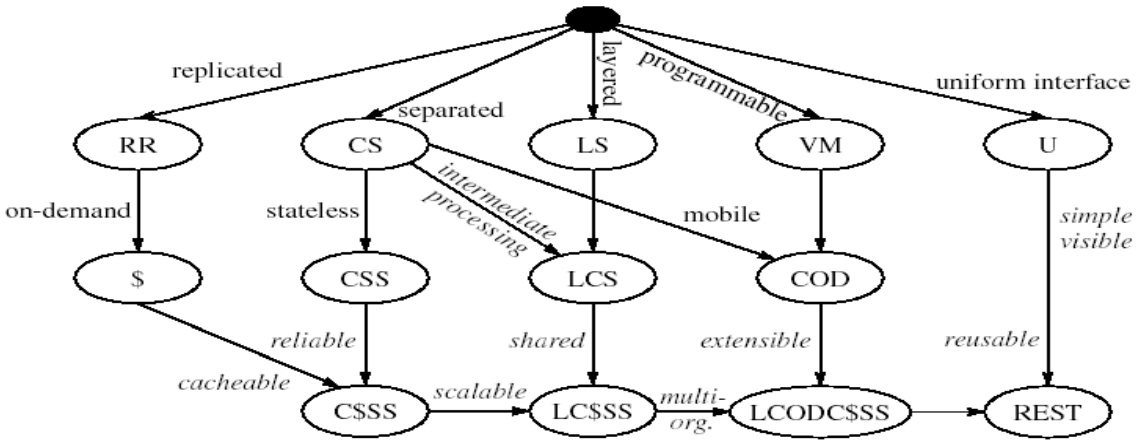

Ilustración 6: Estilo Rest

Una aplicación REST transfiere representaciones entre componentes usando conectores. Componentes: incluyen agentes de usuario (Mozilla, URL) y servidores de origen (Apache, IIS). Los componentes de REST obedecen estas restricciones:

Las interacciones son stateless. Los recursos se identifican mediante URIs**.** No hay tal cosa como servicios ni objetos, sólo recursos. No se permite el paso de cookies y se propone el reemplazo de MIME (Multipurpose Internet Mail Extensions) por su discrepancia arquitectónica con la semántica de HTTP. Hipermedia es la máquina de estado de la aplicación. No hay necesidad de toolkits: sólo URIs y XML.

1.5.2. PATRONES ARQUITECTÓNICOS

“Como un elemento en el mundo, cada patrón es una relación entre cierto contexto, cierto sistema de fuerzas que ocurre repetidas veces en ese contexto y cierta configuración espacial que permite que esas fuerzas se resuelvan. Como un elemento de lenguaje, un patrón es una instrucción que muestra la forma en que esta configuración espacial puede usarse, una y otra vez, para resolver ese sistema de fuerzas, donde quiera que el contexto la torne relevante. El patrón es, en suma, al mismo tiempo una cosa que pasa en el mundo y la regla que nos dice cómo crear esa cosa y cuándo debemos crearla. Es tanto un proceso como una cosa; tanto una descripción de una cosa que está viva como una descripción del proceso que generará esa cosa.” (ALEXANDER C., 1977).

“Un sistema bien estructurado está lleno de patrones” (REYNOSO, 2005)

“Los patrones de arquitectura se pueden ver como la descripción de un problema en particular y recurrente de diseño, que aparece en contextos de diseño arquitectónicos específicos, y representa 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 en que estos colaboran entre sí. “(FRANK BUSCHMANN, 1996)

Entre los Patrones de Arquitectura se encuentran: **layer, microkernel, pipe-filter, broker, model view controller, reflection.**

La tabla representada muestra las categorías y los patrones que estas contienen.

Categoría	**Patrón**
Estructura	Capas
	Conductos y filtros
	Pizarra
Sistemas distribuidos	Intermediario
Sistemas interactivos	Modelo-Vista-Controlador
	Presentación-Abstracción-Control
Sistemas adaptables	Reflejo
	Microkernel

Tabla 1: Patrones de arquitectura. Categoría.

Tuberías y Filtros: Descripción: Cada componente tiene un conjunto de entradas y un conjunto de salidas. Un componente lee entradas y las transforma en salidas. Restricciones**:** Los filtros deben ser independientes. No deben compartir estado con otros filtros. Los filtros realizan la labor independientemente del flujo de entrada. Ventajas: Es fácil de entender e implementar. Es posible implementar procesos complejos con editores gráficos de líneas de tuberías o con comandos de línea.

Capas (Layers): ayuda a estructurar aplicaciones que pueden estar descompuestas en grupos de subtareas en las cuales cada grupo de subtareas está en un nivel particular de abstracción.

Model View Controller (MVC): divide una aplicación interactiva en tres componentes. El modelo contiene los datos y funcionalidades de fondo. Se utiliza cuando es necesario modularizar la interfaz de usuario, las reglas del negocio y el control de eventos.

Broker: Es empleado para estructurar sistemas distribuidos de software con componentes desacoplados que interactúan por invocaciones remotas de servicio. Un componente del agente es responsable de coordinar comunicación, algo semejante como reenviar demandas múltiples, como para excepciones y resultados transmisores.

Microkernel: Aplicado a los sistemas de software que deben poder adaptarse a requisitos de sistema cambiantes. Separa un corazón funcional mínimo de funcionalidad extendida y partes creadas por el usuario. El microkernel también sirve de un conector para enchufar estas extensiones y coordinar su colaboración.

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

En este epígrafe se han descrito los diferentes patrones de arquitectura, algunos de los cuales serán empleados en la propuesta de solución, de acuerdo a las características del sistema y el estilo arquitectónico que se decida emplear.

1.6. Arquitectura de simuladores y métodos de cálculo.

Secuencial-Modular: Los cálculos se realizan unidad por unidad, secuencialmente. Los procesos con reciclos deben ser descompuestos en varias secuencias de cálculo hasta lograr convergencia, usando los balances de masa y energía como criterio para terminar el cálculo. Esta estrategia de cálculo es utilizada en la mayoría de los simuladores de estado estacionario: Aspen, CHemCAD, ProVision, Hysys, Prosim, Winsim.

Orientada a Ecuaciones: En este caso todas las ecuaciones del modelo, algebraicas no lineales y diferenciales, se integran en un único conjunto y se resuelven simultáneamente. Este esquema es más flexible que el Secuencial-Modular, sin embargo requiere más esfuerzo de programación y se consumen más recursos de computación. Las corrientes de reciclo no representan una dificultad en la resolución del modelo. Esta arquitectura está más orientada a la solución de modelos dinámicos. En general los resultados se muestran como tablas o gráficas, donde se muestra el comportamiento de las variables en función del tiempo. Aspen Dynamics y Simulink utilizan esta arquitectura.

Módulos Simultáneos: Esta estrategia de solución combina los Módulos Secuenciales y Solución Orientada a Ecuaciones. Modelos rigurosos de las operaciones unitarias son resueltos secuencialmente, mientras que modelos lineales son resueltos globalmente para interconectar los resultados de cada módulo. Este parece ser el enfoque que a futuro se dará en los simuladores comerciales.

CAPE-OPEN: En cuanto a arquitectura para simuladores se refiere, el estándar CAPE-OPEN es un ejemplo a seguir.

“Constituye una tecnología para la integración. Un estándar de la industria, de libre disposición para las interfaces entre los componentes de software que constituyen herramientas de simulación de procesos de software. Permite el éxito de la colaboración entre los proveedores de software, los usuarios finales y académicos. Una tecnología aplicada en la mayoría de herramientas de simulación de procesos. Es multiplataforma y utiliza importantes frameworks y middlewares como CORBA, COM, J2EE, EJB, etc., orientada a componentes, tiene una concepción de interfaces orientadas a objetos y usa técnicas de reingeniería y encapsulación. Entre los ambientes para la simulación de procesos desarrollados sobre esta plataforma están ProsimPlus de la compañía ProSim SA, simulador estacionario para la producción de oxígeno, entre otros.” (CORRALES VALDÉS, 2007)

Qué permite el estándar CAPE-OPEN: Más barato, mejor y un rápido diseño, funcionamiento y control de procesos.

- ✓ **Plug Plug-and-play:** Capacidad para integrar un componente de la biblioteca de objetos (unidad de operaciones, termo modelos, etc.). Capacidad para integrar sin problemas en el seno de propiedad de componentes en entornos comerciales.
- ✓ **Nicho de software:** Capacidad de conexión de módulos específicos para el nicho de simuladores. Pequeños nichos y los proveedores de software proporcionará componentes CO-compatibles.
- ✓ **Retorno de la Inversión:** Los estudios individuales cuestan menos, debido a las ventajas de la técnica de ser capaz de mezclar y coincidir.

El estudio realizado en este epígrafe ha constatado, que para llevar a cabo la simulación de un proceso determinado, en el subsistema al cual se le quiere desarrollar la arquitectura, es necesario seguir el método modular secuencial, ya que el simulador que lo contiene es de este tipo, y de estado estacionario además. Por lo que la arquitectura propuesta debe estar preparada para soportar la gran cantidad de operaciones que conlleva la simulación y para ello se seguirá el ya mencionado estándar CAPE-OPEN.

1.7. METODOLOGÍAS

Las metodologías son el conjunto de procedimientos, técnicas, herramientas y un soporte documental que ayuda a los desarrolladores a realizar un nuevo software. No existe una metodología de software universal. Las características de cada proyecto exigen que el proceso sea configurable. Una metodología puede seguir uno o varios modelos de ciclo de vida, es decir, indica qué es lo que hay que obtener a lo largo del desarrollo del proyecto pero no cómo hacerlo.

1.7.3. Metodología a emplear

RUP (Proceso Racional Unificado): Es una metodología para el desarrollo de un proyecto de software que define claramente quien, cómo, cuándo y qué debe hacerse en este. Como tres características esenciales está **dirigido por los Casos de Uso**: que orientan el proyecto a la importancia para el usuario y lo que este quiere, está **centrado en la arquitectura**: que relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y es **iterativo e incremental**: divide el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más

depurada. La metodología RUP es más apropiada para proyectos grandes (Aunque también pequeños), dado que requiere un equipo de trabajo capaz de administrar un proceso complejo en varias etapas. En proyectos pequeños, es posible que no se puedan cubrir los costos de dedicación del equipo de profesionales necesarios.

RUP maneja 6 principios claves:

Adaptación del proceso: El proceso deberá adaptarse a las características propias de la organización. El tamaño del mismo, así como las regulaciones que lo condicionen, influirán en su diseño específico. También se deberá tener en cuenta el alcance del proyecto.

Balancear prioridades: Los requerimientos de los diversos inversores pueden ser diferentes, contradictorios o disputarse recursos limitados. Debe encontrarse un balance que satisfaga los deseos de todos.

Colaboración entre equipos: El desarrollo de software no lo hace una única persona sino múltiples equipos. Debe haber una comunicación fluida para coordinar requerimientos, desarrollo, evaluaciones, planes, resultados, etc.

Demostrar valor iterativamente: Los proyectos se entregan, aunque sea de un modo interno, en etapas iteradas. En cada iteración se analiza la opinión de los inversores, la estabilidad y calidad del producto, y se refina la dirección del proyecto así como también los riesgos involucrados.

Elevar el nivel de abstracción: Este principio dominante motiva el uso de conceptos reutilizables tales como patrón del software, lenguajes 4GL o esquemas (frameworks) por nombrar algunos. Éstos se pueden acompañar por las representaciones visuales de la arquitectura, por ejemplo con UML.

Enfocarse en la calidad: El control de calidad no debe realizarse al final de cada iteración, sino en todos los aspectos de la producción.

El ciclo de vida de RUP: Divide el proceso en cuatro fases, dentro de las cuales se realizan varias iteraciones en número variable según el proyecto y en las que se hace un mayor o menor hincapié en los distintas actividades.

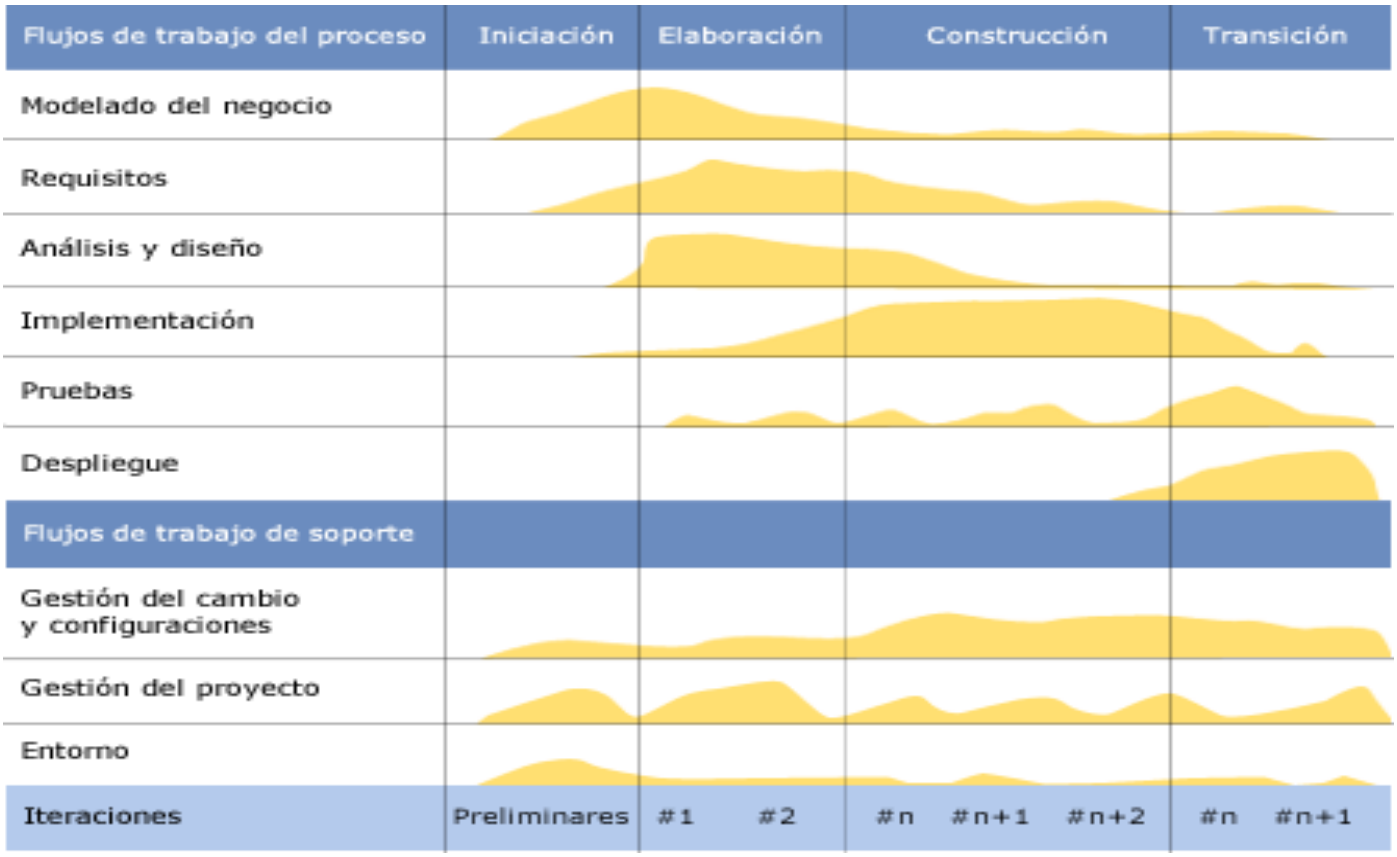

Ilustración 7: Esfuerzo asociado a las disciplinas según las fases de RUP

1.7.3.1. ¿Por qué emplear RUP?

RUP ofrece varias ventajas que hacen que se la metodología a emplear en el desarrollo de la arquitectura para el subsistema de modelos y propiedades, ya que además de ser utilizada en la mayoría de los proyectos en la universidad, ofrece algunas de las mejoras que se enuncian a continuación:

Es un modelo donde puedes tener un control total en la forma que tiene en cuenta todos los detalles que tal vez modelos anteriores olvidaban o que no le daban mucha importancia. Toma lo mejorcito de cada modelo anterior, es a lo que suele llamarse UNIFICADO. Y es que se basa en las mejores prácticas que se han intentado y se han probado en el campo.

Busca detectar defectos en las fases iniciales. Realiza el Análisis y diseño, tan completo como sea posible. Diseño genérico, intenta anticiparse a futuras necesidades. Existe un contrato prefijado con los

clientes. Intenta reducir el número de cambios tanto como sea posible. Es una de las metodologías más utilizadas a nivel mundial. Su aplicación ha arrojado buenos resultados en proyectos nacionales e internacionales. Está diseñada de forma que exista un entendimiento continuo y gradual de todos los implicados en el proyecto. Define roles fundamentales que tienen a su cargo la generación de artefactos y documentación correctos por cada fase y flujo que tributan a un buen producto final. Centra su ciclo de vida en la arquitectura. Otra de las facilidades que permite RUP es precisamente su característica Iterativo e incremental, permitiendo el trabajo en versiones. “Vivimos en un mundo rodeado de versiones, no solo en el entorno de software sino en muchas cosas más como autos o celulares e incluso los artículos electrónicos de nuestra casa, y ¿por qué todo el mundo trabaja con versiones? pues fácil es lo mejor para el usuario y el desarrollador, ya que el usuario con cada versión queda cada vez más satisfecho y el ingeniero nunca o casi nunca va a estar desocupado, sin trabajo.” (ANÓNIMO, 2008)

Después de haber visto y analizado toda la serie de ventajas y características que ofrece RUP, es la metodología seleccionada para el desarrollo de la arquitectura del modelo de subsistemas y propiedades. Y precisamente una de sus ventajas es que está centrado en la arquitectura, aspecto sumamente importante para el desarrollo de cualquier producto de software.

1.8. LENGUAJE UNIFICADO DE MODELADO (UML)

Es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad; está respaldado por el OMG (Object Management Group). UML se utiliza para definir y detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Los objetivos de UML son muchos, pero se pueden sintetizar sus funciones: **Visualizar**: Permite expresar de una forma gráfica un sistema de forma que otro lo puede entender. **Especificar**: Posibilita especificar cuáles son las características de un sistema antes de su construcción. **Construir**: A partir de los modelos especificados se pueden construir los sistemas diseñados.

Documentar: Los propios elementos gráficos sirven como documentación del sistema desarrollado, pudiendo ser útiles para futuras revisiones. Se puede aplicar en el desarrollo de software entregando gran variedad de formas para dar soporte a una metodología de desarrollo (tal como el Proceso Unificado Racional o RUP), pero no especifica en sí mismo qué metodología o proceso usar.

1.8.1. Razones para emplearlo

Como se hizo referencia anteriormente a que la metodología que se quiere emplear es RUP, cabe destacar que aunque UML es bastante independiente del proceso de desarrollo que se siga, los mismos creadores de UML han propuesto su propia metodología de desarrollo, denominada el Proceso Unificado de Desarrollo. RUP utiliza el UML para expresar gráficamente todos los esquemas de un sistema software.

"Los beneficios que se consiguen al utilizar UML son varios, por un lado el uso de lenguajes visuales facilitan su asimilación y entendimiento por parte del equipo de desarrollo; el tiempo invertido en el desarrollo de la arquitectura se minimiza; la detección y resolución de errores se agiliza siempre y cuando se haga uso de herramientas adecuadas de diagnóstico y depuración; y la trazabilidad y documentación del proyecto se realiza de una forma ordenada y guiada por los casos de uso. Pero si hay una ventaja que destaca sobre todas las demás es la notable efectividad y productividad que se consigue en labores de diseño arquitectónico y mantenimiento haciendo uso de UML frente a la realización de las mismas tareas en ausencia de modelos".(LEÓN, 2000)
Otras ventajas:

- Ha sido apoyado por prácticamente todas las empresas importantes de informática.
- Se ha aceptado como un estándar por la OMG.
- Prácticamente todas las herramientas CASE y de desarrollo la han adaptado como lenguaje de modelado.
- Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.
- Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).Cubre las cuestiones relacionadas con el tamaño propio de los sistemas complejos y críticos. Es un lenguaje muy expresivo que cubre todas las vistas necesarias para desarrollar y luego desplegar los sistemas.
- Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.

1.9. HERRAMIENTA CASE A EMPLEAR

La modelación orientada a objetos ha llegado a su punto de esplendor desde la aparición de UML. Como soporte a este lenguaje de modelado algunas herramientas han hecho su aparición en la red de redes. Las compañías que las tienen a su cargo han desarrollado más de una versión que van enfocadas, en algunos casos a mejoras respecto a versiones anteriores, y en otros casos a darles un giro a su utilidad hacia un campo más específico. A continuación se describe la herramienta a emplear para el modelado.

1.9.1. Rational Suite 2003

Racional Suite ofrece un conjunto completo de herramientas integradas que incorporan las mejores prácticas de ingeniería de software y abarcan todo el ciclo de vida de desarrollo de software.

Rational Rose Enterprise Edition es una de las herramientas más técnicas para el desarrollo de software, debido a que se encarga de llevar a cabo tanto la automatización de los sistemas para la posterior generación de código (esto es, realización de los distintos diagramas y generación del código posterior), como para labores de ingeniería inversa (es decir, realización de los diagramas una vez conocido el código). De ahí que pueda facilitar el tener un simple diagrama de clases en frente para la comprensión del sistema en sí. Sin duda, en esto Rational Rose Enterprise Edition es una forma de ayuda para comprensión del sistema y de sus distintos componentes, y lo mejor es que puedes aplicar ingeniería inversa a una multitud de códigos distintos, siempre que obviamente estén orientados a objetos.

“Esta herramienta integra todos los elementos que propone la metodología RUP para cubrir el ciclo de vida de un proyecto y supone la utilización de varios modelos para realizar un diseño del sistema utilizando los recursos gráficos del lenguaje UML. Contiene varias aplicaciones entre las que se encuentran: Requisite Pro, Rational Unified Process, SoDA, Clear Quest, Clear Case, Rose 2003, entre otras, proporcionando esta última que es la seleccionada, valiosos mecanismos para desarrollar un buen sistema informático sobre la base de la documentación que genera sobre todo en relación al análisis y diseño.” (SÁNCHEZ, 2007)

1.10. Conclusiones

En el análisis de la teoría existente se pudo apreciar la gran variedad de simuladores comerciales, algunos de los cuales son poderosas herramientas de cálculo, con inmensos bancos de datos que poseen las propiedades físicas de miles de compuestos y sustancias químicas, selección de modelos termodinámicos, cálculo de equipos, condiciones de operación y demás características que debe soportar la arquitectura propuesta para brindarle versatilidad al simulador en general. Debido a esto el estilo que se quiere desarrollar es una arquitectura en capas teniendo en cuenta el estándar CAPE-OPEN, los métodos secuenciales modulares, ya que el simulador es de este tipo, así como la metodología seleccionada y la tecnología de desarrollo a emplear para dar cumplimiento al objeto de estudio de esta investigación. Además de que siempre es recomendable separar en capas un software, se tuvo en cuenta la gran cantidad de cálculos que implica un simulador por lo que en este caso la propuesta de arquitectura se podría separar en datos, procesamiento y emplear un patrón de diseño, que sirva de fachada para comunicar la lógica de negocio con los demás subsistemas. Teniendo presentes factores como la calidad, robustez, además de hacer todo lo posible por lograr que sea multiplataforma, lo que amplía las posibilidades de uso del subsistema en cualquier simulador de procesos, y más en un mundo altamente tecnológico como el actual, donde cada vez se hace más popular el uso de otros sistemas operativos de la plataforma Linux y donde varios de los simuladores está licenciados bajo la GNU.

CAPÍTULO II: DESCRIPCIÓN Y ANÁLISIS DE LA ARQUITECTURA PROPUESTA

Con este capítulo se pretende realizar la descripción de la arquitectura propuesta para el subsistema de modelos y propiedades del simulador. Con ello se debe tener en cuenta aspectos constatados anteriormente tales como: arquitectura de software, estilos y patrones arquitectónicos.

En el capítulo precedente se pudo apreciar la importancia que tiene el rol del arquitecto dentro del equipo de desarrollo, debido a que es el encargado de establecer la correspondencia entre la funcionalidad que va a tener el sistema y la forma de la aplicación. Su principal función en la fase de elaboración, es definir la línea base de la arquitectura, donde se va a estar trabajando sobre los casos de uso arquitectónicamente significativos, por lo que en la medida en que avance el proyecto de software, la arquitectura debe estar preparada para que incluya los demás casos de uso del sistema.

Para definir la línea base de una arquitectura que capture toda la estructura de subsistemas que se necesitan para una aplicación se debe:

- **Definir la infraestructura con la que se cuenta**: Computadoras, dispositivos, sistemas operativos, servidores, redes de comunicación. Identificar las propiedades del sistema globalmente, a través del tipo de aplicación que se está construyendo, si es distribuida, centralizada, interactiva, etc.
- **Seleccionar los patrones arquitectónicos**: En dependencia del tipo de aplicación y la infraestructura con la que se cuenta. Combinar luego los patrones.
- **Identificar los subsistemas funcionales**: Con los métodos de análisis y diseño y luego llenar la distribución que se escogió desarrollar, para más tarde integrar los subsistemas funcionales con la infraestructura del sistema.

La descripción de la arquitectura se representa mediante una serie de vistas arquitectónicas diferentes para representar distintos aspectos del sistema. Como se había planteado previamente que la metodología a emplear sería RUP, el mismo define dicha representación a través de cuatro 4+1 vistas, o sea que cuatro de ellas están regidas por una rectora: la Vista de Casos de Uso. Siendo las restantes, la Vista Lógica, Vista de Procesos, Vista de Despliegue y la Vista de Implementación.

Esencialmente, las vistas de la arquitectura se pueden ver como abstracciones o simplificaciones de los modelos construidos en las que se hacen más visibles las características importantes, y se dejan de lado los detalles. La arquitectura es un método relevante para aumentar la calidad de cualquier compilación de modelo durante el desarrollo del sistema. Las vistas se muestran en el Documento de Arquitectura de Software. Se puede añadir otras, como una vista de seguridad, para representar otros aspectos específicos de la arquitectura de software.

2.1. REQUERIMIENTOS

En la actualidad el desarrollo de software muchas veces no satisface las expectativas de los clientes, y de ahí que comience el rechazo a varios productos. Al no realizarse un buen entendimiento con el usuario sobre la capacidad o condición que su sistema debe cumplir, no se llevará a cabo una buena gestión de requisitos, lo cual conlleva al fracaso, ya que al final no se obtiene el producto esperado. Es por ello que a la hora de desarrollar una buena arquitectura no se debe dejar a un lado los requerimientos**.**

“La arquitectura es uno de los medios para que un proyecto cumpla los requisitos. Por supuesto, hay numerosos aspectos de desarrollo de software que contribuyen a que los requisitos se cumplan. La arquitectura ofrece la estructura para el desarrollo, mejorar el control, por lo que el proyecto puede ser entregado con una mayor seguridad.”(JUAREZ, 2008)

Los casos de uso son empleados para expresar las funcionalidades presentes en nuestro sistema. De ahí que a la hora de desarrollar la línea base de la arquitectura se tengan en cuenta los guiones de uso más significativos, los cuales serán descritos en este espacio del capítulo, junto a los requisitos funcionales y no funcionales que deben estar presentes para desarrollar una arquitectura que se adecue al subsistema de modelos matemáticos y propiedades para un simulador de la industria química.

2.1.1. Requisitos Funcionales

R.1. Gestionar componente químico de la BD de componentes.

R.1.1. Insertar nuevo componente químico.

R.1.1.1. Insertar componente químico puro.

R.1.1.2. Insertar componente químico compuesto.

R.1.2. Gestionar propiedades de un componente personalizado.

R.1.2.1. Modificar propiedades de un componente personalizado.

R.1.2.2. Estimar propiedades de componente químicos compuesto.

R.1.3. Realizar búsqueda de componente.

R.1.4. Clonar componente.

R.1.5. Eliminar componente personalizado.

R.2. Gestionar biblioteca de componentes

R.2.1. Listar bibliotecas de componentes

R.2.2. Crear biblioteca de componentes personalizada

R.2.3. Gestionar componentes de una biblioteca personalizada

R.2.3.1. Adicionar componente a una biblioteca.

R.2.3.2. Eliminar componente de una biblioteca.

R.2.3.3. Listar componentes de una biblioteca.

R.2.4. Eliminar biblioteca de componentes personalizada.

R.3. Gestionar proyecto de simulación.

R.3.1. Crear nuevo proyecto.

R.3.2. Salvar fichero de proyecto.

R.3.3. Abrir fichero de proyecto.

R.4. Simular proceso químico.

R.4.1. Seleccionar componentes químicos presentes en el proceso.

R.4.2. Gestionar paquetes de propiedades.

R.4.2.1. Listar paquetes de propiedades.

R.4.2.2. Asignar paquetes de propiedades al proceso.

R.4.3. Gestionar Unidad Operación.

R.4.3.1. Listar unidades de operación.

R.4.3.2. Adicionar unidad de operación o modulo.

R.4.3.3. Modificar propiedades específicas de la unidad.

R.4.3.4. Eliminar unidad de operación.

R.4.3.5. Obtener estado de una unidad de operación.

R.4.4. Gestionar corriente.

R.4.4.1. Listar corrientes.

R.4.4.2. Crear corriente.

R.4.4.3. Gestionar corriente de entrada o información.

R.4.4.3.1. Modificar composición de la corriente.

R.4.4.3.2. Modificar propiedades de la corriente.

R.4.4.4. Obtener propiedades de la corriente.

R.4.4.5. Conectar corriente a unidad de operación.

R.4.4.6. Eliminar corriente.

R.4.4.7. Calcular propiedades.

R.5. Gestionar resultados de una simulación.

R.5.1. Guardar resultados de la simulación.

R.5.2. Obtener resultados de la simulación de un proceso.

R.5.3. Listar resultados.

R.5.4. Cargar resultados.

R.5.5. Eliminar resultados.

R.6. Manipular unidades de medida.

2.1.2. Requisitos No Funcionales

Interfaces Software: El subsistema debe brindar interfaces para los subsistemas de Interfaz Grafica y Análisis de los Resultados.

Usabilidad: La aplicación podrá ser usada por cualquier personal capacitado, en especial por ingenieros químicos. Contendrá un manual de usuario con las descripciones básicas sobre cómo trabajar con la aplicación incluyendo las especificaciones o características de los procesos. No tiene restricciones de usuarios.

Rendimiento: Las operaciones unitarias se efectuaran inmediatamente se disponga de los datos necesarios para realizar los cálculos.

Seguridad: Confiabilidad: No existen restricciones de acceso a la información manejada. Integridad: Solo se puede modificar o eliminar aquella información que haya sido introducida por el usuario.

Disponibilidad: El subsistema brinda información de los procesos, componentes, propiedades, flujos y corrientes cada vez que le sean solicitadas.

Soporte: Se le debe dar un mantenimiento periódico al subsistema y a la Base de Datos de componentes.

Software: Para la instalación se debe disponer de un sistema operativo superior o igual a Windows XP. El sistema debe ser una aplicación de escritorio.

Hardware: Para instalar la aplicación se necesitan 1 GB de disco duro disponibles, máquinas con 256 MB de RAM y Procesador Pentium 3 o superior.

Restricciones en el Diseño y la Implementación: Se usará como lenguaje de programación C#. Se utiliza UML como lenguaje de modelado. Permitir integrarse con otros formatos de aplicaciones conocidas.

Estándares Aplicables: Integrado al estándar CAPE OPEN.

2.2. ESTUCTURACIÓN A TRAVÉS DE ESTILOS

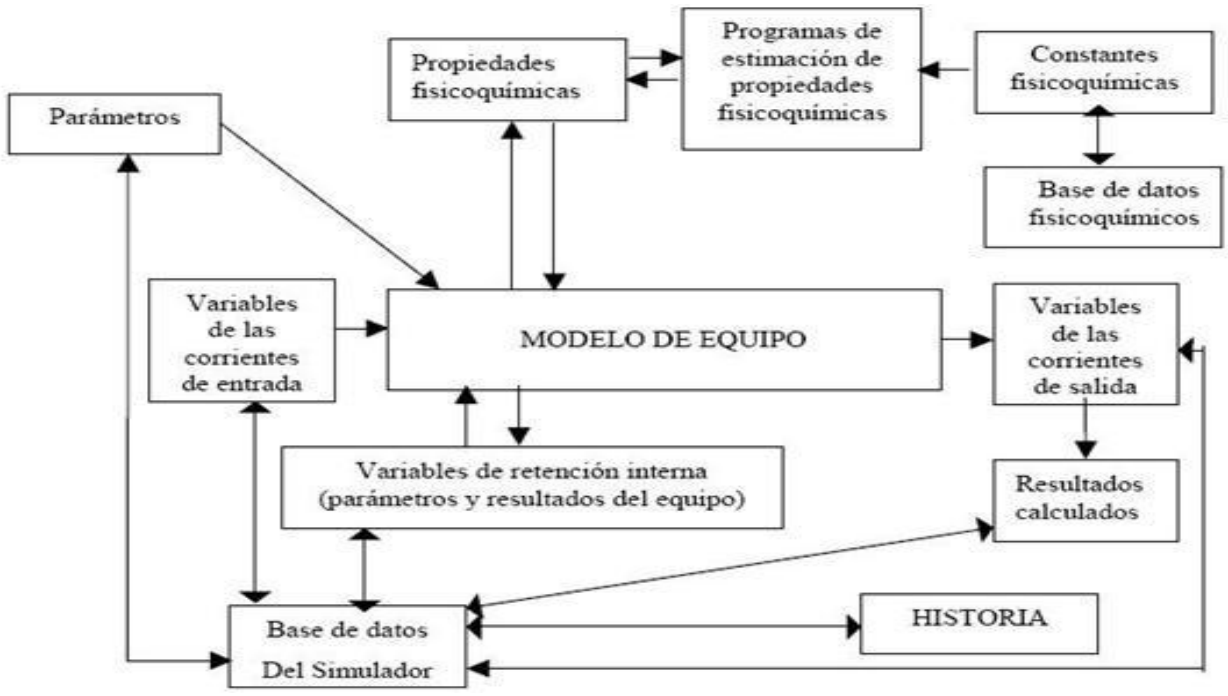

Ilustración 8: Arquitectura típica de un simulador modular secuencial

Para realizar la arquitectura del subsistema de modelos y propiedades es necesario que se posibilite la simulación concurrente de procesos industriales. Además de tener presentes aspectos tales como el esquema de funcionamiento de un módulo, o sea considerar los datos de las corrientes de entrada y salida, la composición de las mismas, etc. También la interrelación entre el módulo y la base de datos de componentes, en cuanto al intercambio de parámetros como las constantes físico-químicas. Así mismo se deben analizar los niveles de cálculo, donde el más potente algoritmo de cálculo se torna ineficiente si las propiedades físico-químicas no son conocidas o incurren en errores importantes. Debido a esto la gestión del paquete de propiedades del proceso y el cálculo de propiedades son aspectos esenciales para la arquitectura del subsistema.

La propuesta que se pretende aplicar se centrará en lo que corresponde a los modelos matemáticos y las propiedades físico-químicas, y es por ello que está basada en el estilo Capas (Layers), debido a que el simulador a tratar es del tipo modular secuencial en estado estacionario, lo que facilita que los cambios en un módulo no afecten la arquitectura del sistema.

En el subsistema tratado se pueden identificar secciones tales como la lógica central del simulador y la estimación de propiedades fisicoquímicas, las cuales pudieran agruparse en un nivel de aplicación o lógica general de administración, puesto que los módulos representativos de los distintos equipos tienen una fisicoquímica asociada para reproducir la operación real de la planta a través de una serie de transformaciones, que implican modelos matemáticos, algoritmos de cálculo y programas de estimaciones fisicoquímicas, para obtener los resultados calculados. El otro nivel estaría dado por modelos matemáticos y una biblioteca de componentes, así como las constantes fisicoquímicas de estos últimos, por lo que sería un nivel para la manipulación de datos. Además de una presentación que contaría con las interfaces necesarias, para interactuar con el subsistema de interfaz gráfica y el de análisis de resultados, que constituyen los actores del subsistema de modelos matemáticos y propiedades. Aunque debe destacarse que el sistema al que se le quiere aplicar el estilo arquitectónico propuesto forma parte de una aplicación de escritorio, donde todos sus componentes correrán sobre la misma computadora, por lo tanto, lo correcto arquitectónicamente sería hablar de un solo nivel pero de tres capas.

Capa Presentación: En este caso no sería la típica capa de presentación ya que el subsistema de modelos y propiedades, comprende toda la lógica de negocio y persistencia de datos del simulador. Donde los usuarios o más bien los actores del mismo, estarían dados por los subsistemas de interfaz gráfica y de análisis de resultados. De ahí que lo que se pretende con esta capa es brindar como se mencionaba anteriormente, una interfaz para cada uno de los subsistemas que interactúan con el de modelos matemáticos y propiedades, agrupadas en un paquete Intercambio.

Capa Negocio: "Está conformada por los subsistemas, los cuales se ajustan a los requisitos y casos de uso arquitectónicamente significativos. Desde el punto de vista de diseño, esta capa es contenedora de las clases entidades y controladoras. Únicamente se comunica con la capa de Acceso a Datos." (FERNÁNDEZ, 2008)

Esta capa sería la encargada de la lógica general del simulador, o sea administrar los distintos procesos que deben ejecutarse para lograr la simulación de un proceso dado. Es decir que deberá procesar el diagrama de flujos, decidir si puede resolverse en una secuencia lineal o si existen reciclos, seleccionar sobre cuáles variables deberá iterarse, determinar en función de las corrientes de corte el orden en el cual serán resueltos los equipos, etc. O sea que, "…deberá manipular un banco de algoritmos y programas de estimación que permitan, dado el Diagrama de Flujo de Información (DFI) de la planta, realizar el rasgado, particionado y ordenamiento o secuencia de resolución." (SCENNA, 1999)

Capa de Persistencia de Datos: En esta sección se encuentran almacenados los datos de las corrientes de salidas y los resultados de equipos, así como las constantes fisicoquímicas de los componentes necesarios para la estimación de propiedades, todo esto para la resolución de un determinado equipo, empleándose un esquema de persistencia para archivos planos. Lo que se quiere lograr en esta capa es definir un conjunto reutilizable de clases que prestan servicios a los objetos persistentes. Posibilitando el almacenamiento y recuperación de dichos objetos. En este caso particular la persistencia se basa en la manipulación de ficheros utilizando el esquema para archivos planos, si en un futuro se quiere usar una base de datos relacional solo hay que agregar un intermediario de esquema relacional y se resolvería el problema, sin necesidad de que los demás componentes se vean afectados, haciendo el diseño más extensible.

2.3. PATRONES A EMPLEAR

2.3.1. PATRONES ARQUITECTÓNICOS

Patrón de Capas (**layer**): Descompone la aplicación en un conjunto de capas independientes y ordenadas jerárquicamente, cada nivel o capa usa los servicios de la capa inmediatamente inferior y ofrece servicios a la capa inmediatamente superior.

"El uso de este patrón brinda la posibilidad de reutilizar un mismo nivel en varias aplicaciones, permite la estandarización, el cambio de nivel no afecta el resto, ahora si no podemos desviarnos de algunos puntos interesantes a la hora de diseñar utilizando este patrón, y estos son: un número excesivo de niveles puede ser ineficiente y un nivel muy bajo hace que nuestras aplicaciones se vuelvan complejos e ineficientes.

Si se hace un mal diseño, nos encontramos frente a la posibilidad de que cambios de funcionalidad se transmitan de un nivel a otro." (FUENTES SUÁREZ, 2007)

Modelo de Dominio (Domain Model)

Problema: En el caso del subsistema de modelos y propiedades del simulador que se quiere desarrollar, la lógica del dominio del sistema es compleja lo que hace difícil la representación de los conceptos y reglas del negocio, implicando una complicada serie de cálculos.

Solución: Permite expresar en objetos todos los componentes y abstracciones, dando todos los beneficios de la programación Orientada a Objetos.

Tubos y Filtros (Piper and Filters)

El patrón de arquitectura de tubos y filtros provee una estructura para procesar flujos de datos. Cada paso de procesamiento se encapsula en un filtro. Los datos se pasan usando los tubos entre filtros adyacentes. Recombinado los filtros se pueden construir distintas familias de sistemas relacionados.

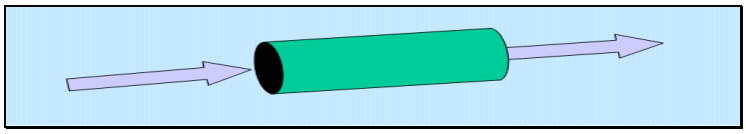

Ilustración 9: Patrón tuberías y filtros

Se quiere emplear este patrón ya que permite que el proceso se divida en varios pasos secuenciales de procesamiento, donde cada filtro consume y procesa sus datos en forma incremental. En el caso del sistema, los módulos o unidades de operación unitarias, constituyen los filtros, mientras que las tuberías están dadas por el mecanismo de comunicación entre los paquetes donde se encuentran los módulos antes mencionados. Además de que la estructura propia del simulador al emplear modelos matemáticos del tipo modular secuencial, es ideal para el encapsulamiento en filtros de las funcionalidades de las unidades de operación.

2.3.2. PATRONES DE DISEÑO

Patrones GoF

Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifican y describen formas de solucionar problemas que ocurren de forma frecuente en el desarrollo. Es una experiencia real, probada, que funciona y nos ayudan a no cometer los mismos errores. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). Se utilizan en situaciones frecuentes**,** ya que se basan en la experiencia acumulada al resolver problemas reiterativos. Ayudan a construir software basado en la reutilización (a construir clases reutilizables). Los propios patrones se reutilizan cada vez que se vuelven a aplicar.

Patrón fachada o facade: Es empleado en la solución para comunicar la lógica de la aplicación con los subsistemas de interfaz gráfica y análisis de resultados a través de fachadas contenidas en el paquete Intercambio. Solo internamente dichas fachadas interactúan con las clases concretas.

Tiene como propósito proporcionar una única interfaz a un conjunto de interfaces de un subsistema. Definiendo una interfaz de más alto nivel que facilita el uso de un subsistema. En este caso que la arquitectura consta de niveles: una fachada define el punto de entrada para cada nivel-subsistema.

Una fachada ofrece los siguientes beneficios:

1) Facilita a los clientes el uso de un subsistema, al ocultar sus componentes.

2) Proporciona un acoplamiento débil entre un subsistema y los clientes: cambios en los componentes no afectan a los clientes.

3) No se impide a los clientes el uso de las clases del subsistema si lo necesitan.

En este caso dentro del simulador que se viene desarrollando el cliente para el subsistema de modelos matemáticos y propiedades serían: el subsistema de interfaz gráfica y el de análisis de resultados, que invocan determinadas funciones a través de estas fachadas, permitiendo así más flexibilidad en el desarrollo de estos sistemas.

Patrón Singleton: Su propósito es asegurar que una clase tiene una única instancia y garantizar un punto de acceso global. Una situación semejante se presenta con los paquetes de propiedades para la simulación, debido a que un mismo paquete es empleado en varios procesos. De ahí que se le aplicará este patrón a dicho elemento.

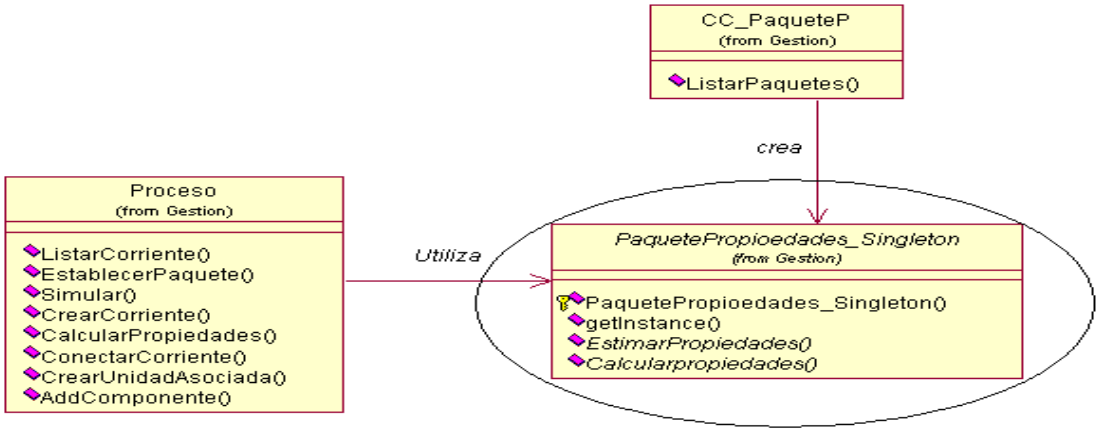

Ilustración 10: Aplicación del Patrón Singleton

Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. GRASP es un acrónimo que significa General Responsibility Assignment Software Patterns (patrones generales de software para asignar responsabilidades). Es por ello que fueron empleados en el subsistema de modelos y propiedades con el fin de diseñar eficazmente el software orientado a objetos. Por ejemplo la creación de objetos es una de las actividades más frecuentes en un sistema orientado a objetos. En consecuencia, conviene contar con un principio general

para asignar las responsabilidades concernientes a ella. El diseño, bien asignado, puede soportar un bajo acoplamiento, una mayor claridad, el encapsulamiento y la reutilizabilidad, aplicando a algunas clases de la arquitectura el patrón creador. Además de que los patrones bajo acoplamiento y alta cohesión, constituyen principios que debemos recordar durante las decisiones de diseño. Con el primero las clases no se afectan por cambios de otros componentes. Son fáciles de entender por separado y fáciles de reutilizar.

Mientras que una alta cohesión caracteriza a las clases con responsabilidades estrechamente relacionadas, que no realicen un trabajo enorme. En la perspectiva del diseño orientado a objetos, la cohesión (o, más exactamente, la cohesión funcional) es una medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. Con este patrón se gana en claridad y facilidad para entender el diseño. Se simplifican el mantenimiento y las mejoras en funcionalidad. A menudo se genera un bajo acoplamiento y finalmente puede decirse que la ventaja de una gran funcionalidad es que soporta una mayor capacidad de reutilización, porque una clase muy cohesiva puede destinarse a un propósito muy específico.

Otros Patrones empleados en la solución propuesta

Intermediario (Broker): Propone construir una clase que se encargue de materializar, desmaterializar y guardar un objeto en otro objeto caché (Clase intermediaria). Donde Cada objeto persistente puede tener su propia clase intermediaria y que los mecanismos de almacenamiento pueden contar con varias clases de Intermediarios.

A continuación se propone un ejemplo de su uso en el paquete de Persistencia, donde se tendrá un intermediario para cada objeto del proceso de simulación de acuerdo a la necesidad de su empleo, dígase con esto uno para los componentes y otro para las propiedades de los mismos, así como el de los resultados obtenidos. Todos ellos heredan del intermediario de la aplicación para la manipulación de archivos planos y este a su vez del principal, o sea, Intermediario de Esquema de Persistencia, el cual es el encargado de guardar los objetos en memoria.

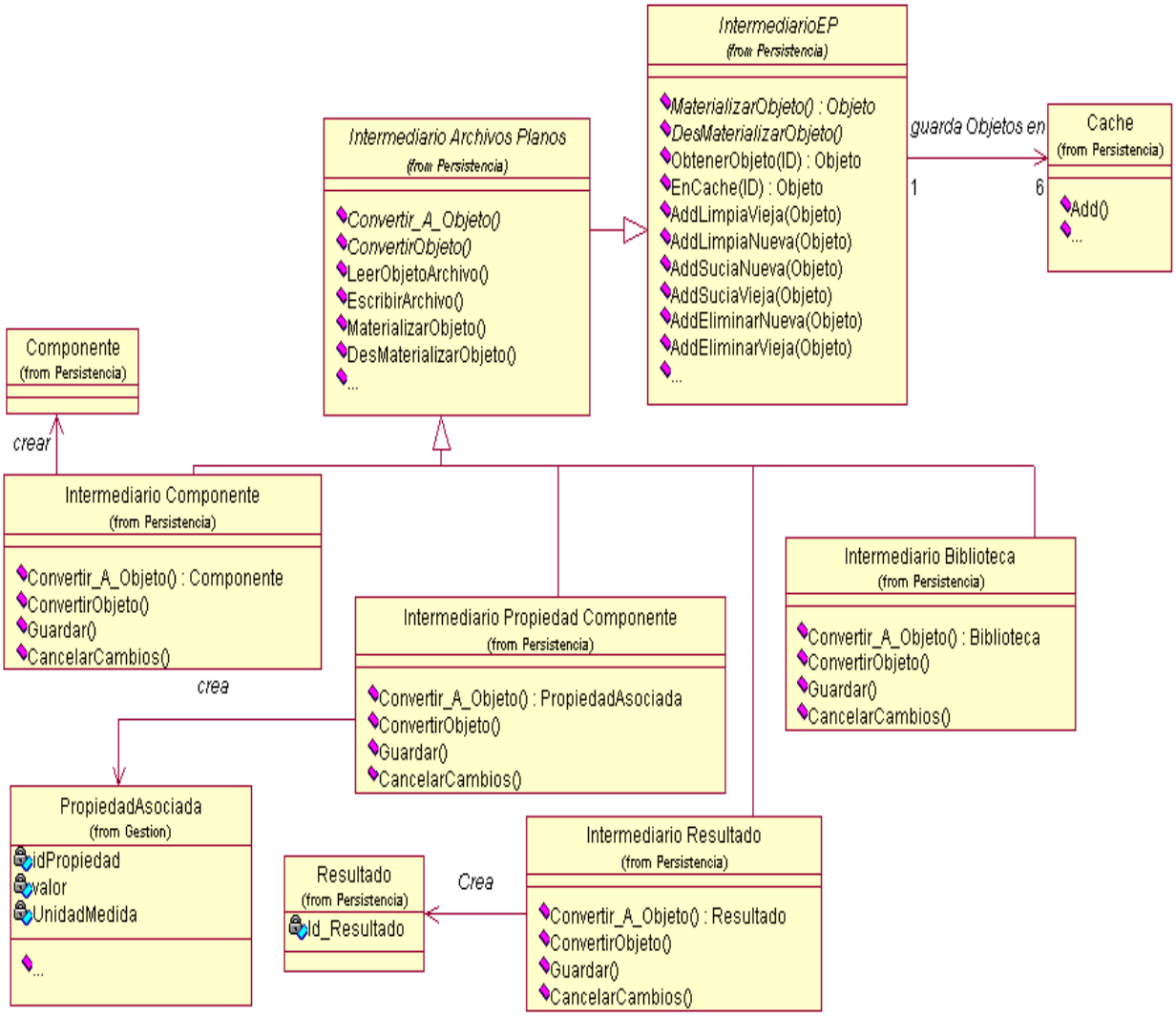

Ilustración 11: Ejemplo del patrón Intermediario (Broker)

Patrón Método de Plantilla: Es el patrón utilizado para el diseño de clases intermediarias. Se define una clase plantilla, la cual puede ser utilizada para definir el esqueleto de un algoritmo, presenta partes variables que se pueden modificar al heredarse a una subclase y otras invariables, que no pueden ser modificadas. Estos métodos pueden o no estar en una subclase y por general llama a otros métodos. Es decir, un método de una subclase será ejecutado sólo si este es llamado desde la clase en la cual fue definido. Por ejemplo, cada una de estas clases heredan el método CrearIntermediario (), para construir su Intermediario correspondiente, ya sea el de componente, el de biblioteca o el del proceso en sí.

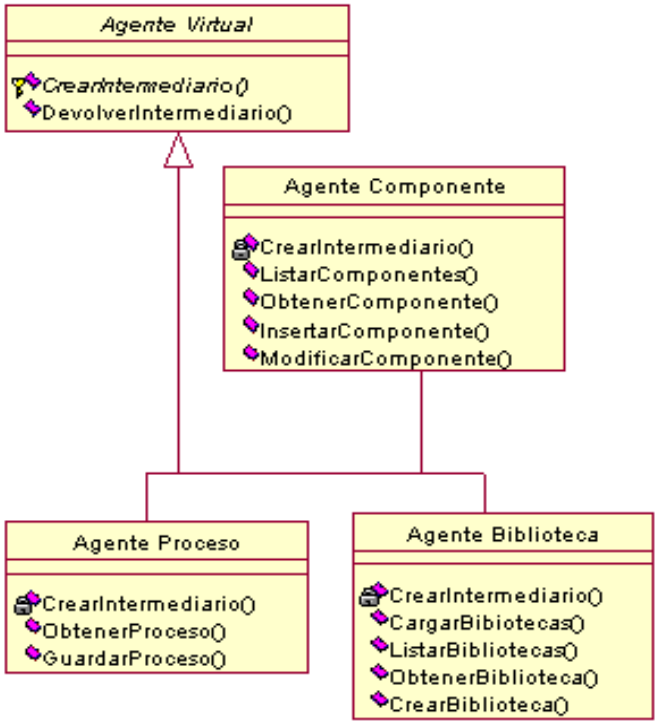

Ilustración 12: Patrón Método de Plantilla

Patrón Administración de Caché: Propone asignar a los intermediarios la responsabilidad de administrar el caché. Si se tiene un intermediario diferente para cada tipo de objeto persistente, cada uno de ellos deberá tener su propia caché. Al materializar un objeto este se deja en caché con su identificador como clave. El intermediario primero buscará en este antes de materializar un objeto. Otra forma de conservar los objetos es en varias cachés, según el estado que presenten dentro del contexto de la transacción actual.

En la página siguiente se muestra un ejemplo de su empleo en el paquete de Persistencia.

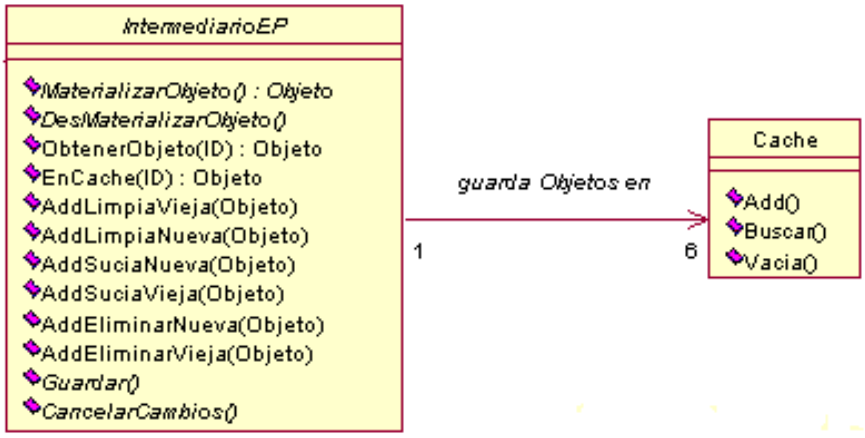

Ilustración 13: Patrón Administración de Caché

2.4. SOPORTE DE DESARROLLO

2.4.1. La plataforma .NET de Microsoft

Está diseñada para que se puedan desarrollar componentes de software utilizando casi cualquier lenguaje de programación, de forma que lo que escribamos en un lenguaje pueda utilizarse desde cualquier otro de la manera más transparente posible (utilizando servicios web como middleware). Esto es, en vez de estar limitados a un único lenguaje de programación, permitimos cualquier lenguaje de programación, siempre y cuando se adhiera a unas normas comunes establecidas para la plataforma .NET en su conjunto. De hecho, existen compiladores de múltiples lenguajes para la plataforma .NET: Visual Basic .NET, C#, Managed C++, Oberon, Component Pascal, Eiffel, Smalltalk, Cobol, Fortran, Scheme, Mercury, Mondrian/Haskell, Perl, Python, SML.NET.

“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. Contiene el CLR (Common Language Runtime), el .NET Framework *Clases* y características avanzadas como ADO.NET (para acceso a bases de datos), *ASP.NET* (para generar páginas activas) y Win Forms (para construir aplicaciones Windows). Adicionalmente puede emplearse Visual Studio.NET que permite hacer todo la anterior desde una interfaz visual basada en ventanas.”(BON, 2007)

2.4.2. Microsoft Visual Studio

Como herramienta de desarrollo se escogió el Visual Studio. Net 2005, y como lenguaje de programación el C#. Por lo que como primer requisito se debe tener instalado el IDE antes mencionado.

Es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Por qué emplearlo: Además de ser la herramienta empleada en el polo de Simulación, Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones web, así como servicios web en cualquier entorno que soporte la plataforma .NET (a partir de la versión net 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas web y dispositivos móviles.

Visual Studio .NET proporciona diversas plantillas de proyecto que pueden utilizarse para iniciar el desarrollo de aplicaciones distribuidas sin tener que empezar de cero. Las plantillas de empresa definen la estructura inicial de una aplicación distribuida, y proporcionan una guía de arquitectura y tecnología para el diseño de la aplicación. Aparte de las ya predefinidas, se pueden crear otras personalizadas que los programadores tienen la posibilidad de utilizar en un entorno de equipo.

“El entorno de Visual Studio 2005 es completamente configurable, pudiéndose adaptar a las necesidades y gustos de todo el mundo. Viene equipado con un avanzado sistema de macros que nos permiten personalizar y automatizar las tareas que realicemos de forma repetitiva. El IDE para macros nos permite desarrollar y depurar macros con las que manipular de forma automática casi cualquier elemento de Visual Studio .NET (proyectos, editor de código, ventanas de herramientas, compilación y despliegue de nuestras aplicaciones)”.(BON, 2007)

Además de que con el lenguaje escogido, se puede implementar la aplicación en Mono, la cual es una excelente plataforma para desarrollar aplicaciones en Linux y al ser multilenguaje, soporta el C#, por lo que esto ayudaría a una futura migración de Windows a Linux, gracias a que Mono contiene APIs de compatibilidad para .NET y permite emplear la experiencia, conocimiento y código de esta plataforma.

2.5. VISTAS ARQUITECTÓNICAS

2.5.1. VISTA DE CASOS DE USO:

Se utiliza en la disciplina Requisitos para proporcionar una base para planificar el contenido técnico de las iteraciones. Representa un subconjunto del artefacto Modelo de Casos de Uso y lista los casos de usos o escenarios más significativos, con las funcionalidades centrales del sistema. Si este último se hace extenso entonces debería organizarse en paquetes, lo cual facilitaría la comprensión de dicha vista. En la misma se representa el diagrama de casos de uso con una breve descripción adjunta.

Visto los requerimientos, los patrones a emplear en la solución y la herramienta de desarrollo para el subsistema de modelos matemáticos y propiedades, es hora de adentrarse en los casos de uso significativos para la arquitectura y el porqué de su clasificación.

- ***Caso de uso Simular:*** el objetivo fundamental de un simulador es el cálculo de las propiedades físicas y químicas de los equipos y de las corrientes que fluyen entre ellos. Por lo que es vital simular el proceso, para comprobar la arquitectura.

- ***Caso de Uso Gestionar Unidad Operación:*** Permite gestionar todo lo relacionado con las unidades de operación unitarias o módulos, listar unidades de operación, adicionar y eliminar unidades del proceso de simulación, modificar propiedades de una unidad y obtener estado de la unidad.

- ***Caso de Uso Gestionar Corriente:*** Permite gestionar las corrientes de los procesos de simulación, y realizar sobre ellas acciones vitales, como crear, listar, eliminar y obtener propiedades de una corrientes, además de conectarlas a unidades de operación.

- ***Caso de Uso Gestionar Corriente de Entrada***: Este caso de uso hereda todas las funcionalidades del caso de uso padre Gestionar Corriente, además de permitir modificar la composición de la corriente y las propiedades de la misma.

- ***Caso de Uso Calcular Propiedades*:** El comportamiento de este caso de uso se inserta explícitamente dentro del caso de uso base Gestionar Corriente Entrada, ya que una vez entrados los parámetros de composición de la misma, se calculan automáticamente sus propiedades. Este

comportamiento también se incluye dentro del caso de uso simular.

- ***Caso de uso Gestionar Paquete de Propiedades:*** Permite listar los paquetes de propiedades o asignarle un nuevo paquete de propiedades al proceso de simulación en cuestión.

- ***Caso de Uso Gestionar Componente Proceso*****:** Siempre que se quiere formar una determinada corriente para realizar los cálculos de las operaciones unitarias es necesario y obligatorio seleccionar los componentes que conforman la misma, así como realizar otras operaciones sobre los mismos como adicionar un nuevo componente o eliminar uno anteriormente seleccionado, de ahí el por qué de su clasificación.

- ***Caso de Uso Buscar Componente:*** Fue incluido como un caso de uso arquitectónicamente significativo, ya que a la hora de iniciar todo el proceso a simular siempre es necesario realizar la búsqueda de un componente determinado, debido a esto su comportamiento se incluye específicamente dentro del caso de uso gestionar componentes presentes en el proceso, pues al realizar cualquier operación sobre las sustancias, es necesario realizar una búsqueda previa

- ***Caso de Uso Guardar Resultados*****:** Permite salvar toda la información de los cálculos y las estimaciones realizadas sobre uno o varios módulos, que más tarde podría ser utilizada en otras simulaciones a través del subsistema de interfaz gráfica o para el análisis de resultados por parte del subsistema del mismo nombre. No se incluye en Gestionar Resultados ya que el actor que inicia en este caso la acción, sería el subsistema de Interfaz Gráfica, al invocar a través de la fachada la operación de guardar resultados.

- ***Caso de Uso Gestionar Resultados*****:** Su clasificación viene dada por la importancia que tiene la obtención y análisis de resultados para el subsistema del mismo nombre, el cual después de llevada a cabo la simulación invoca diferentes operaciones contenidas en la fachada para este subsistema. Este caso de uso permite obtener, listar, cargar y eliminar los resultados de la simulación llevada a cabo durante un proceso determinado.

Representación de los casos de usos arquitectónicamente significativos.

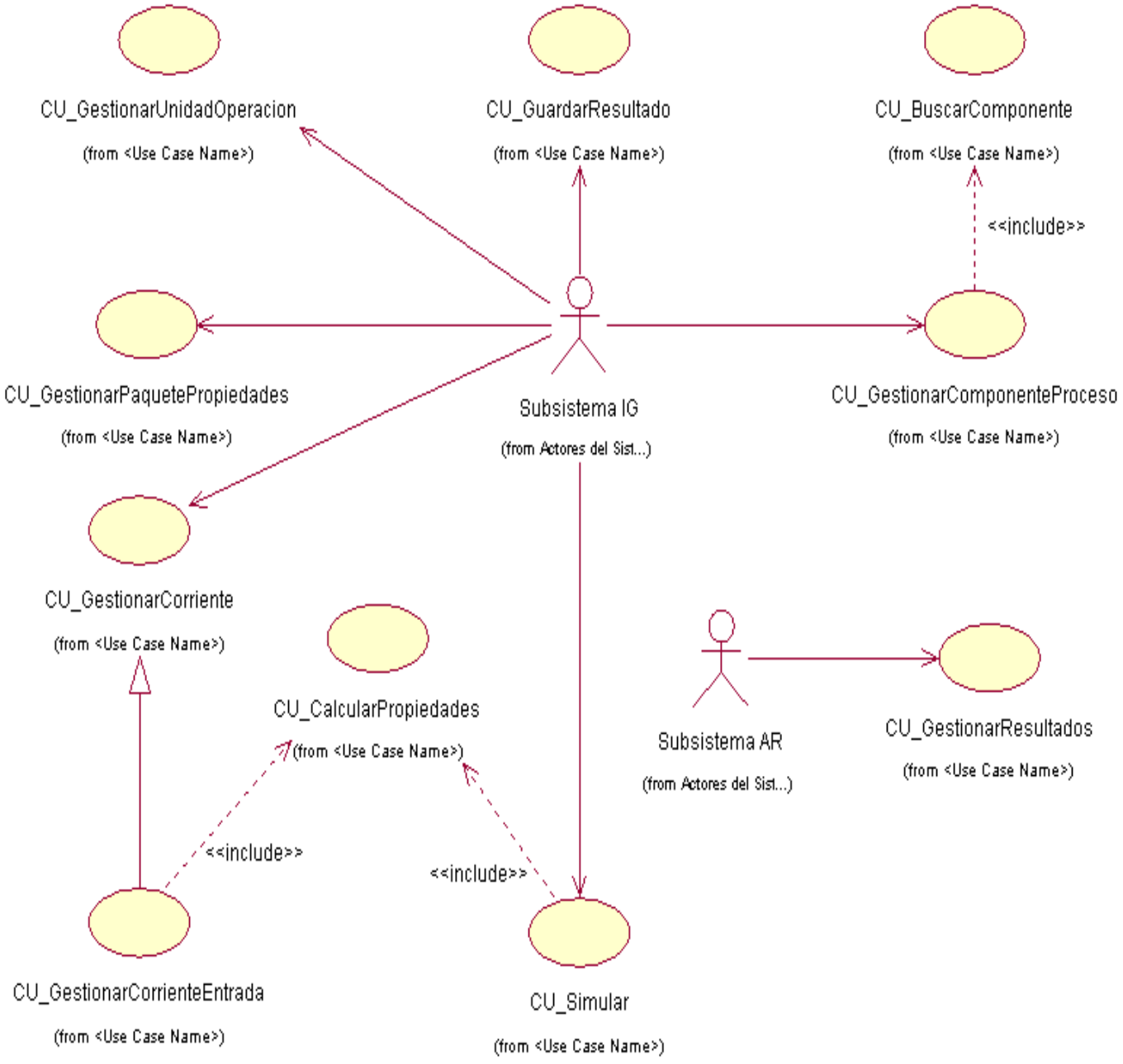

Ilustración 14: Casos de Uso arquitectónicamente Significativos

2.5.2. VISTA LÓGICA

Vista Lógica: Proporciona la base para comprender la estructura y la organización del diseño del sistema. Representa los elementos de diseño más importantes para la arquitectura. Describiendo las clases principales, su organización en paquetes y subsistemas. También describe las realizaciones de casos de uso significativos como por ejemplo las que representan aspectos dinámicos del sistema.

2.5.2.1. Elementos del modelo arquitectónicamente significantes

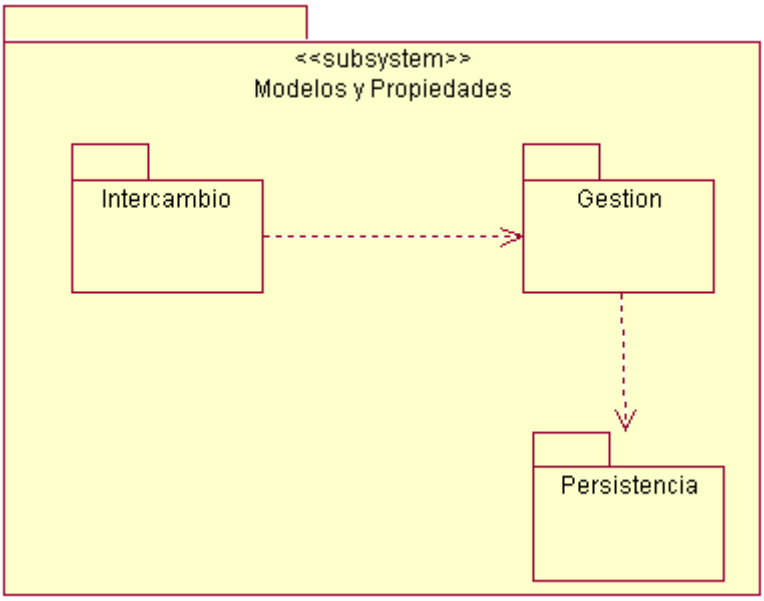

Ilustración 15: Subsistema de modelos y propiedades

El subsistema al cual se le quiere desarrollar la arquitectura propuesta es el de modelos matemáticos y propiedades, dentro del cual se agruparon algunas clases por los paquetes que se consideraron arquitectónicamente significativos para comprender mejor el diseño. Por ejemplo a la hora de llevar a cabo la simulación es necesario gestionar toda la información relacionada con las corrientes que entran y salen a una unidad de operación unitaria o módulo, teniendo en cuenta para ello el cálculo de dichas unidades, así como la estimación de las propiedades físico-químicas de estas corrientes a las cuales también es necesario gestionarle los componentes. De ahí la importancia de los paquetes Gestión y Persistencia y la relación que existe entre ellos. ¿Pero como los demás subsistemas pueden acceder a las operaciones

que necesiten del subsistema de modelos matemáticos y propiedades? Es en este punto donde entra a jugar su rol el paquete de Intercambio, donde estarán contenidas las fachadas para cada uno de estos subsistemas, ya sea el de Análisis de Resultados, o el de Interfaz Gráfica.

2.5.2.2. Visión general de la arquitectura. Alineamiento de paquetes, subsistemas y capas.

Para una mejor comprensión del diseño se ha estructurado el software en capas, donde en cada una de estas, está contenido un paquete con las clases arquitectónicamente significativas.

En este epígrafe se presenta el diseño de paquetes jerárquicamente, explicando las dependencias entre ellos, y se muestra el contenido de cada paquete de forma recursiva.

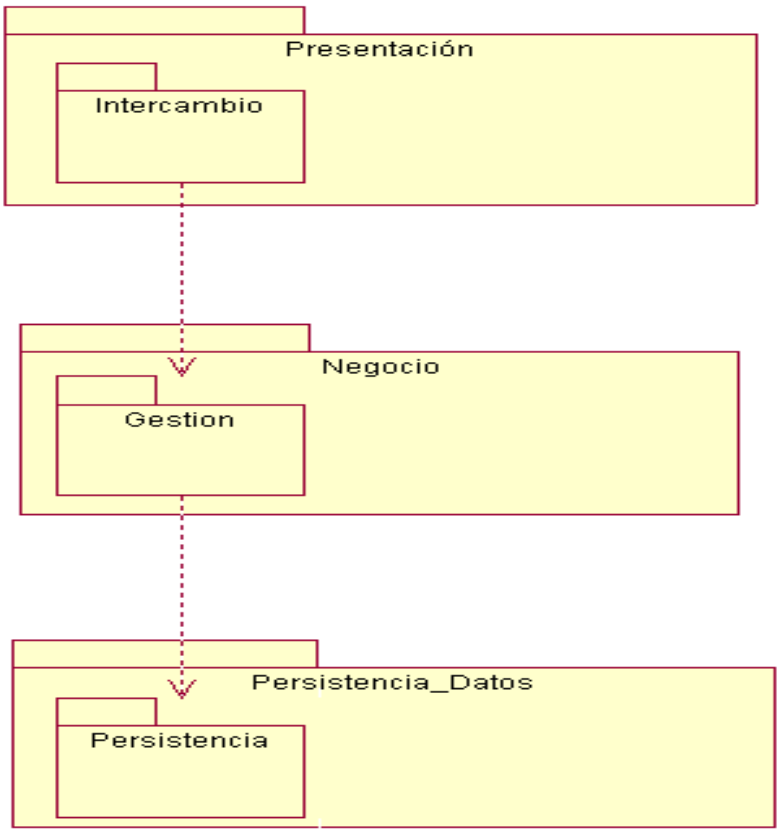

Ilustración 16: Paquetes del diseño por capas

Intercambio: Se encuentra en la capa de presentación debido a que contiene las fachadas para cada uno de los subsistemas con que interactúa el de modelos matemáticos y propiedades. Abarcando así las operaciones específicas para cada uno de estos.

Gestión: Su ubicación en la capa de negocio viene dada por la necesidad de gestionar toda la lógica central del simulador. Dígase con esto la gestión de los componentes, de las corrientes y propiedades de las mismas, necesarios para la estimación de estas últimas, lo cual conlleva realizar operaciones sobre un paquete de propiedades y complejos cálculos, no solo de estas propiedades físico-químicas, sino también de las unidades de operación unitarias, a las que entran y salen corrientes a través de diferentes puertos, los cuales especifican las propiedades de las corrientes que pasarán a través de ellos.

Persistencia: En este paquete se diseñó un esquema de persistencia, el cual constituye un conjunto reutilizable de clases que presentan servicios a los objetos persistentes. Posibilitando el almacenamiento y recuperación de objetos. Se utiliza para trabajar con bases de datos relacionales, una API de servicios de datos orientados a registros (Microsoft ODBC) u otro mecanismo de almacenamiento. Para este caso del subsistema de modelos y propiedades, se ha empleado en el tratamiento de archivos planos de la aplicación, debido a que es necesario referenciar y listar los componentes del simulador, a emplear en diversos procesos de acuerdo a sus propiedades. Este esquema está diseñado sobre una serie de patrones como el Intermediario, lo que hace posible materializar, desmaterializar y guardar objetos, algo sumamente importante en la gestión del proceso como tal, y de los resultados que se obtienen como producto de la simulación.

2.5.2.3. Realización de los Casos de Uso Significativos.

2.5.2.3.1. Paquete Intercambio

En este paquete lo más importante son las fachadas que se le ofrecen a los demás subsistemas que necesitan información del subsistema de modelos matemáticos y propiedades, siendo el caso de los ya mencionados Análisis de Resultados e Interfaz Gráfica. Como se puede apreciar en la figura, cada una de estas fachadas necesita de operaciones específicas que implementan las clases del paquete de Gestión.

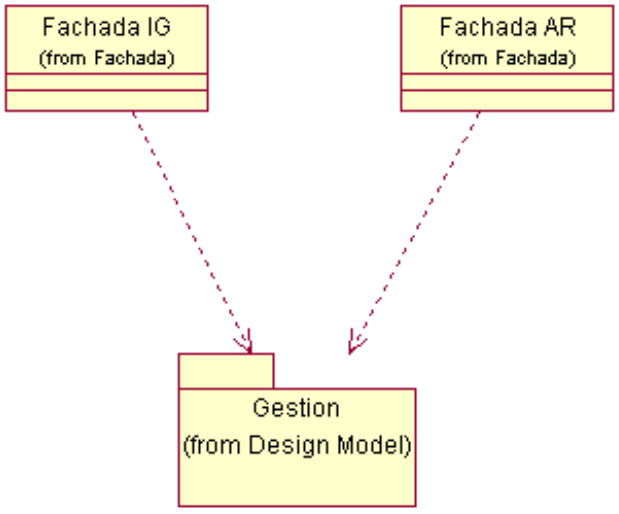

Ilustración 17: Diagrama de clases. Paquete Intercambio

2.5.2.3.2. Paquete Gestión

Contiene clases de obligada importancia para la simulación de un proceso dado como la clase controladora CC_Proceso que posibilita asignar el paquete de propiedades a utilizar, así como la clase Proceso que permite simular, crear una corriente, listar las corrientes de un determinado proceso, conectarlas a un módulo o unidad de operación, eliminarlas, calcular y obtener sus propiedades. Además de posibilitar gestionar todo lo relacionado con las unidades de operación, los paquetes de propiedades y los componentes de las corrientes antes mencionadas. Para una mejor comprensión del diseño del

sistema se llevarán a cabo la realización de los casos de uso arquitectónicamente significativos, los cuales están representados de forma general a través de las clases más importantes en el paquete descrito.

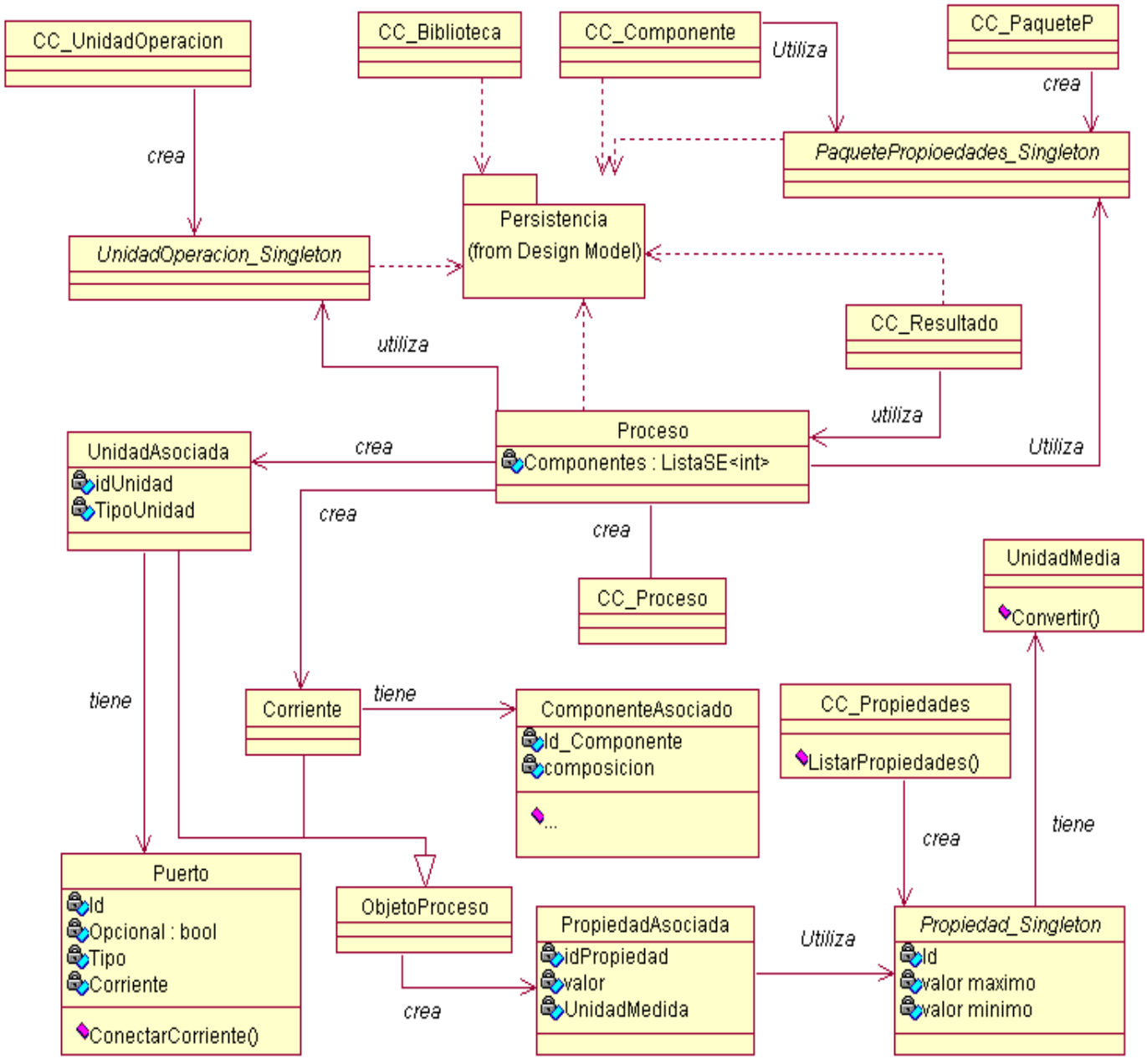

Ilustración 18: Diagrama de clases. Paquete Gestión

2.5.2.3.2.1. Realización del Caso de Uso Gestionar Corriente. Paquete Gestión.

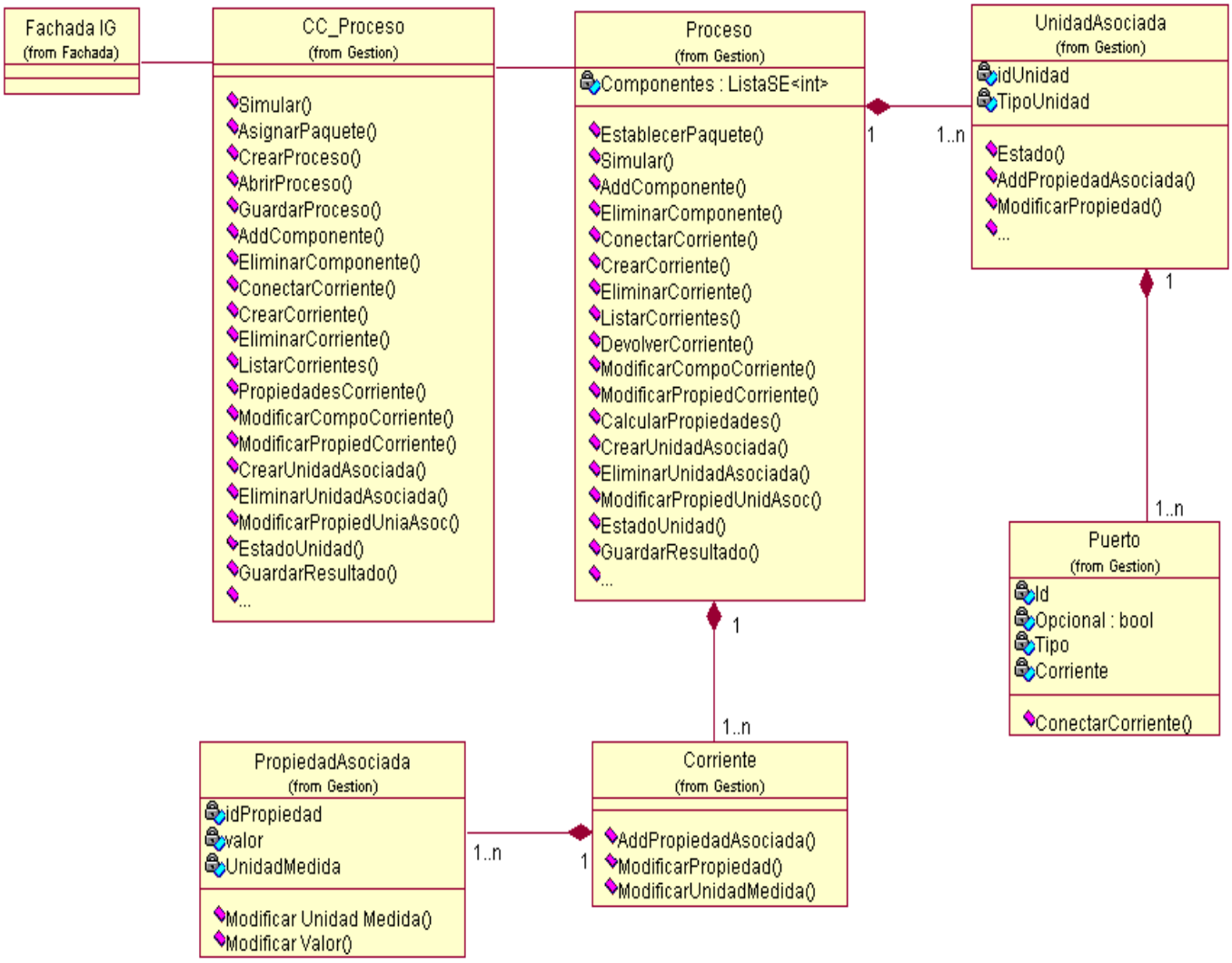

Ilustración 19: Diagrama de clases. Gestionar Corriente

Para gestionar una corriente es necesario tener en cuenta las operaciones que se realizan sobre estas, por ejemplo a la hora de crearlas es necesario ver los componentes que van a tener así como las propiedades de los mismos. Para ello se diseñaron las clases Corriente y ComponenteAsociado. Además se ha definido la clase Proceso para manejar todas las operaciones que se realizan sobres las corrientes como listarlas, obtener sus propiedades, conectarlas a las unidades de operación y eliminarlas.

Cuando se crea un determinado proceso, operación que está recogida en la clase CC_Proceso, se van adicionando objetos asociados a este y las corrientes constituyen uno de estos objetos, los cuales tienen propiedades asociadas.

Debido a esto se modelaron las clases, ObjetoProceso y PropiedadAsociada, para tener en cuenta toda la físico-química presente en las corrientes de un determinado proceso, con la manipulación de las unidades de medidas correspondientes.

Descripción de las clases con los métodos más importantes.

Nombre: CC_Proceso	
Descripción: En capsula las operaciones relacionadas con la gestión de los procesos. Tiene como principal función la llamada a los métodos de la clase Proceso.	
Métodos	**Descripción**
AsignarPaquete()	Selecciona el paquete de propiedades a emplear en el proceso.
CrearProceso(Nombre)()	Crea un nuevo proceso de simulación.
GuardarProceso(URL)	Archiva los datos de un proceso en una URL determinada.
AbrirProceso(URL)	Permite cargar la información de un proceso a partir de la URL donde fu salvado.
Simular()	Inicia el proceso de simulación, pasando el tipo de proceso e invocando el método de la clase del mismo nombre.

Tabla 2: Descripción de la clase CC_Proceso.

Nombre: Proceso	
Descripción: Es una de las clases más importantes del simulador debido a que engloba las operaciones más significativas a la hora de simular un proceso determinado.	
Atributos	**Tipo**
Componentes	ListaSE<int>
Métodos	**Descripción**
CrearCorriente ()	Adiciona una nueva corriente al proceso con todos sus atributos.
ListarCorriente ()	Devuelve un listado de las corrientes presentes en el proceso.
EliminarCorriente()	Elimina el objeto corriente con toda su información.
Simular ()	Describe los cálculos necesarios para simular todo un proceso, invocando a la simulación de los equipos, de acuerdo al tipo de proceso.
AddEquipo ()	Agrega las unidades de operación necesarias para llevar a cabo un proceso determinado.
ConocerEstadoEquipo ()	Dado un equipo invoca el método Estado () del mismo para ver si está listo para simularse o no.
ConectarCorriente ()	Establece las corrientes que entrarán a un determinado equipo del proceso.
AddComponente ()	Adiciona los componentes que serán empleados en el proceso.
EliminarComponente()	Busca los componentes y borra los objetos especificados.

Tabla 3: Descripción de la clase Proceso

Nombre: Corriente	
Descripción: Su función está dada por las operaciones de las corrientes presentes en un proceso de simulación.	
Métodos	**Descripción**
AddPropiedadAsociada ()	Agrega todas las propiedades asociadas a las corrientes.
ModificarPropiedad()	Modifica las propiedades de las corrientes a través de nuevos datos.
ModificarUnidadMedida()	Permite expresar las unidades de medida en otros valores.

Tabla 4: Descripción de la clase Corriente

Nombre: UnidadAsociada	
Descripción: Encapsula las operaciones relacionadas con las unidades de operación asociadas a un proceso determinado.	
Atributos	**Tipo**
Id	int
TipoUnidad	UnidadOperacion
estado	bool
Métodos	**Descripción**
Estado()	Determinado si un equipo está listo para simularse.
AddPropiedadAsociada()	Agrega las propiedades asociadas a una unidad determinada
ModificarPropiedad()	Permite modificar las propiedades asociadas a la unidad
CrearPuerto()	Adiciona puertos del módulo o unidad de operación unitaria.

ConectarCorriente()	Invoca al método del mismo nombre de la clase Puerto.

Tabla 5: Descripción de la clase UnidadAsociada

Nombre: Puerto	
Descripción: Contiene los atributos y operaciones de los puertos presentes en las unidades de operación.	
Atributos	**Tipo**
Id	int
Opcional	bool
Corriente	Corriente
Métodos	**Descripción**
ConectarCorriente()	Determina de acuerdo a las propiedades de las mezclas que se quieran formar, las corrientes que serán conectadas a la unidad.

Tabla 6: Descripción de la clase Puerto

Nombre: PropiedadAsociada	
Descripción: Es la responsable de todas las operaciones que se realizan sobre las propiedades asociadas a las corrientes de un proceso.	
Atributos	**Tipo**
IdPropiedad	int
valor	double
UnidadMedida	Enumerativo

Métodos	Descripción
ModificarUnidadMedida()	Permite realizar cambios sobre las unidades de medida.
ModificarValor()	Permite modificar los valores de las propiedades.

Tabla 7: Descripción de la clase PropiedadAsociada

Diagramas de secuencia por escenarios. Gestionar Corriente

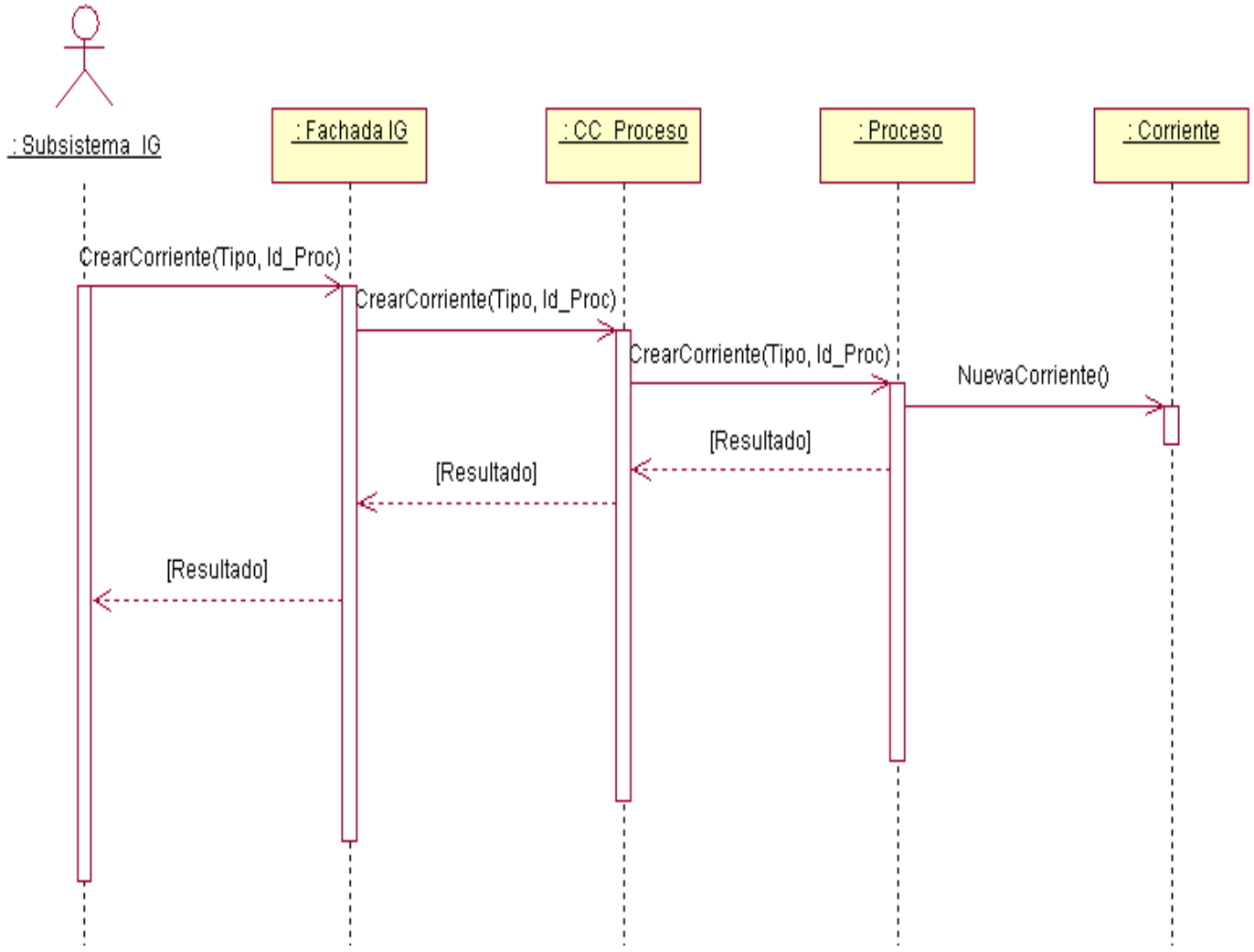

Ilustración 20: Diagrama de secuencia. Escenario: Crear Corriente.

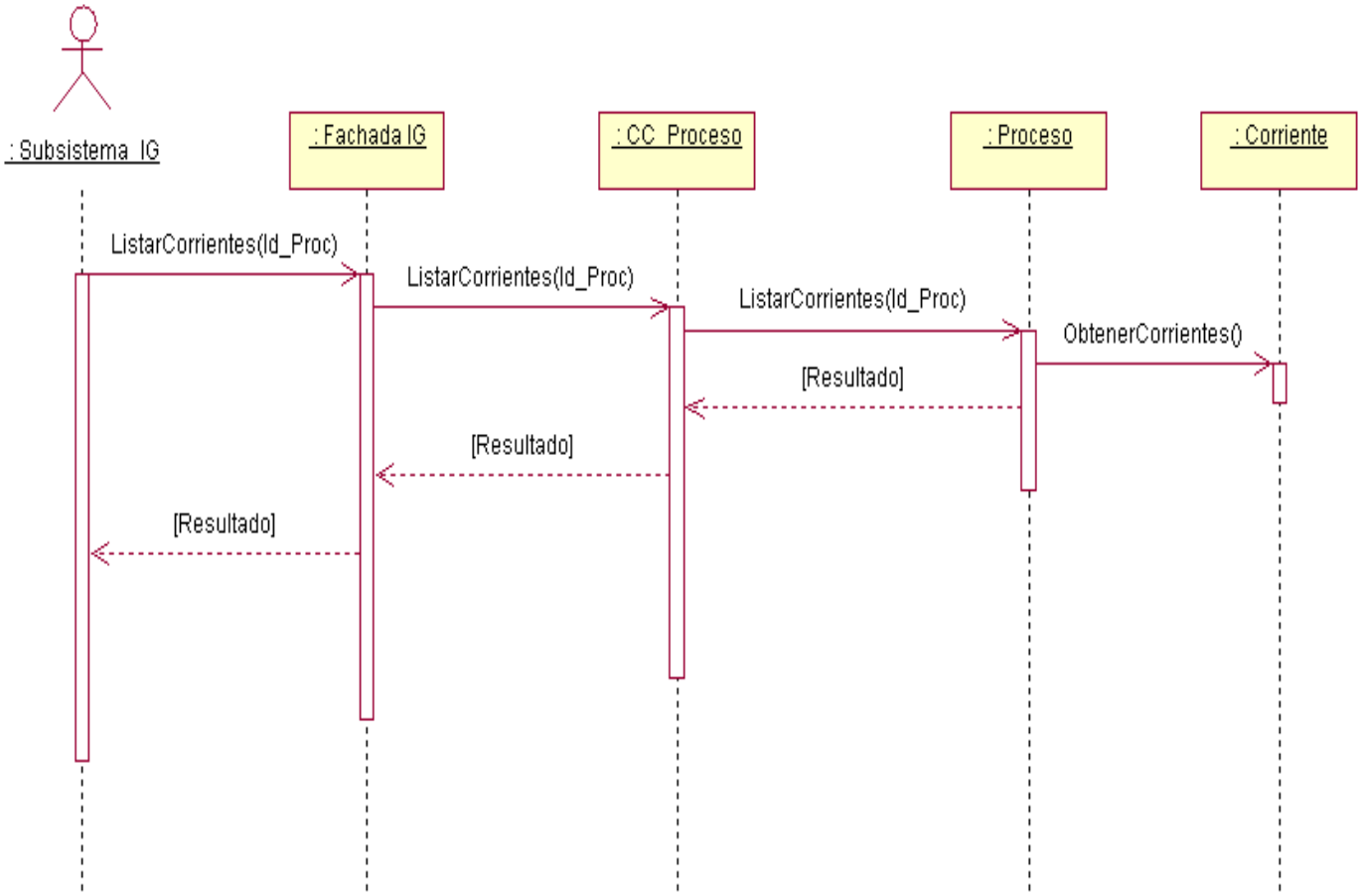

Ilustración 21: Diagrama de secuencia. Escenario: Listar Corriente.

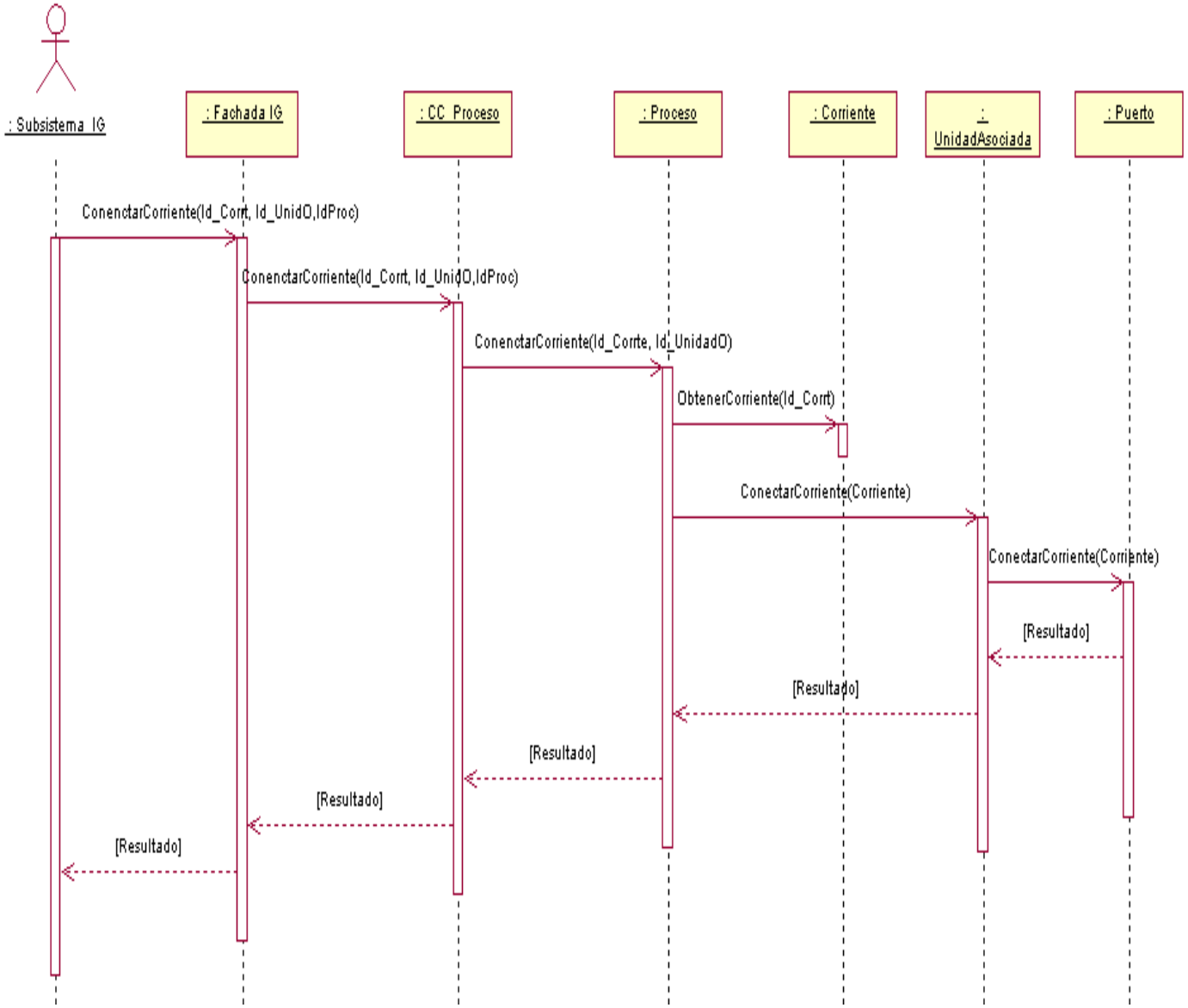

Ilustración 22: Diagrama de secuencia. Escenario: Conectar Corriente

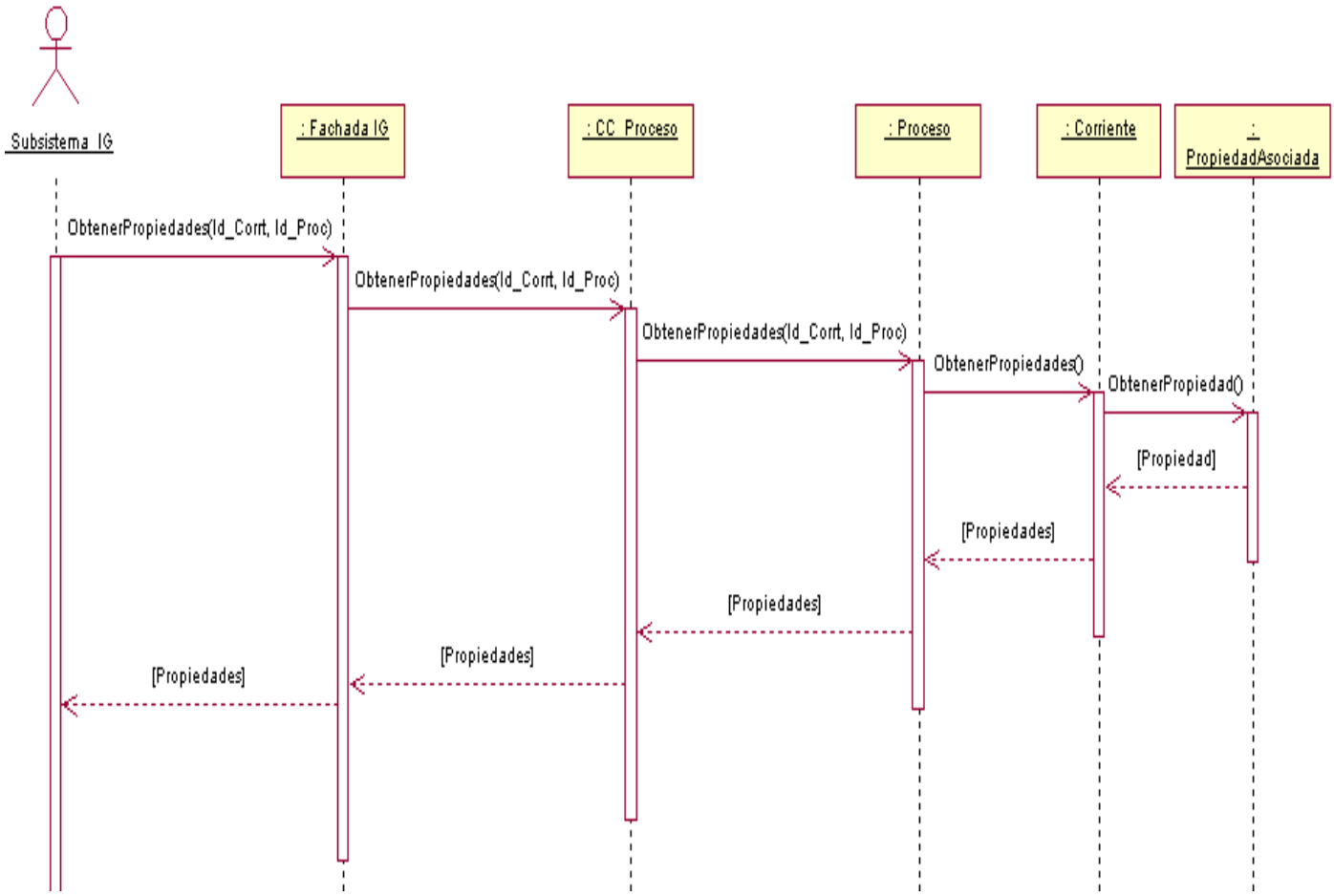

Ilustración 23: Diagrama de secuencia. Escenario: Obtener Propiedades Corriente

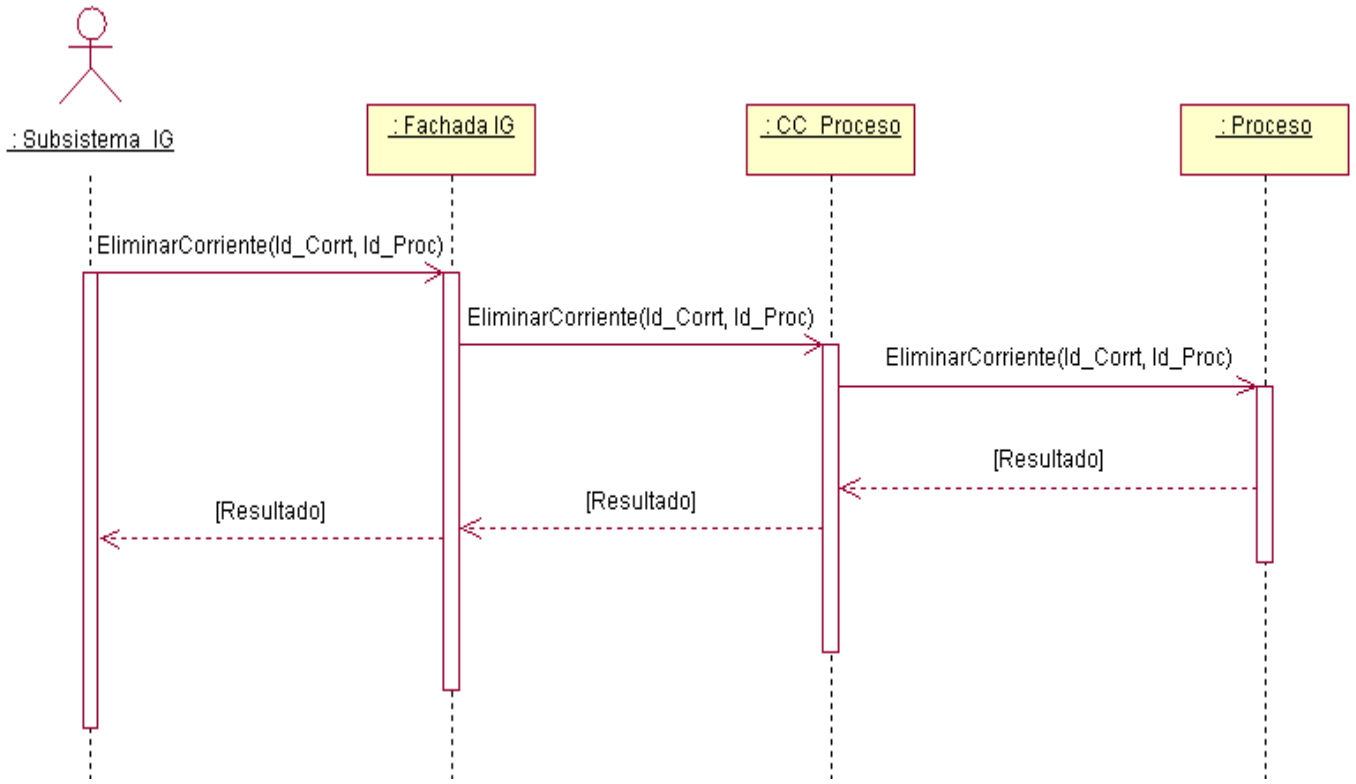

Ilustración 24: Diagrama de secuencia. Escenario: Eliminar Corriente

2.5.2.3.2.2. Realización del Caso de Uso Gestionar Corriente Entrada.

Las corrientes son un objeto necesario dentro del proceso de simulación. Para ello se ha realizado el mayor esfuerzo en garantizar la gestión de las mismas como corresponde. Pueden ser vistas de dos formas, de entrada o salidas, según sea el caso. Se hizo hincapié en las corrientes de entrada ya que las mismas heredan todas las funcionalidades de las corrientes, pero además tienen componentes asociados, lo que hace que sea posible modificar tanto su composición como sus propiedades. Por lo que están presentes en el diseño las mismas clases que en la realización del caso de uso Gestionar Corriente, incluyendo para este tipo específico la clase ComponenteAsociado.

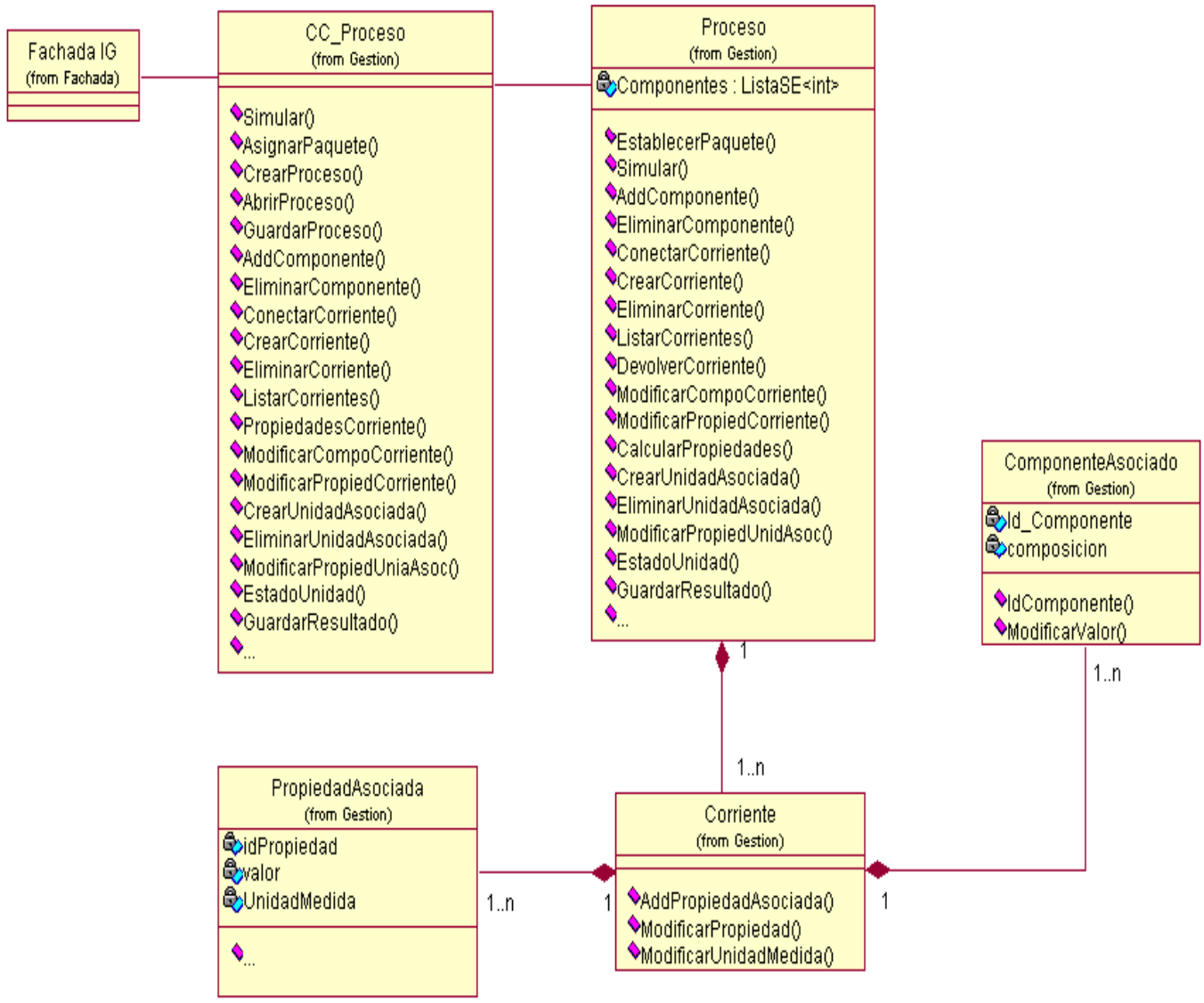

Ilustración 25: Diagrama de clases. Gestionar Corriente de Entrada.

Descripción de las clases

Nota: Ver la realización del caso de uso anterior para la descripción de las tablas CC_Proceso, Proceso, Corriente y PropiedadAsociada.

Nombre: ComponenteAsociado	
Descripción: Encierra las funcionalidades a realizar sobre los componentes presentes en las corrientes de entrada de un proceso determinado.	
Atributos	**Tipo**
Id_Componente	int
Métodos	**Descripción**
IdComponente	Devuelve el id de un componente determinado.

Tabla 8: Descripción de la clase ComponenteAsociado

Diagramas de secuencia por escenarios. Gestionar Corriente Entrada.

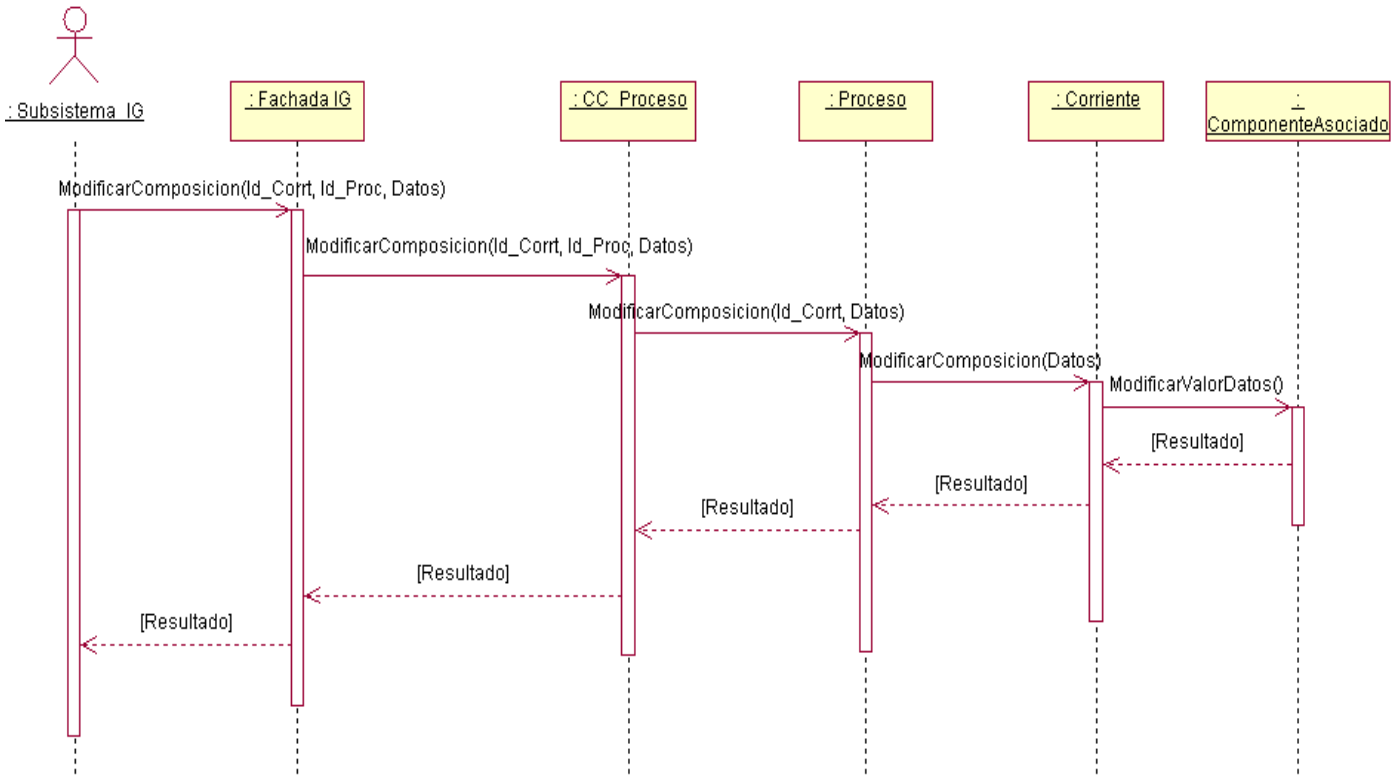

Ilustración 26: Diagrama de secuencia. Escenario: Modificar Composición.

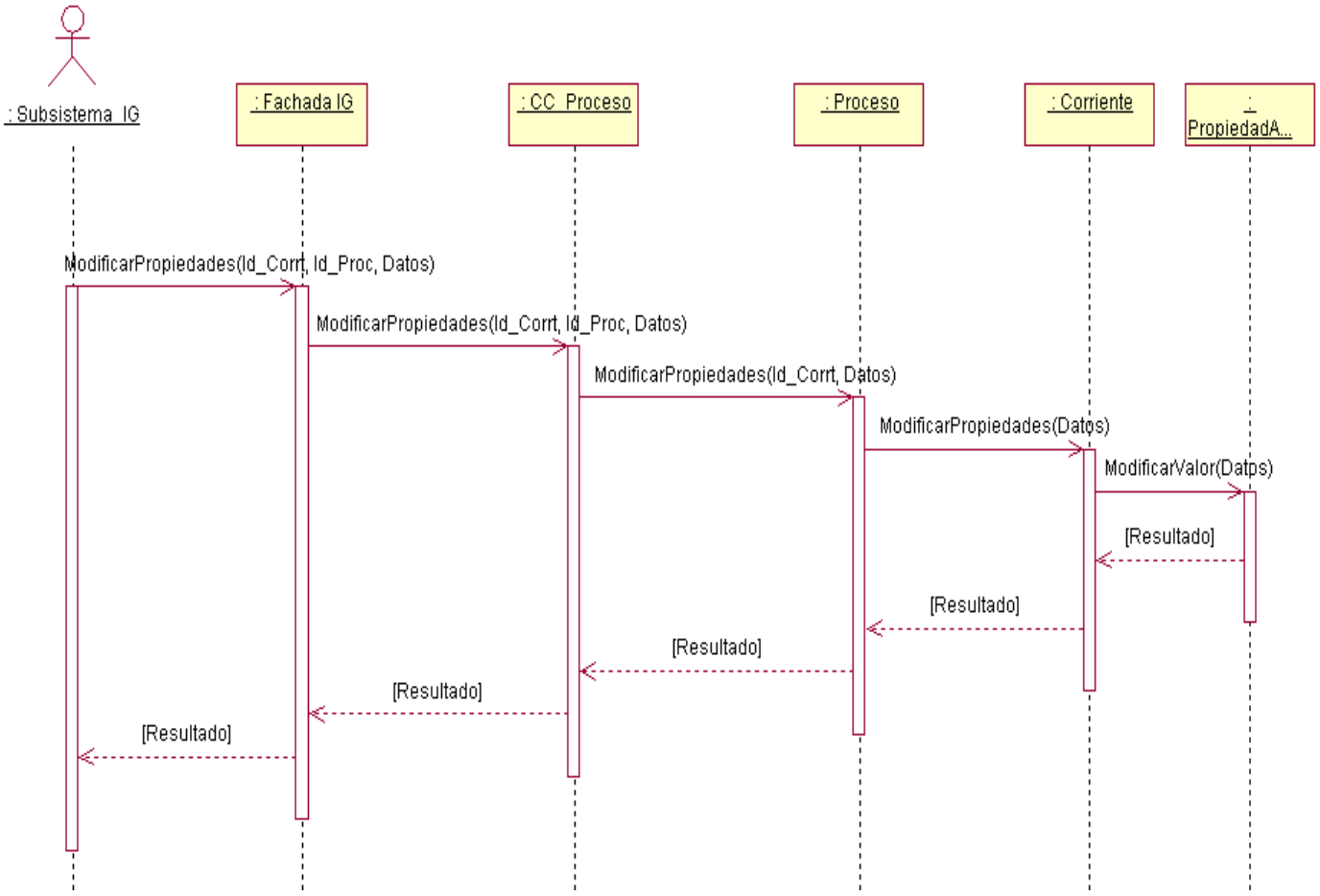

Ilustración 27: Diagrama de secuencia. Escenario: Modificar Propiedades.

2.5.2.3.2.3. Realización del Caso de Uso Gestionar Unidad de Operación Unitaria

En todo proceso de simulación es necesario gestionar lo relacionado con las unidades de operación unitarias o módulos representativos de los diferentes equipos de una fábrica. Para crear un proceso determinado se empleó la clase controladora CC_Proceso y para manejar todo lo relacionado con la gestión de las unidades, ya fuera adicionarlas, listarlas o eliminarlas, se concibió la clase Proceso. En esta última también se encuentra la operación relacionada con la simulación del equipo. Para ello es necesario conocer el estado del mismo, o sea si está listo o no para simularse, además de manejar la información relacionada con las propiedades asociadas a la unidad de operación y con este fin se incluye la clase UnidadAsociada, la cual contiene además la responsabilidad de crear los puertos a los que serán

conectados las corrientes. Teniendo en cuenta también que un tipo de unidad de operación unitaria puede ser utilizada en varios procesos, se le aplicó el patrón singleton a la clase UnidadOperación_Singleton y así obtener una sola instancia de la misma, ya que un proceso está compuesto sólo por módulos específicos, los que presentan una físico-química asociada, diseñándose entonces la clase PropiedadAsociada.

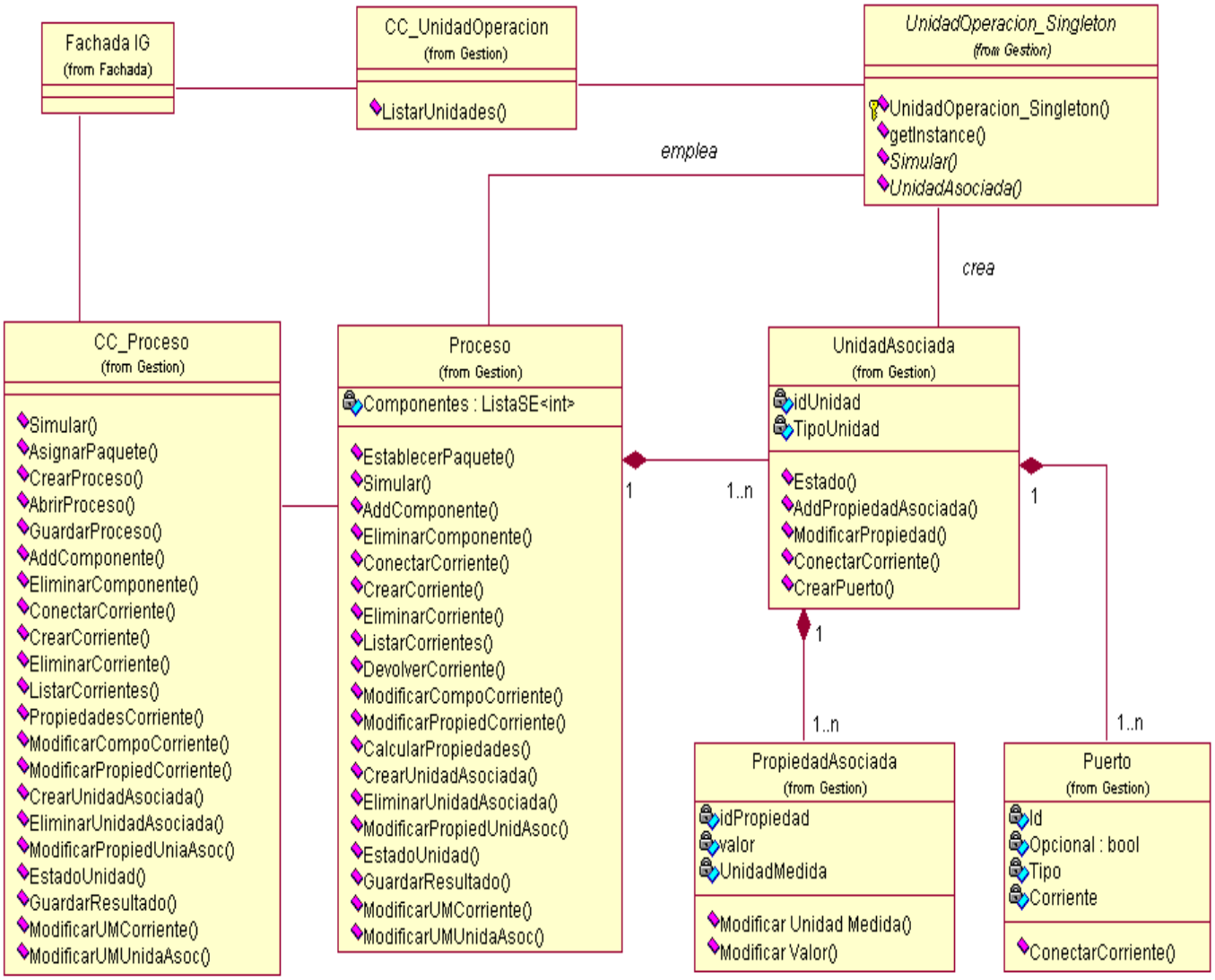

Ilustración 28: Diagrama de clases. Gestionar Unidad Operación.

Descripción de las Clases

Nota: Para la descripción de las clases CC_Proceso, Proceso, UnidadAsociada y Puerto ver la realización del caso de uso Gestionar Corriente.

Nombre: CC_UnidadOperacion	
Descripción: Su principal función es listar todos los equipos que pueden ser empleados en la simulación.	
Métodos	**Descripción**
ListarUnidades()	Devuelve un listado con las unidades que pueden ser seleccionadas para un proceso.

Tabla 9: Descripción de la clase CC_UnidadOperacion

Nombre: UnidadOperacion_Singleton	
Descripción: Su responsabilidad principal es asegurar que la clase tiene una única instancia y asegurar un punto de acceso global.	
Métodos	**Descripción**
UnidadOperacion_Singleton()	Implementación del patrón aplicado a la clase
getInstance()	Devuelve una instancia de la clase
Simular()	Calcula a través de los modelos matemáticos y los parámetros del módulo, y lleva a cabo el proceso de la simulación en el módulo.
UnidadAsociada()	Devuelve una instancia de la unidad asociada al proceso.

Tabla 10: Descripción de la clase UnidadOperacion_Singleton

Nombre: PropiedadAsociada	
Descripción: Es la responsable de todas las operaciones que se realizan sobre las propiedades asociadas a las unidades de operación.	
Atributos	**Tipo**
IdPropiedad	int
valor	double
UnidadMedida	Enumerativo
Métodos	**Descripción**
ModificarUnidadMedida()	Permite realizar cambios sobre las unidades de medida.
ModificarValor()	Permite modificar los valores de las propiedades.

Tabla 11: Descripción de la clase PropiedadAsociada a la Unidad.

Diagramas de secuencia por escenarios. Gestionar Unidad Operación.

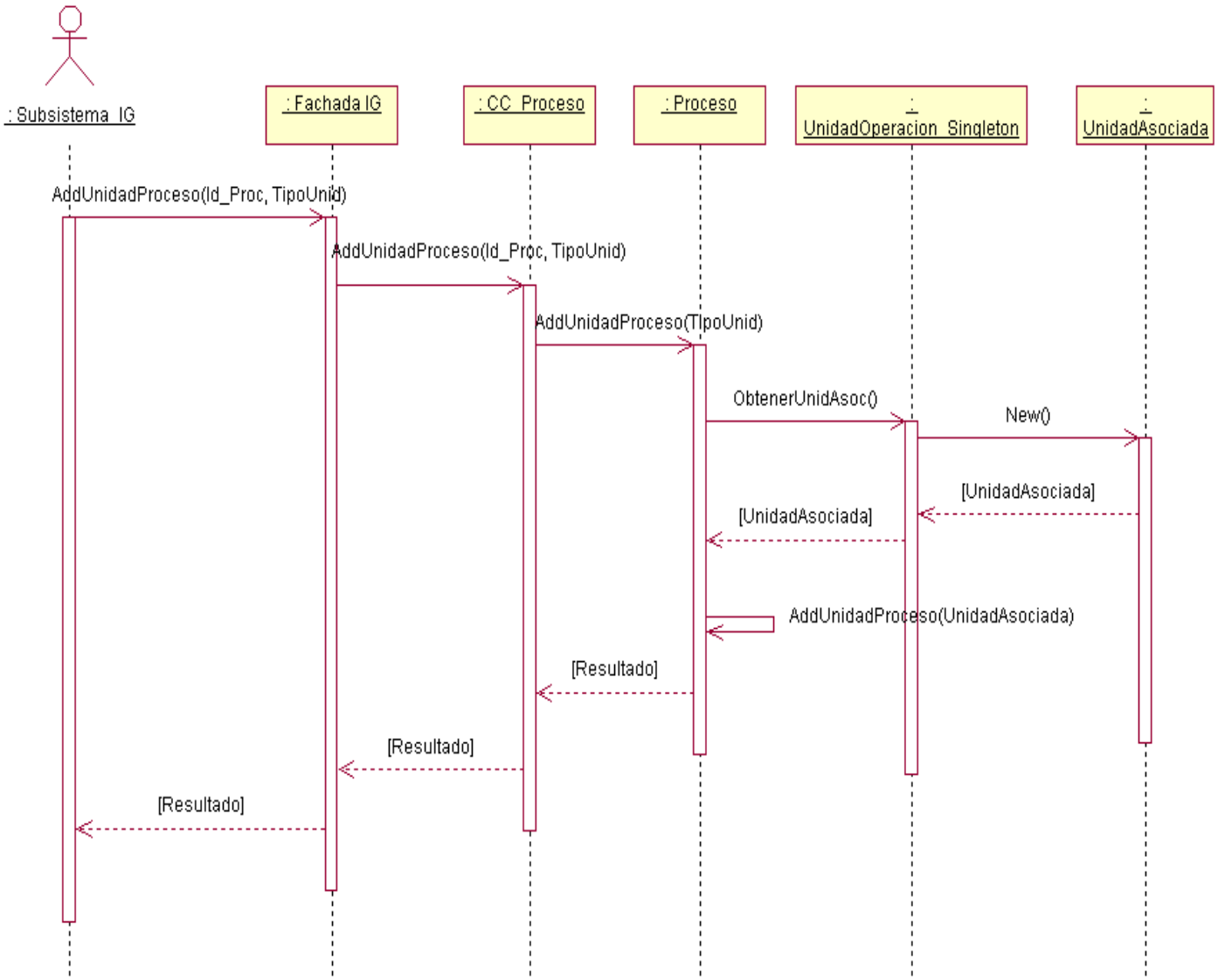

Ilustración 29: Diagrama de secuencia. Escenario: Adicionar unidad al proceso

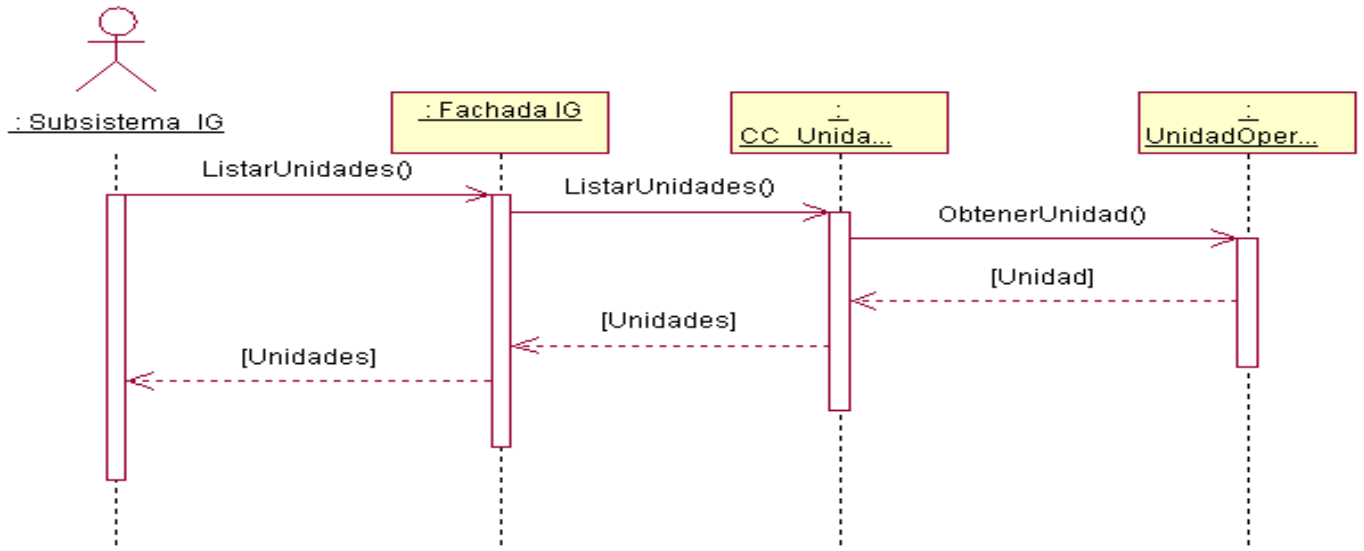

Ilustración 30: Diagrama de secuencia. Escenario: Listar unidades.

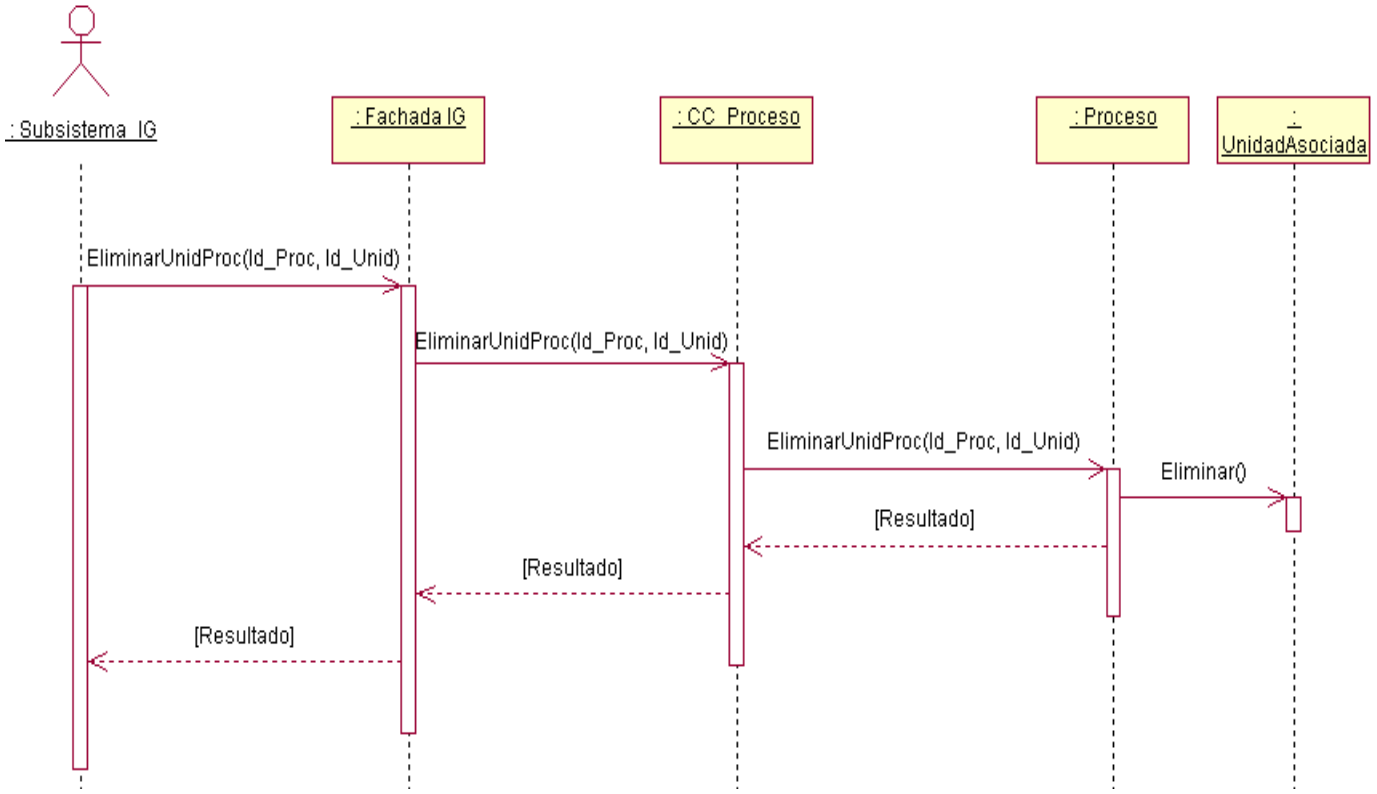

Ilustración 31: Diagrama de secuencia. Escenario: Eliminar unidades

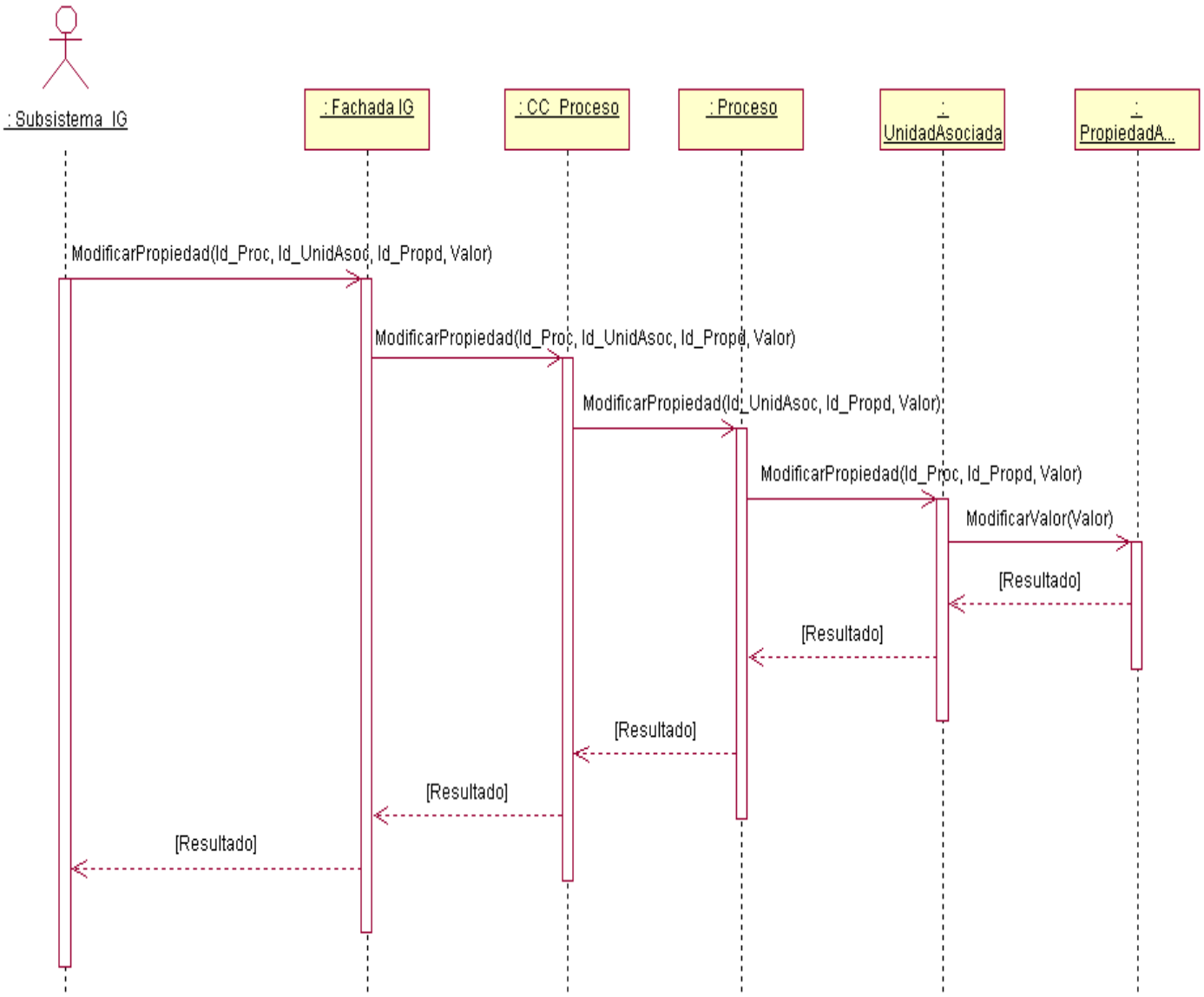

Ilustración 32: Diagrama de secuencia. Escenario: Modificar Propiedades.

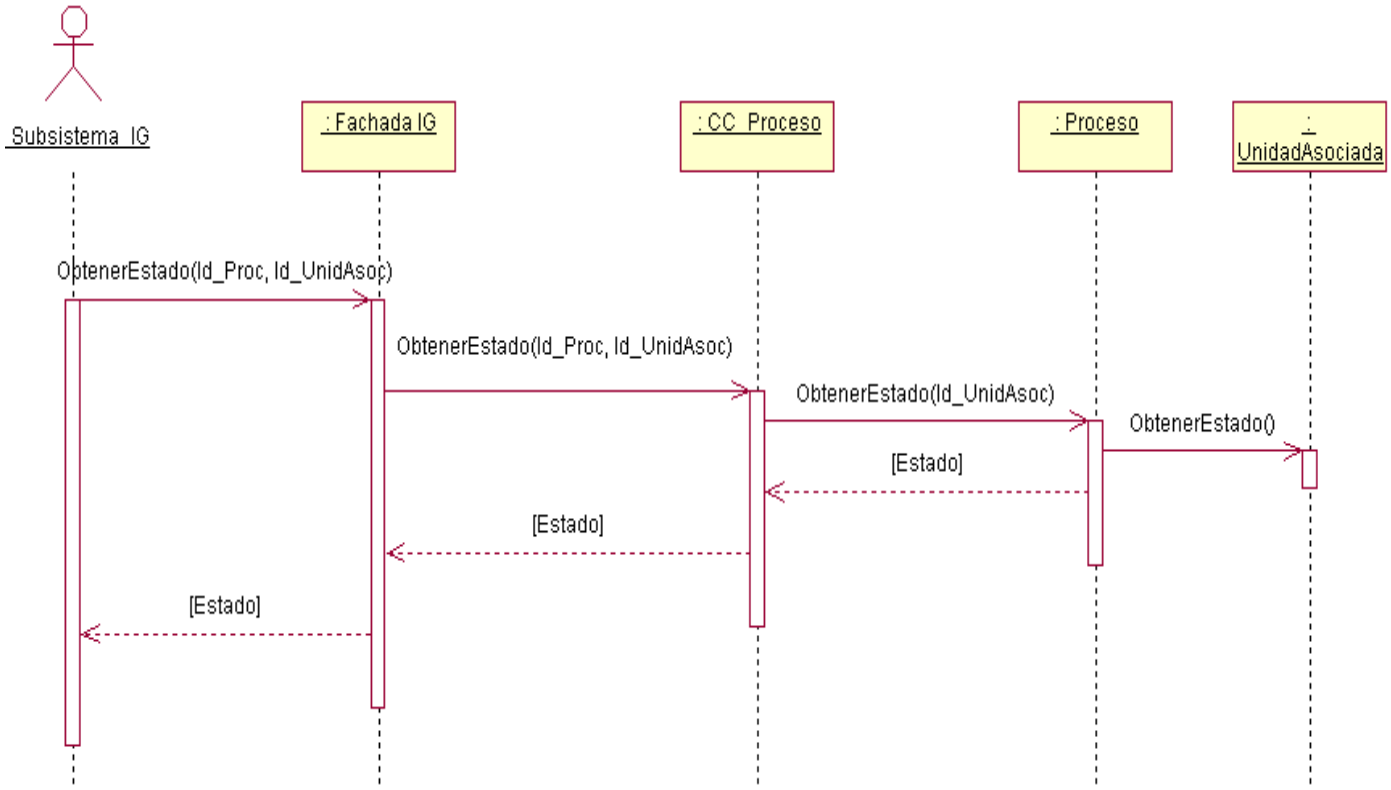

Ilustración 33: Diagrama de secuencia. Escenario: Obtener estado.

2.5.2.3.2.4. Realización del Caso Uso Gestionar Paquete de Propiedades

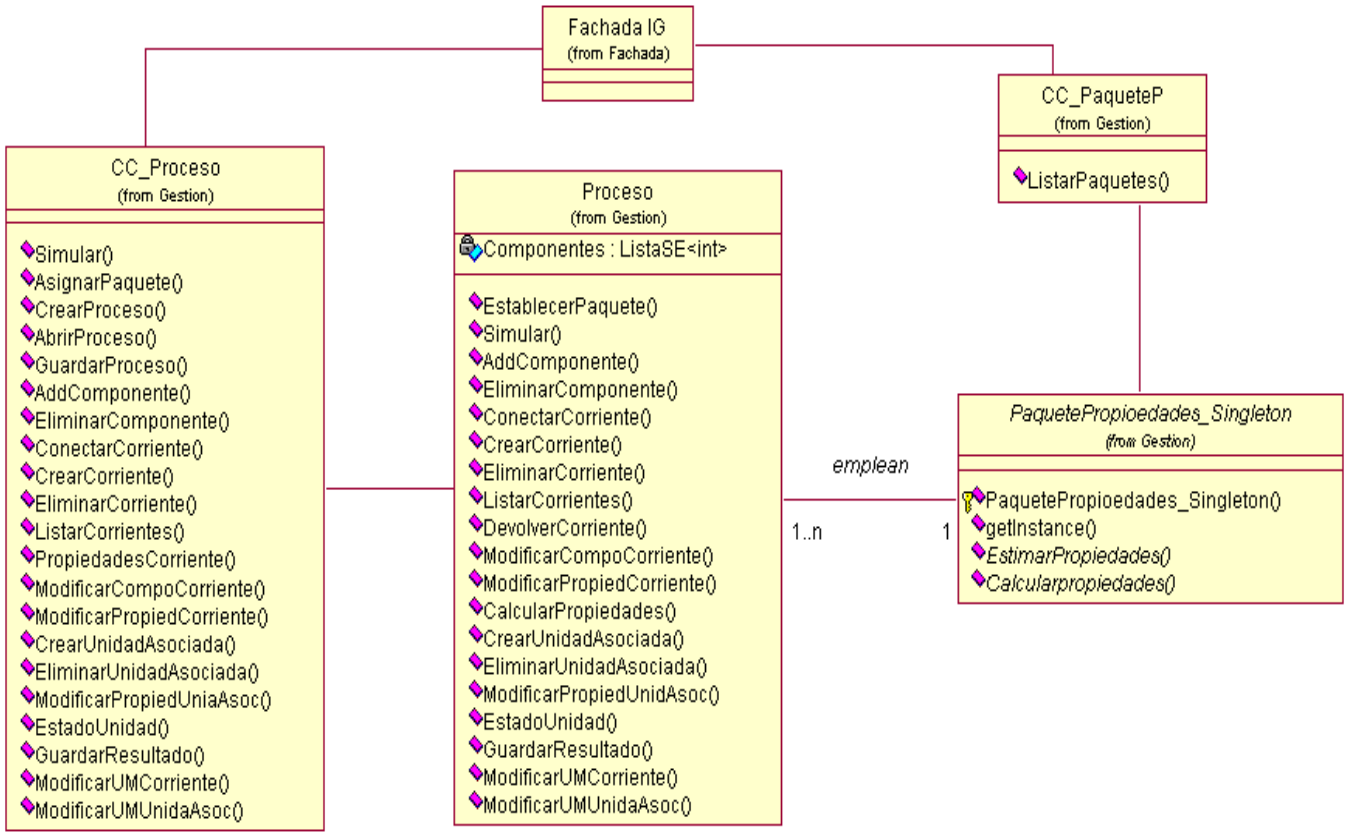

Ilustración 34: Diagrama de clases. Gestionar Paquetes de Propiedades.

En los últimos tiempos se han desarrollado una gran variedad de paquetes informáticos para resolver numéricamente sistemas de ecuaciones que se plantean en problemas de ingeniería. Dado que se deberán simular distintos tipos de mezclas de comportamiento ideal y no ideal, en las que se emplean constantes químicas y físicas, es necesario contar con un banco de modelos para la estimación de un número importante de propiedades físico-químicas.

Es por ello que se definió la clase CC_PaqueteP, la cual permite listar los paquetes que más tarde serán asignados a los diferentes procesos de acuerdo a las mezclas que se quieran formar, para luego realizar la simulación. Esta asignación la lleva a cabo la clase controladora CC_Proceso.

Un proceso solo puede emplear un paquete de propiedades. Ahora bien, el mismo paquete puede que se emplee en diferentes simulaciones, por lo que se le aplicó el patrón singleton a la clase PaquetePropiedades_Singleton y así obtener una sola instancia de la misma, para facilitar la reutilizabilidad y el bajo acoplamiento.

Descripción de las clases.

Nota: Para ver la descripción de las clases CC_Proceso y Proceso, dirigirse a la realización del caso de uso GestionarCorriente**.**

Nombre: CC_PaqueteP	
Descripción: Su función principal es listar los paquetes que serán empleados en la simulación.	
Métodos	**Descripción**
ListarPaquetes()	Devuelve un listado con los paquetes para seleccionar el que se va a emplear en un proceso, de acuerdo a los componentes del mismo.

Tabla 12: Descripción de la clase CC_PaqueteP

Nombre: PaquetePropiedades_Singleton	
Descripción: Al igual que en la clase UnidadOperacion_Singleton su función es posibilitar que la clase tenga una única instancia y asegurar un punto de acceso global. Esto se debe a que un proceso solo puede emplear un paquete de propiedades determinado por lo antes descrito	
Métodos	**Descripción**
PaquetePropioedades_Singleton()	Implementa el patrón aplicado a la clase
getInstance()	Devuelve una instancia de la clase
EstimarPropiedades()	Estimación de valores presentes en las propiedades de estos modelos.

Calcularpropiedades()	Cálculo de la físico-química asociada a los componentes

Tabla 13: Descripción de la clase PaquetePropiedades_Singleton

Diagramas de secuencia por escenarios. Gestionar Paquete de Propiedades.

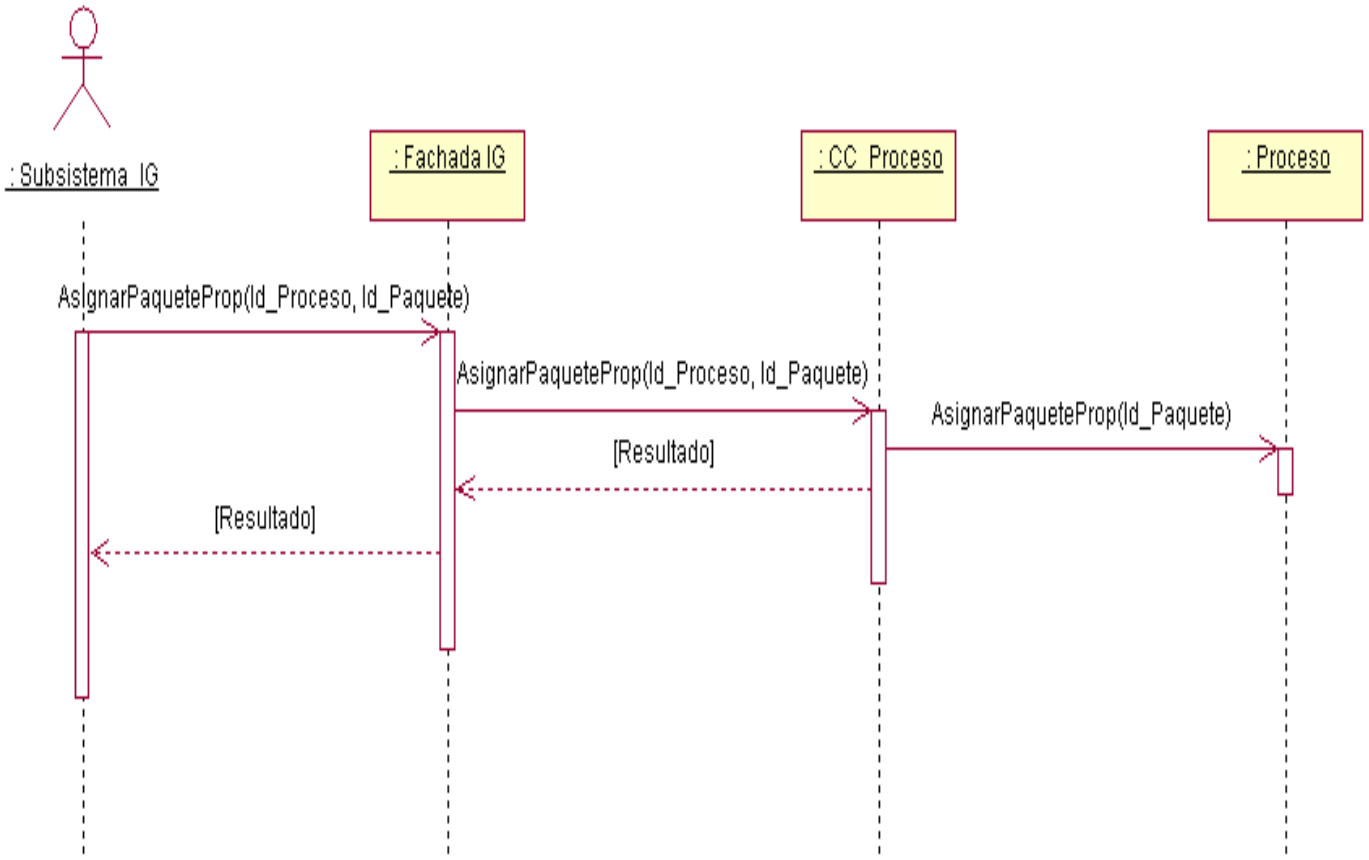

Ilustración 35: Diagrama de secuencia. Escenario: Asignar paquete.

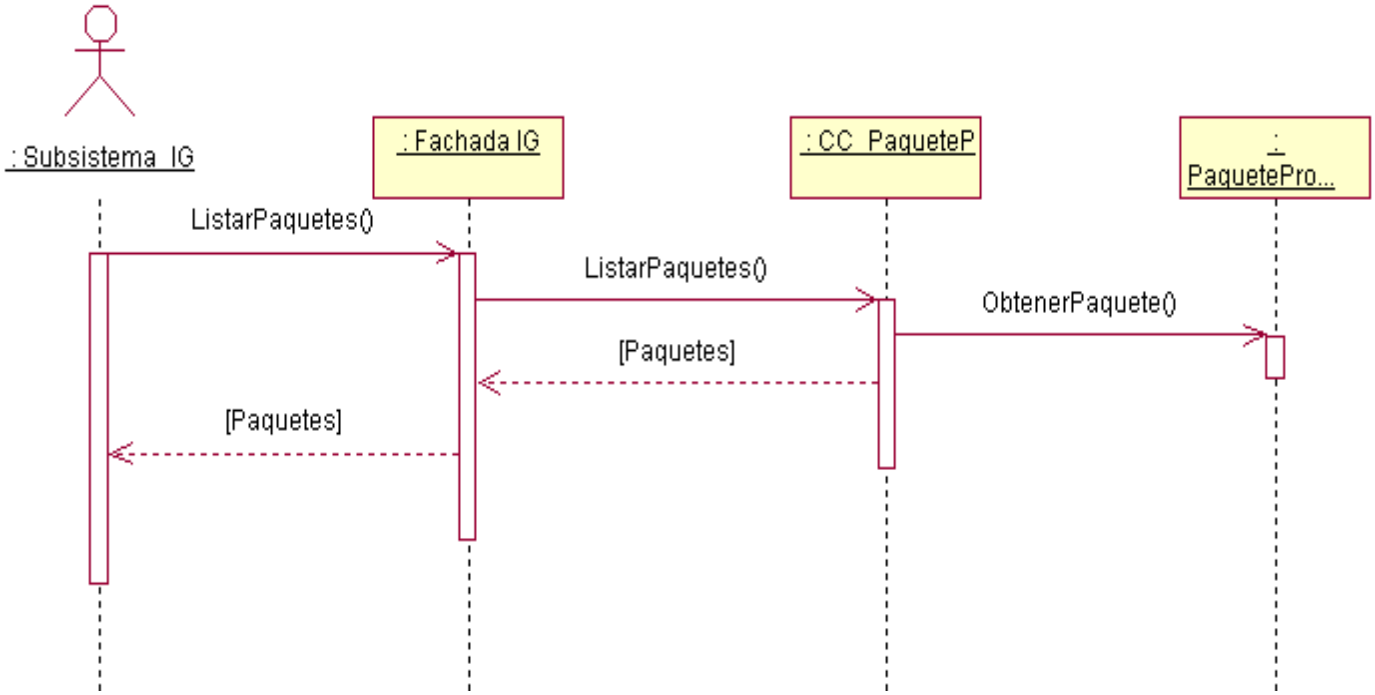

Ilustración 36: Diagrama de secuencia. Escenario: Listar paquetes.

2.5.2.3.2.5. Realización del Caso de uso Gestionar Componentes del Proceso.

Los simuladores de procesos contienen una biblioteca de componentes necesaria para formar determinadas corrientes de un proceso, así como para la selección del paquete de propiedades. En algunos simuladores comerciales son de gran utilidad, ya que una vez detectada el tipo de sustancia en la lista de componentes, se recomienda un modelo termodinámico automáticamente. Es de señalar la importancia de los componentes en todo lo que a físico-química se refiere, ya que de acuerdo a las propiedades de los mismos, se obtienen diferentes parámetros de las corrientes, que son necesarios a la hora de simular un módulo determinado. Con este fin se ha diseñado la clase controladora CC_Componente, encargada de gestionar todo lo que atañe a los componentes, o sea, listarlos, asignarlos, incluyendo además la estimación de propiedades. Esta clase interactúa directamente con el paquete de Persistencia donde van a estar materializados los componentes a emplear en la simulación.

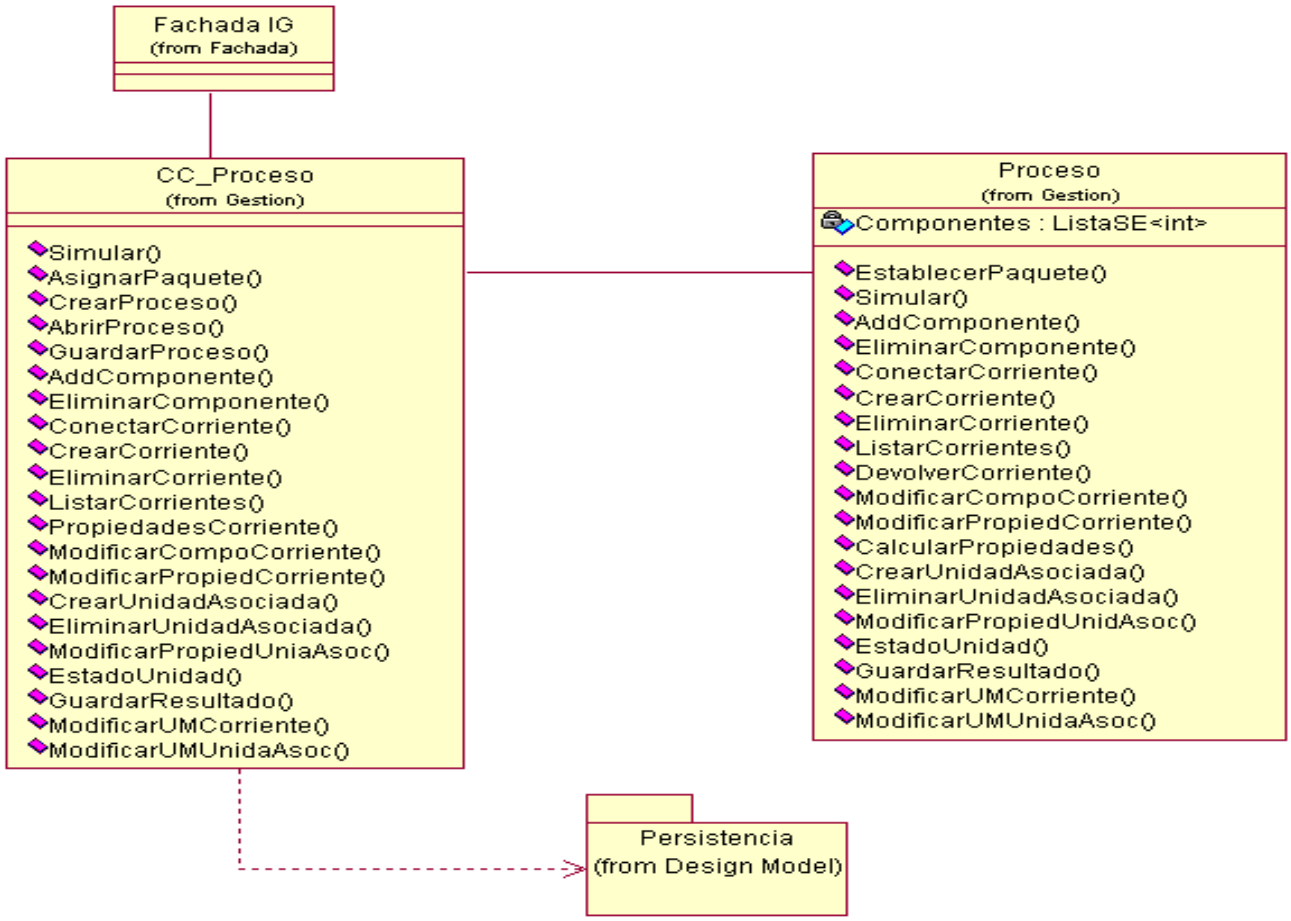

Ilustración 37: Diagrama de clases. Gestionar Componente

Descripción de las clases.

Nota: Para ver la descripción de las clases Proceso y CC_Proceso, remitirse a la realización del caso de uso Gestionar Corriente.

Diagramas de secuencia por escenarios.

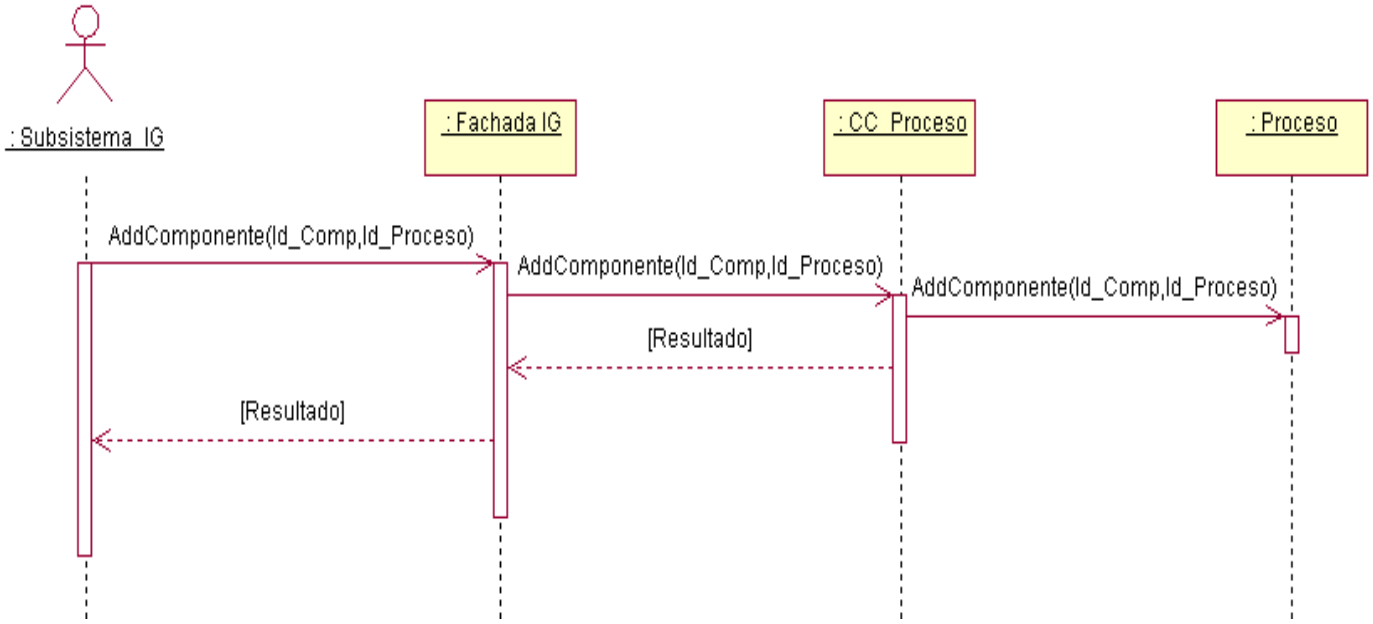

Ilustración 38: Diagrama de secuencia. Escenario: Adicionar Componente.

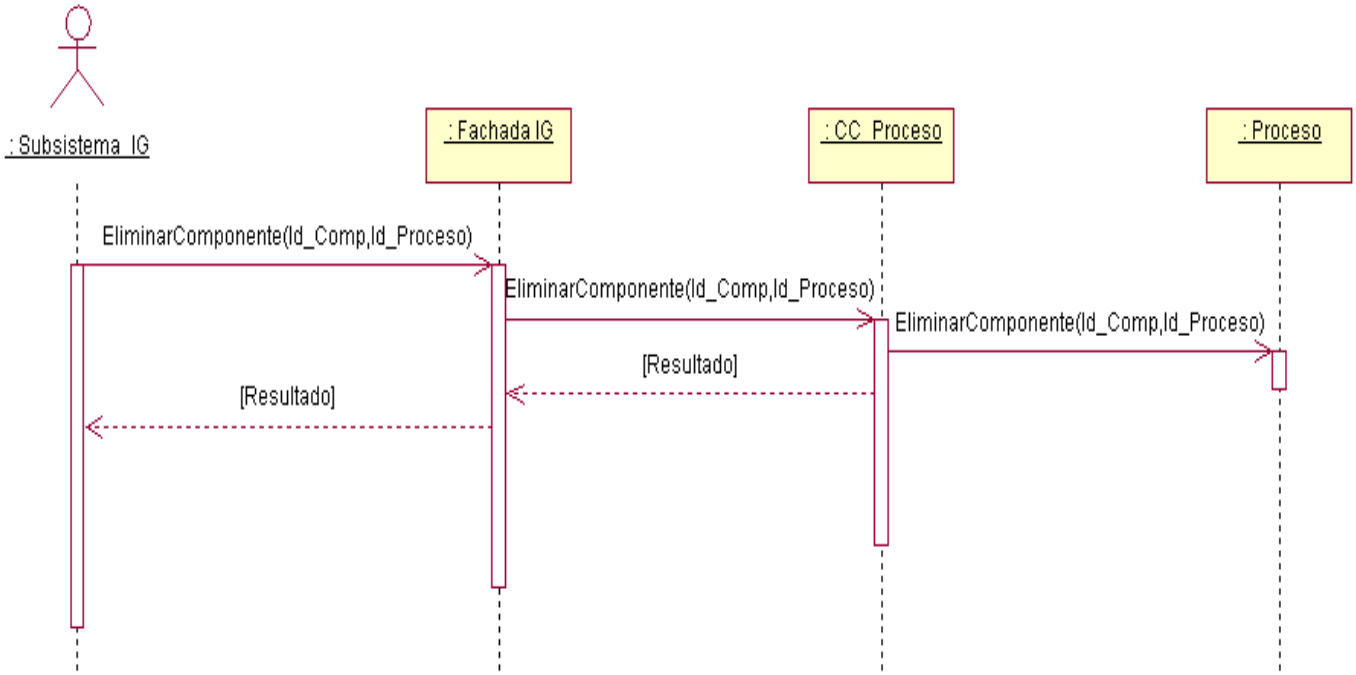

Ilustración 39: Diagrama de secuencia. Escenario: Eliminar Componente.

2.5.2.3.2.6. Realización del Caso de Uso Buscar Componentes. Paquete Gestión.

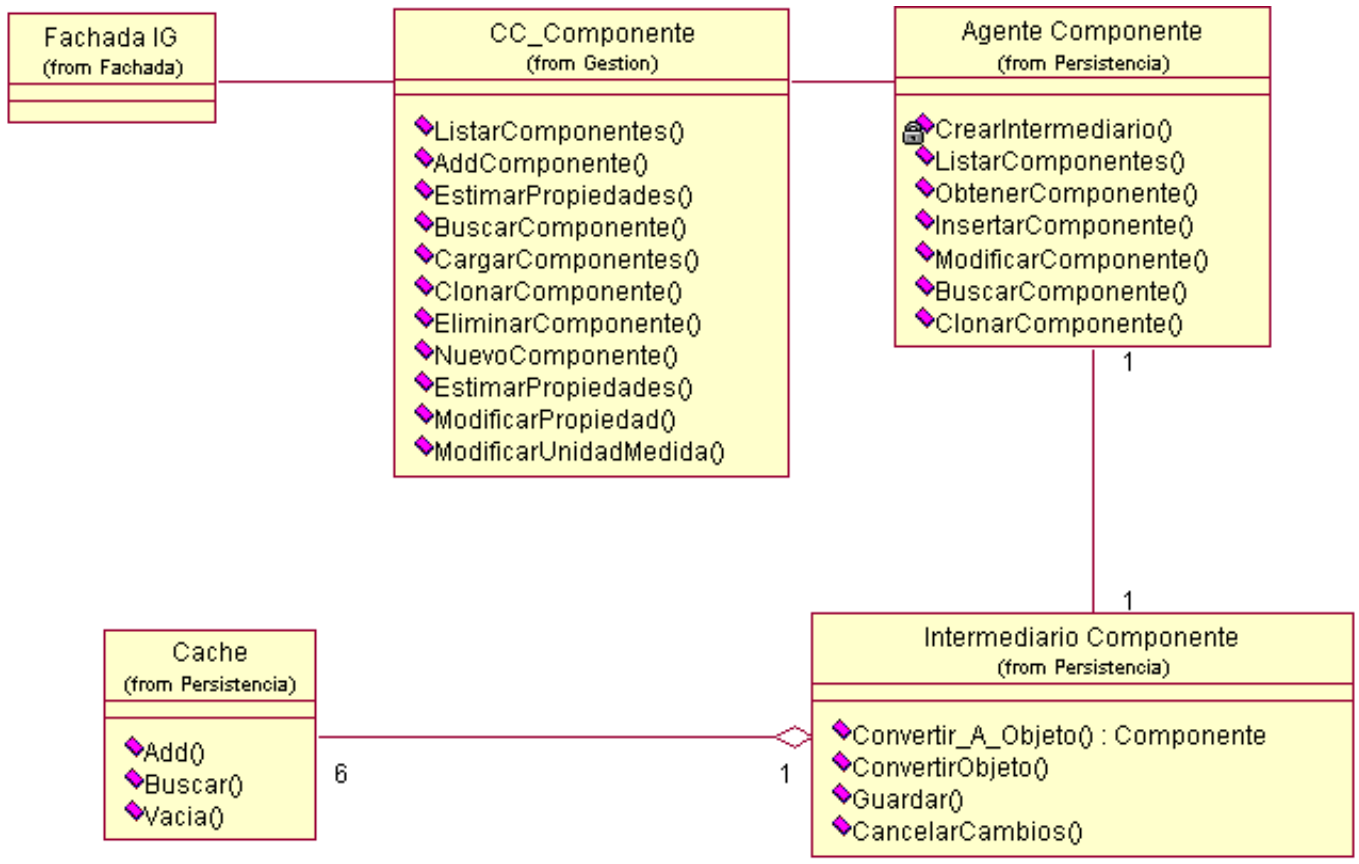

Ilustración 40: Diagrama de clases. Buscar Componentes

Descripción de las clases.

Nombre: CC_Componente	
Descripción: Encargada de gestionar lo relacionados con los componentes del simulador.	
Métodos	**Descripción**
ListarComponentes()	Lista los componentes que pueden ser empleados en un proceso, invocando al método del mismo nombre de la clase

	AgenteComponente
AddComponente()	Verifica que el componente no esté materializado y lo adiciona a la caché.
EstimarPropiedades()	Permite estimar los valores de las propiedades de los componentes

Tabla 14: Descripción de la clase CC_Componente

Nombre: AgenteComponente	
Descripción: Su función principal es que es empleado para comunicar la clase controladora CC_Componente con el paquete de Persistencia.	
Métodos	**Descripción**
CrearIntermediario()	Crea una instancia de una clase que se encarga de materializar, desmaterializar y guardar un objeto en otro objeto caché (Clase intermediaria).
ListarComponentes()	Devuelve un listado con los componentes del simulador
ObtenerComponente()	Busca un objeto y lo referencia, comprobando que ya esté materializado para no hacerlo de nuevo.

Tabla 15: Descripción de la clase AgenteComponente

Nombre: IntermediarioComponente	
Descripción: Clase encargada de materializar, desmaterializar y guardar un objeto en otro objeto caché, o sea, administra la caché.	
Métodos	**Descripción**
Convertir_A_Objeto()	Convierte una representación no orientada a objetos como los registros y datos, en objetos.

Guardar()	Lleva un registro de los cambios que sufre un objeto, realizándose las actualizaciones necesarias.
CancelarCambios()	Elimina todos los cambios registrados.

Tabla 16: Descripción de la clase Intermediario Componente

Nombre: Cache	
Descripción: Objeto creado para almacenar los objetos materializados, los cuales dejan su identificador en la caché, facilitando la velocidad en dicho almacenamiento.	
Métodos	**Descripción**
Add(Objeto)	Materializa un objeto.
Buscar(Id : int)	Devuelve el objeto referenciado a partir de su id.
Vacia()	Determina si está vacía o no la caché.

Tabla 17: Descripción de la clase Cache

Diagramas de secuencia.

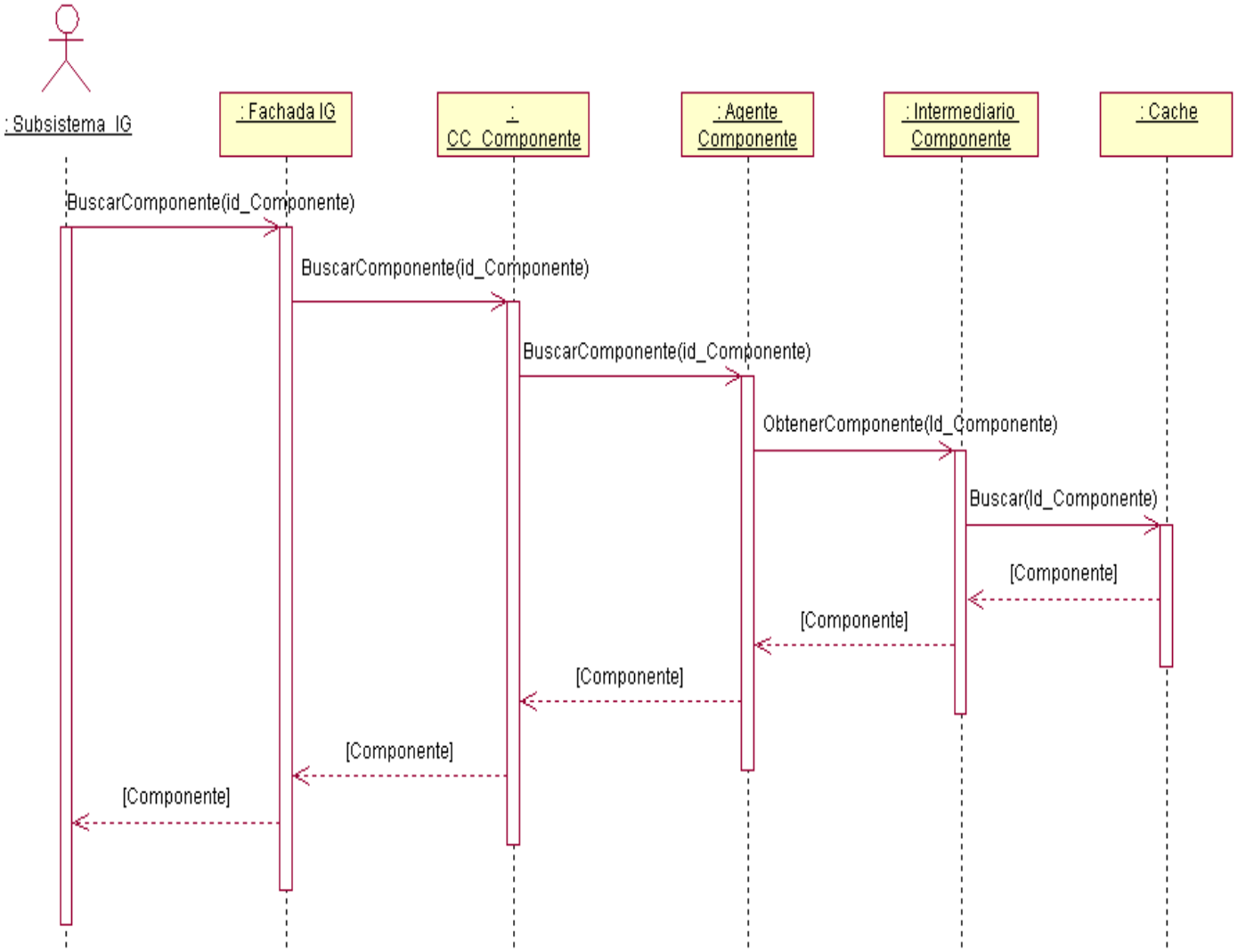

Ilustración 41: Diagrama de secuencia. Buscar Componente

2.5.2.3.2.7. Realización del Caso de Uso Calcular Propiedades. Paquete Gestión.

Los simuladores industriales de calidad se caracterizan por manejar gran cantidad de información, donde un diagrama de flujo de información está formado por los módulos de cálculo que representan lo que ocurre en los equipos de una planta y las corrientes que introducen y sacan información de dichos módulos. Es en este punto donde el cálculo de propiedades entra a jugar un papel fundamental a la hora de simular distintas operaciones o equipos de un proceso, pues se hace necesario estimar toda la físico-química asociada tanto a las unidades de operación unitarias, como a las corrientes antes mencionadas. Es por ello que el cálculo de propiedades constituye un aspecto arquitectónicamente significativo tanto para el análisis como para el diseño.

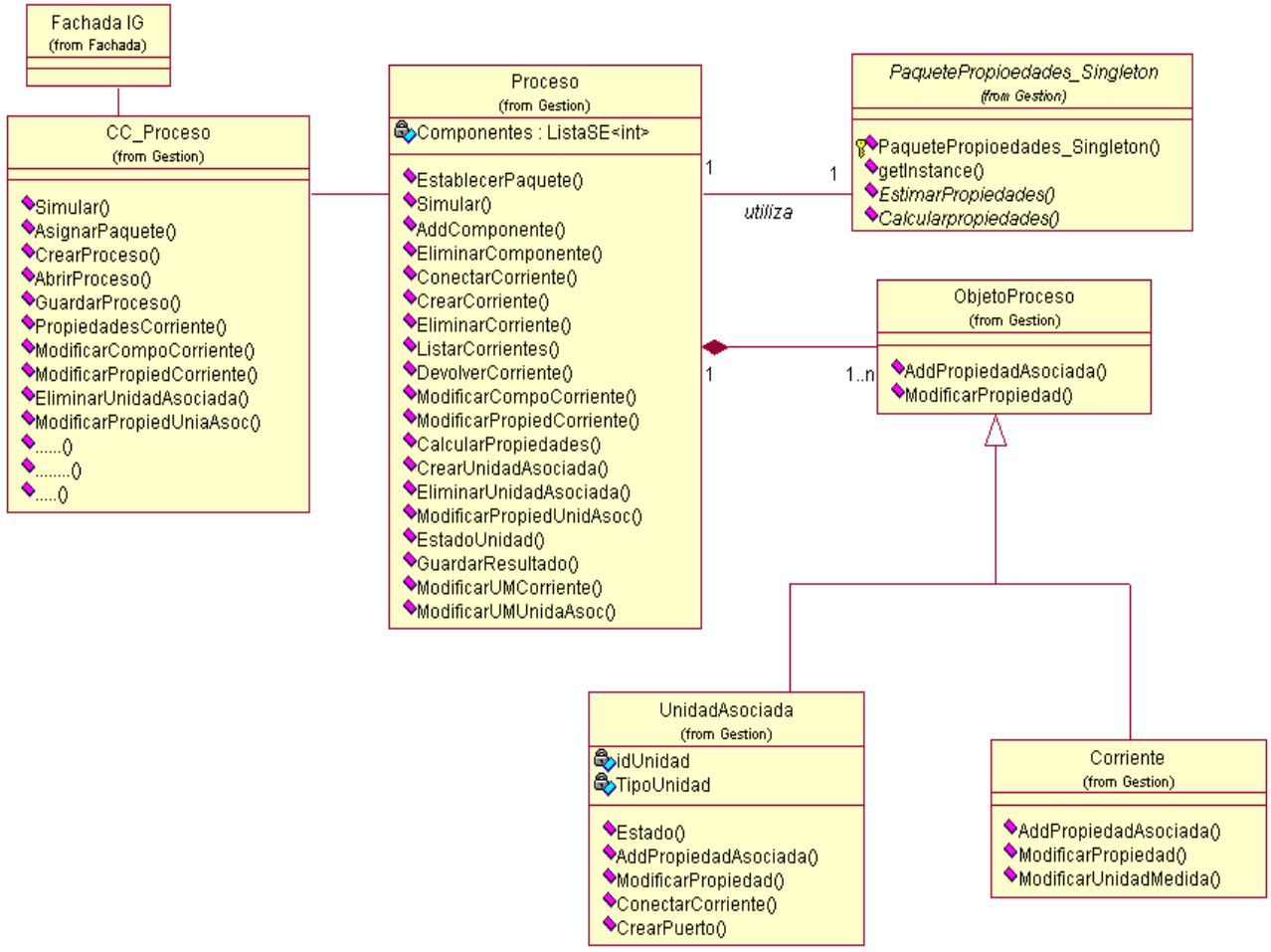

Ilustración 42: Diagrama de clases. Calcular Propiedades

Descripción de las clases.

El comportamiento de este caso de uso es manejado explícitamente por las corrientes presentes en el proceso y el paquete de propiedades seleccionado.

De ahí que la descripción de las clases presentes en este guión, se encuentren en la realización de los casos de uso GestionarCorriente y Gestionar Paquete de Propiedades del proceso. Solo se agrega la clase ObjetoProceso, con el fin de garantizar un bajo acoplamiento.

Nombre: ObjetoProceso	
Descripción: Clase creada con el fin de garantizar un bajo acoplamiento en el diseño y sobre todo la reutilizabilidad, debido a que de ella heredan los diferentes objetos del proceso como son las corrientes y los módulos. Es responsable de las operaciones que se realizan sobre las propiedades de los objetos de un proceso.	
Métodos	**Descripción**
AddPropiedadAsociada()	Agrega las propiedades asociadas a un objeto en dependencia del tipo, o sea, corrientes o unidades de operación. Por lo que es virtual.
ModificarPropiedad()	Modifica las propiedades de un objeto de acuerdo a su tipo, o sea, corrientes o unidades de operación. Es virtual.

Tabla 18: Descripción de la clase ObjetoProceso

Diagrama de secuencia.

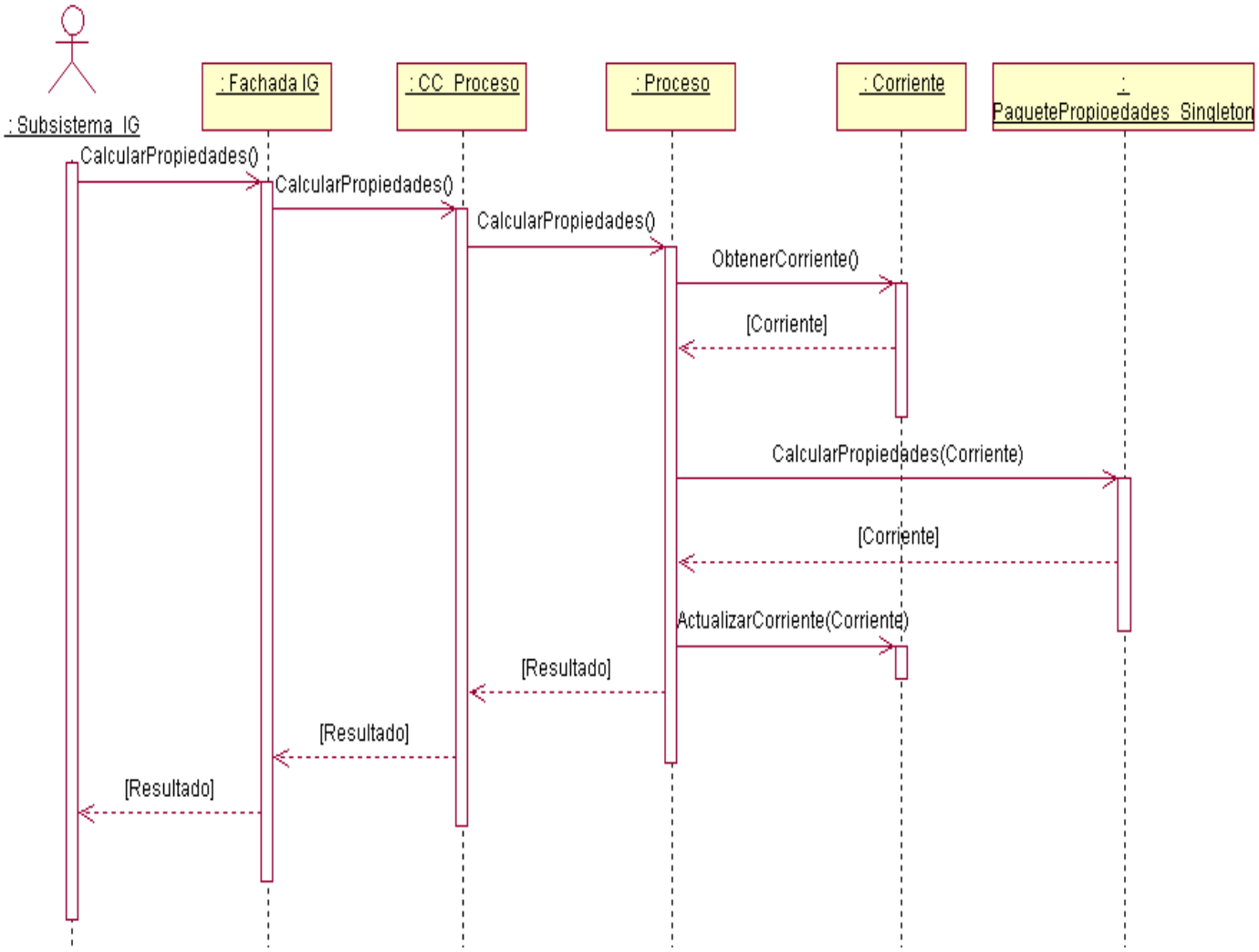

Ilustración 43: Diagrama de secuencia. Calcular Propiedades

2.5.2.3.2.8. Realización del Caso de Uso Guardar Resultados. Paquete Gestión.

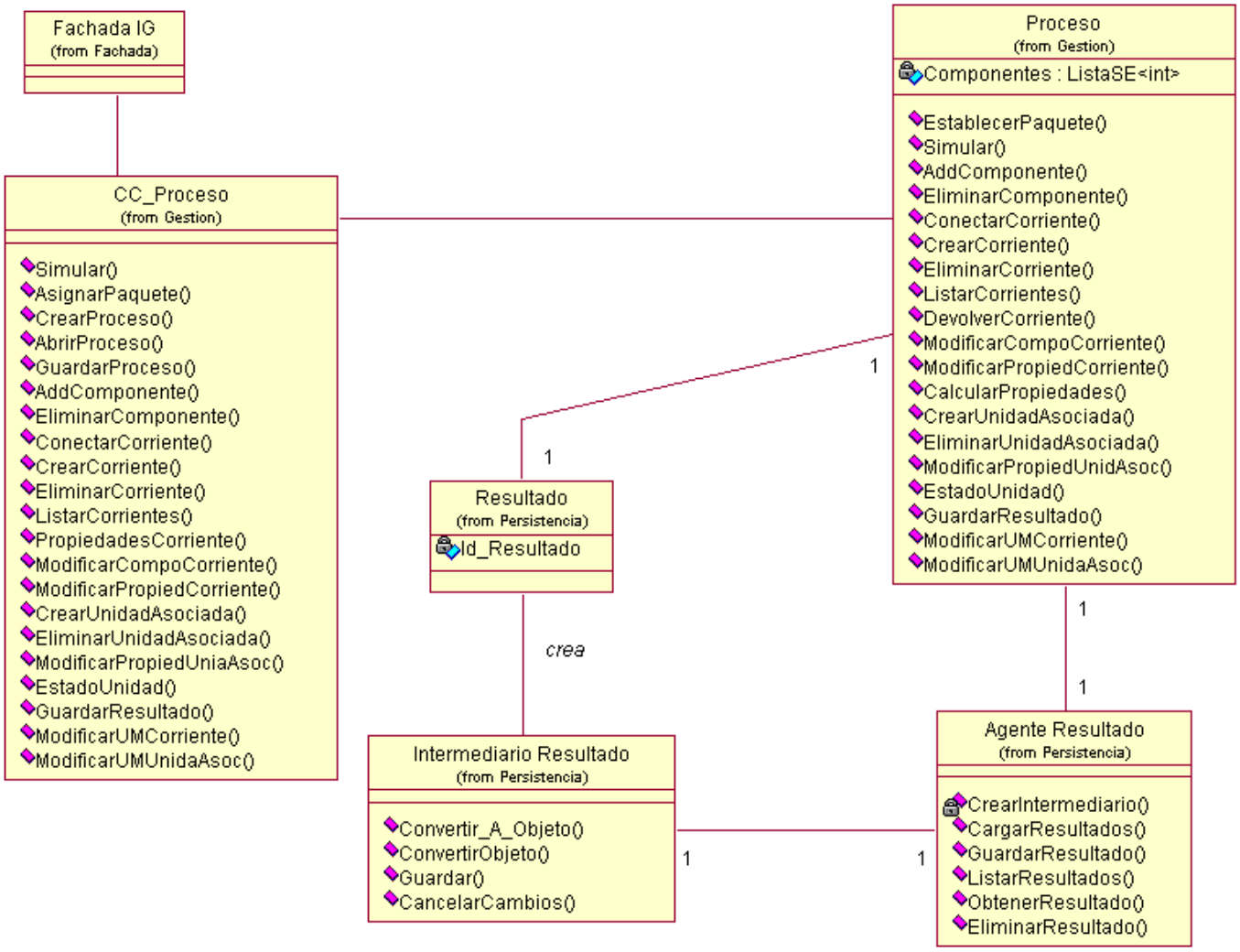

Ilustración 44: Diagrama de clases. Guardar Resultados

El resultado de una simulación, es por lo general el balance de materia y energía, es decir el valor de todas las variables asociadas a corrientes y parámetros intrínsecos de la planta como son los servicios en los intercambiadores, las columnas de destilación, entre otros.

Todo proceso de simulación complejo tiene una sección de salida de resultados. Al finalizar el proceso iterativo de acuerdo al criterio de convergencia definido por el usuario, se procederá a detener el proceso de simulación, retener los resultados, es decir almacenar en el lugar correspondiente todos los valores

convergidos de las corrientes del proceso, los valores y parámetros internos de los equipos, por ejemplo los perfiles internos de torres, composiciones, caudales, temperatura y presiones de cada etapa.

Existirá un sistema de almacenamiento de información (por ejemplo un Administrador de Base de Datos), en el cual pueden almacenarse los resultados de las diversas simulaciones para una misma planta, a los efectos de facilitar la presentación en forma gráfica de los resultados obtenidos, por comparación entre las distintas alternativas simuladas, si fuera necesario.

Para todo ello se tuvieron en cuenta las clases CC_Proceso y Proceso, para gestionar lo relacionado con el proceso de simulación y para la gestión de los resultados en el paquete de Persistencia se diseñaron las clases Agente Resultado, Intermediario Resultado y Resultado.

Descripción de las clases.

Nota: Para ver la descripción de las clases CC_Proceso y Proceso, dirigirse a la realización del caso de uso Gestionar Corriente.

Nombre: Agente Resultado	
Descripción: Su función principal es que es empleado para comunicar la clase controladora CC_Proceso con el paquete de Persistencia, para la gestión de resultados.	
Métodos	**Descripción**
CrearIntermediario()	Crea una instancia de una clase que se eencarga de materializar, desmaterializar y guardar un objeto en otro objeto caché (Clase intermediaria).
ListarResultados()	Devuelve un listado con los resultados de la simulación.
ObtenerResultado()	Busca un objeto y lo referencia, comprobando que ya esté materializado para no hacerlo de nuevo.
EliminarResultado()	Se le notifica al intermediario y se elimina el resultado.

Tabla 19: Descripción de la clase Agente Resultado

Nombre: Intermediario Resultado	
Descripción: Clase encargada de materializar, desmaterializar y guardar los resultados.	
Métodos	**Descripción**
Convertir_A_Objeto()	Convierte los datos a objetos y los materializa en cache.
Guardar()	Lleva un registro de los cambios que sufre un objeto, realizándose las actualizaciones necesarias.
CancelarCambios()	Elimina todos los cambios registrados.

Tabla 20: Descripción de la clase Intermediario Resultado

Nombre: Resultado	
Descripción: Objeto creado para almacenar los objetos materializados, los cuales dejan su identificador en esta clase como atributo.	
Atributos	**Tipo**
Id_Resultado	int

Tabla 21: Descripción de la clase Resultado

Diagrama de secuencia.

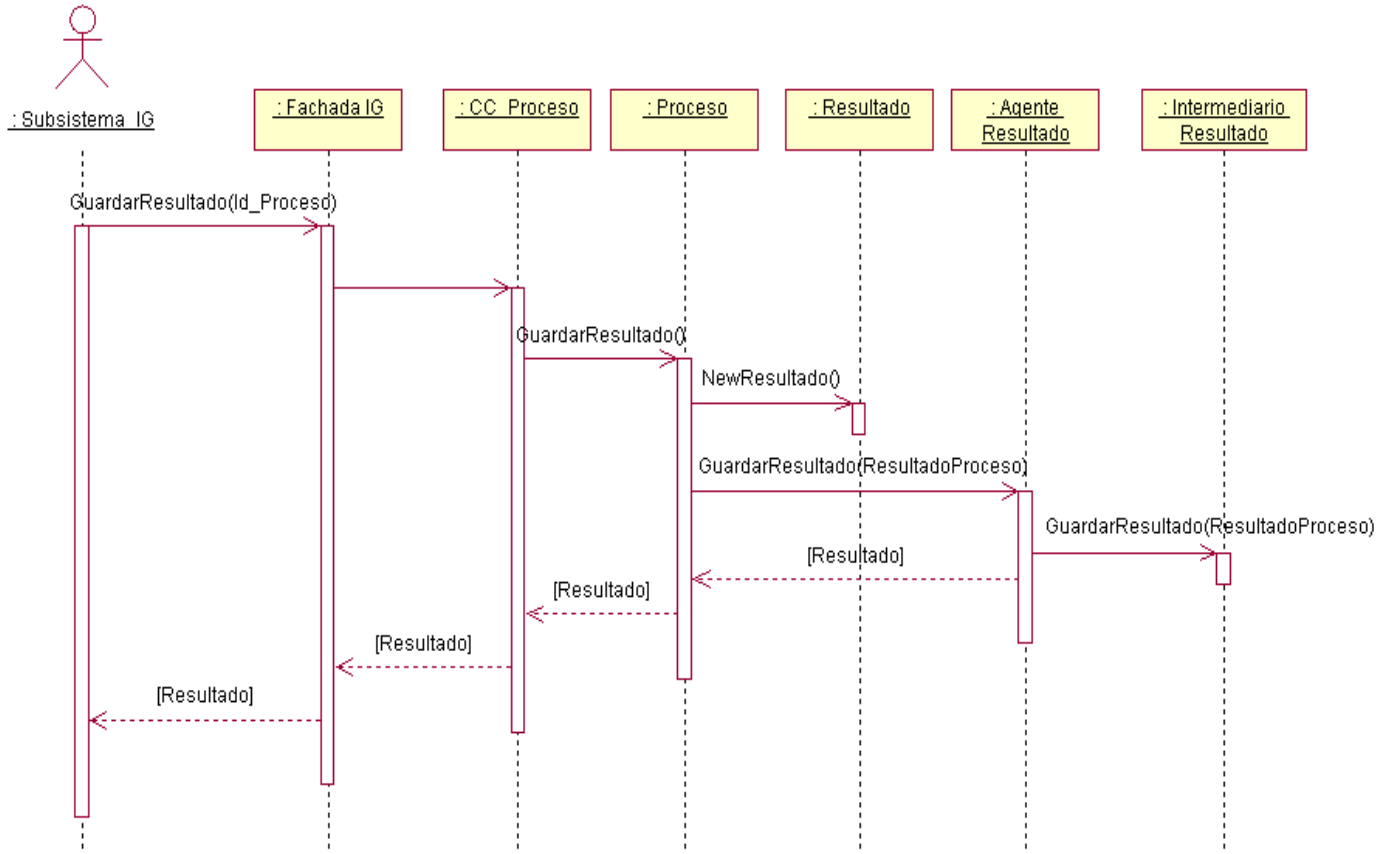

Ilustración 45: Diagrama de Secuencia. Guardar Resultado.

2.5.2.3.2.9. Realización del caso de uso Simular

La simulación de un proceso se inicia con el diagrama de flujo de materiales y energía (DFP); que representa de forma esquemática el flujo tecnológico del sistema de una fábrica. Este esquema está formado básicamente por equipos y corrientes que contienen materiales y energía. Simular incluye además conceptos tales como DFI, lo que no es más que la representación esquemática del flujo de la información en el cálculo de un equipo o sistema. Es por ello que la misión principal de todo simulador es representar una serie de cambios que ocurren en determinados procesos de una fábrica, donde está incluido el comportamiento y cálculo de los módulos o unidades de operación unitarias a los que entran las corrientes con determinados parámetros de información y se obtienen corrientes de salida con nuevos valores, para el análisis de resultado del proceso en sí.

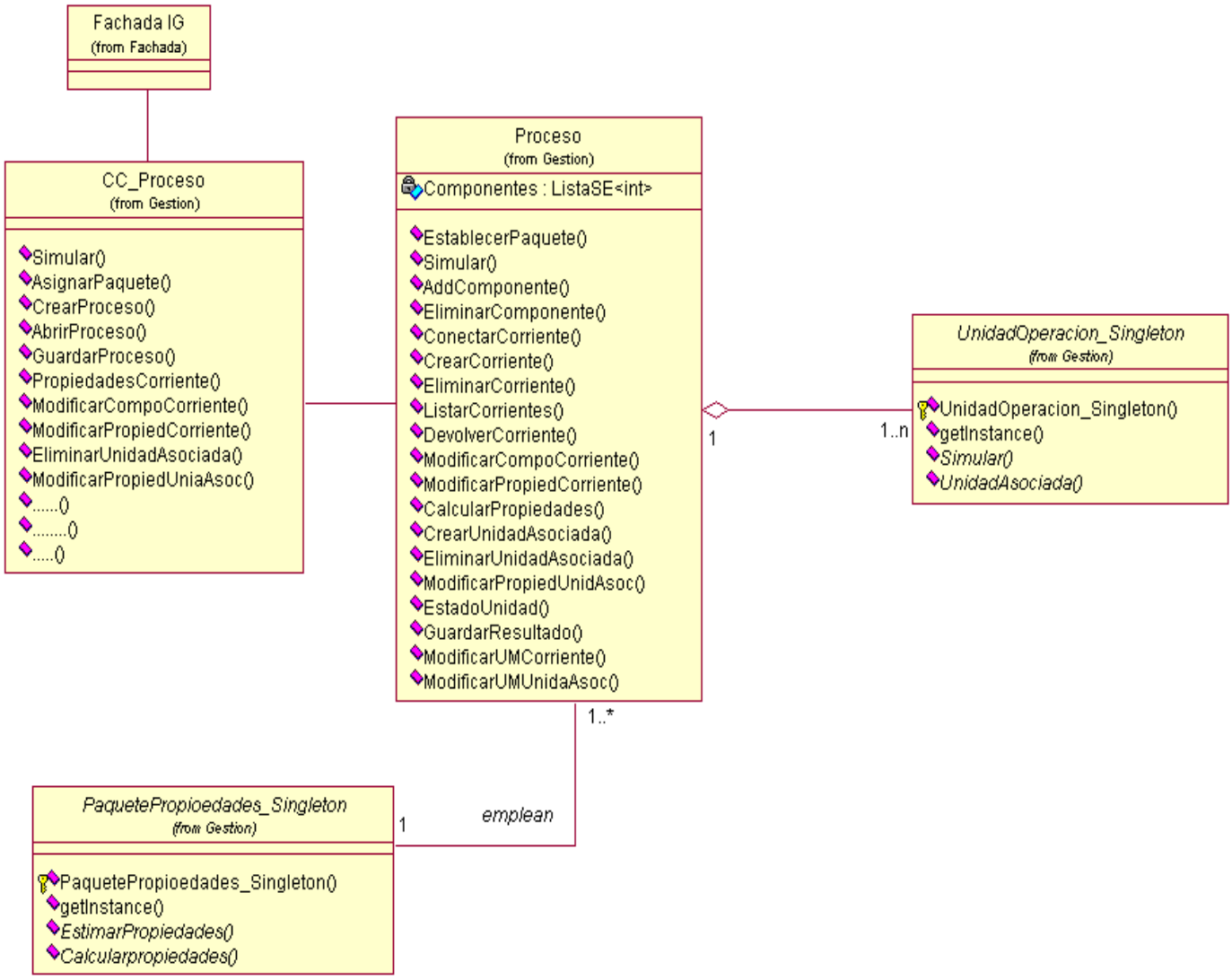

Ilustración 46: Diagrama de clases. Simular.

Descripción de las clases

Nota: Como se expresaba anteriormente la simulación incluye desde la selección del paquete de propiedades a emplear en un determinado proceso, hasta el cálculo de los equipos y las modificaciones a las propiedades de las corrientes. Debido a esto, para ver la descripción de las clases del diagrama anterior, vea la realización de los casos de uso Gestionar Corriente, Gestionar Unidad de Operación y Gestionar Paquete de Propiedades.

Diagrama de secuencia.

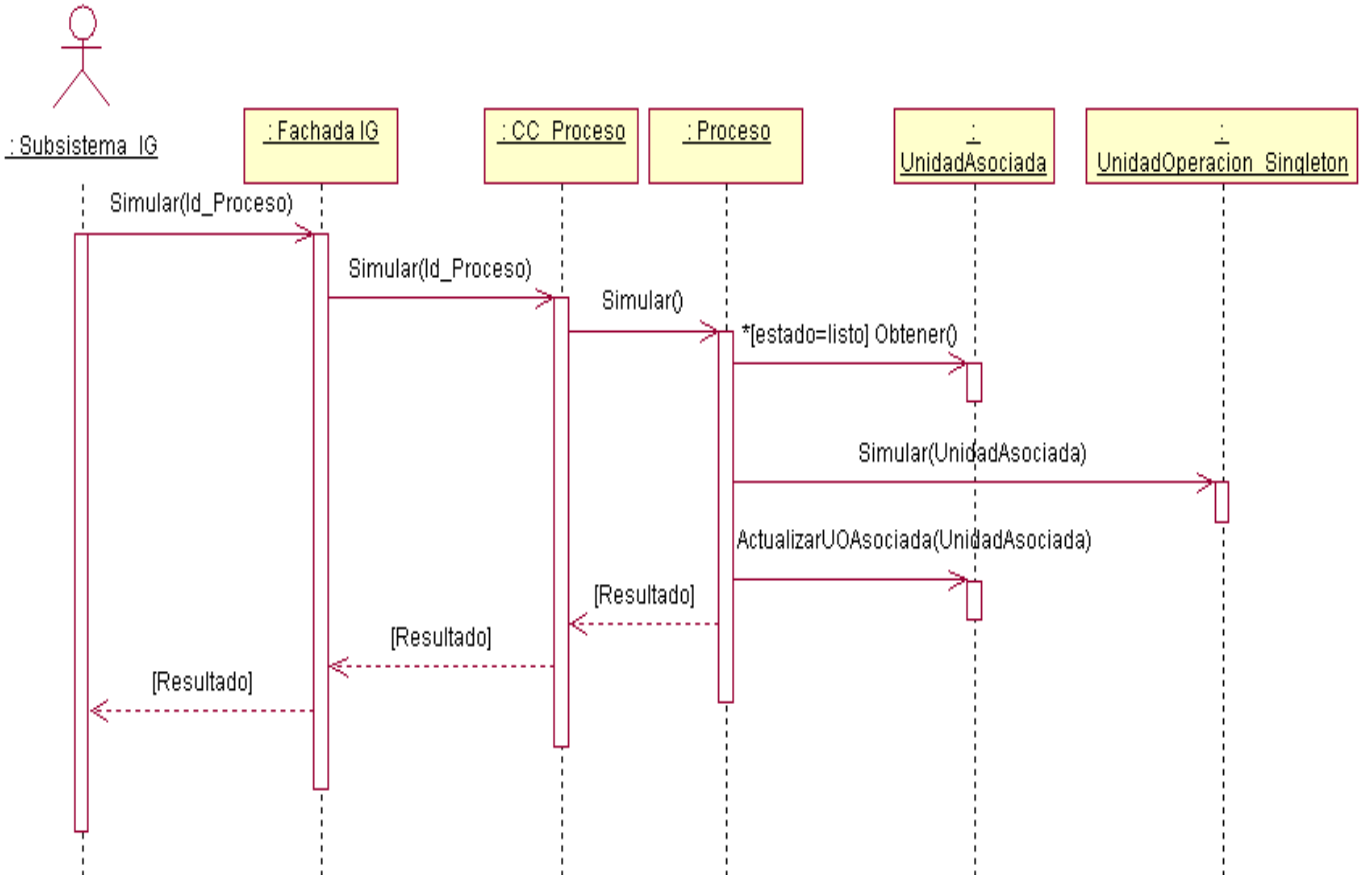

Ilustración 47: Diagrama de secuencia. Simular

2.5.2.3.2.10. Realización del casos de uso Gestionar Resultados

La gestión de resultados reviste gran importancia para realizar una comparación de los datos reales del proceso con los resultados de la simulación para un posterior análisis del mismo, lo cual puede permitir predecir las variaciones en el sistema frente a cambios operacionales. Posibilitando así la evaluación del rendimiento del sistema empleando un simulador de procesos determinado. En este caso, se ha brindado una fachada para que el subsistema de Análisis de Resultados interactúe con la lógica central del simulador. Para ello se han empleado en el diseño, las clases CC_Proceso, CC_Resultado, Proceso, Agente Resultado y Resultado.

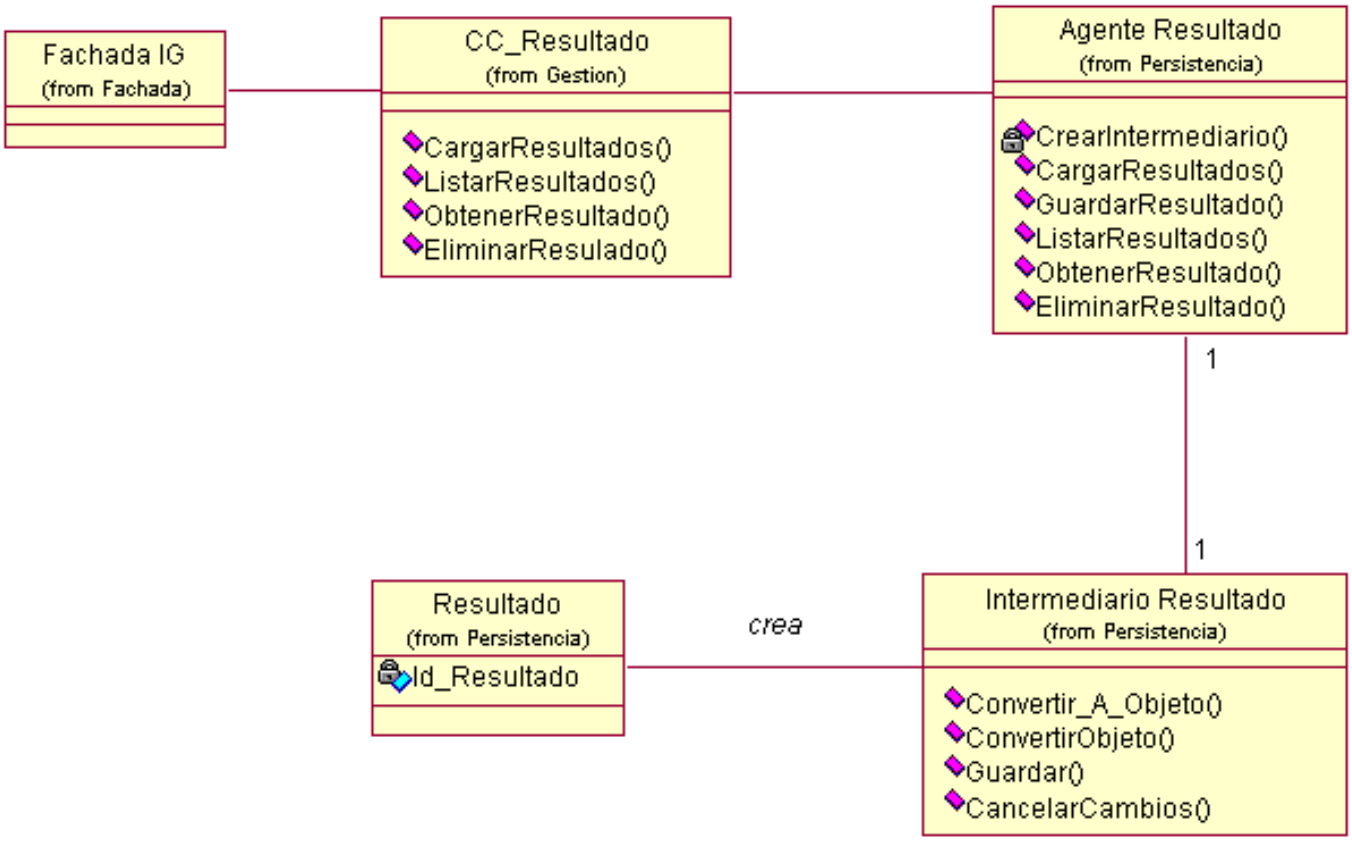

Ilustración 48: Diagrama de clases. Gestionar Resultado

Descripción de las clases

Nota: Para ver la descripción de las clases Agente Resultado, IntermediarioResultado y Resultado, dirigirse a la realización del caso de uso Guardar Resultado.

Nombre: CC_Resultado	
Descripción: Ha sido diseñada con el fin de gestionar todo lo referente a los resultados una vez que se han guardado.	
Métodos	**Descripción**
CargarResultado()	Busca el resultado en la dirección donde fue guardado y lo carga.
ListarResultados()	Devuelve un listado con los resultados de la simulación.
ObtenerRasultado()	Busca un objeto y lo referencia, comprobando que ya esté materializado para no hacerlo de nuevo.
EliminarResultado()	Se le notifica al intermediario y se elimina el resultado.

Tabla 22: Descripción de la clase CC_GestionarResultado

Diagramas de secuencia por escenarios. Gestionar Resultado.

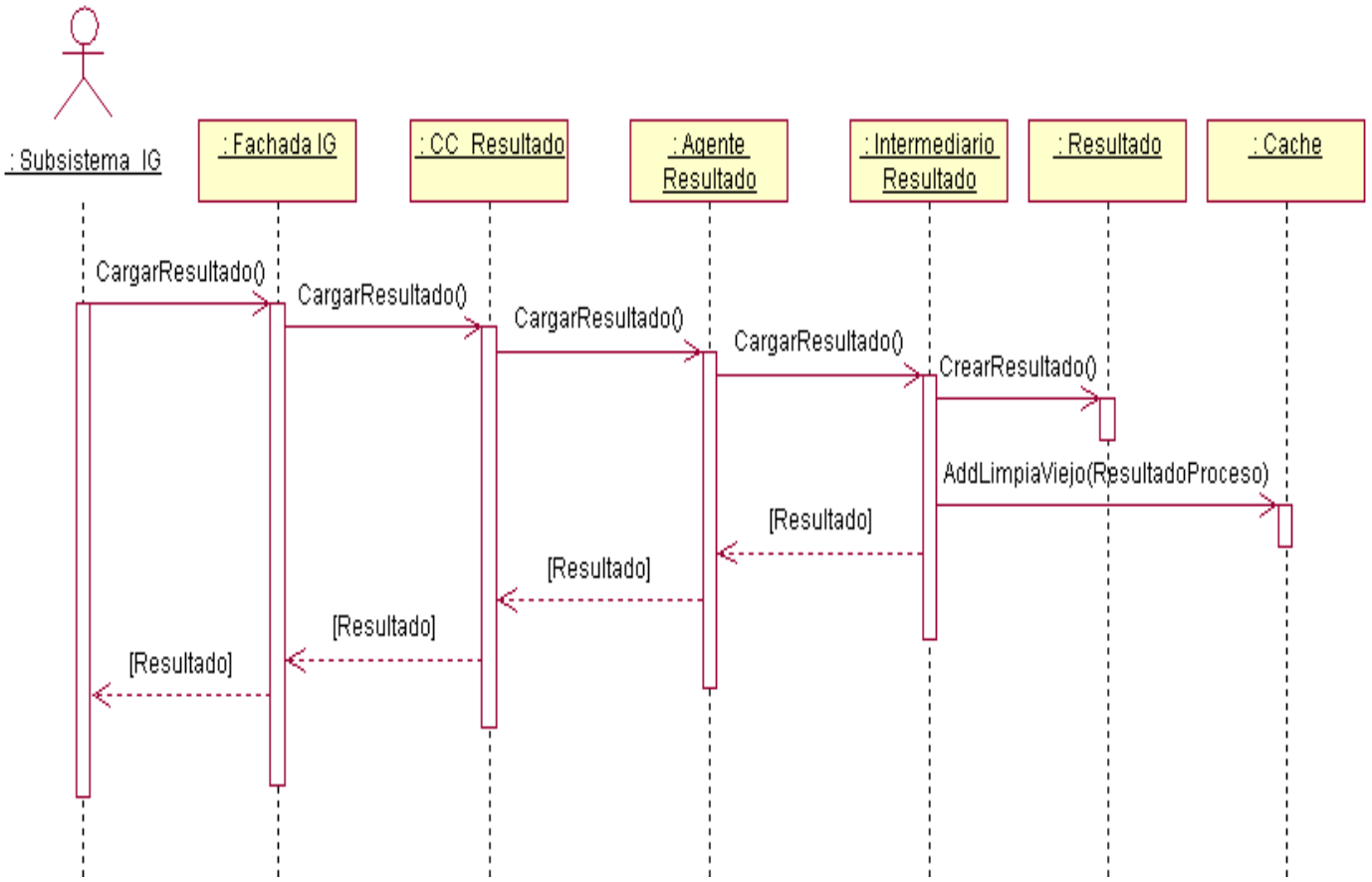

Ilustración 49: Diagrama de secuencia. Cargar Resultado

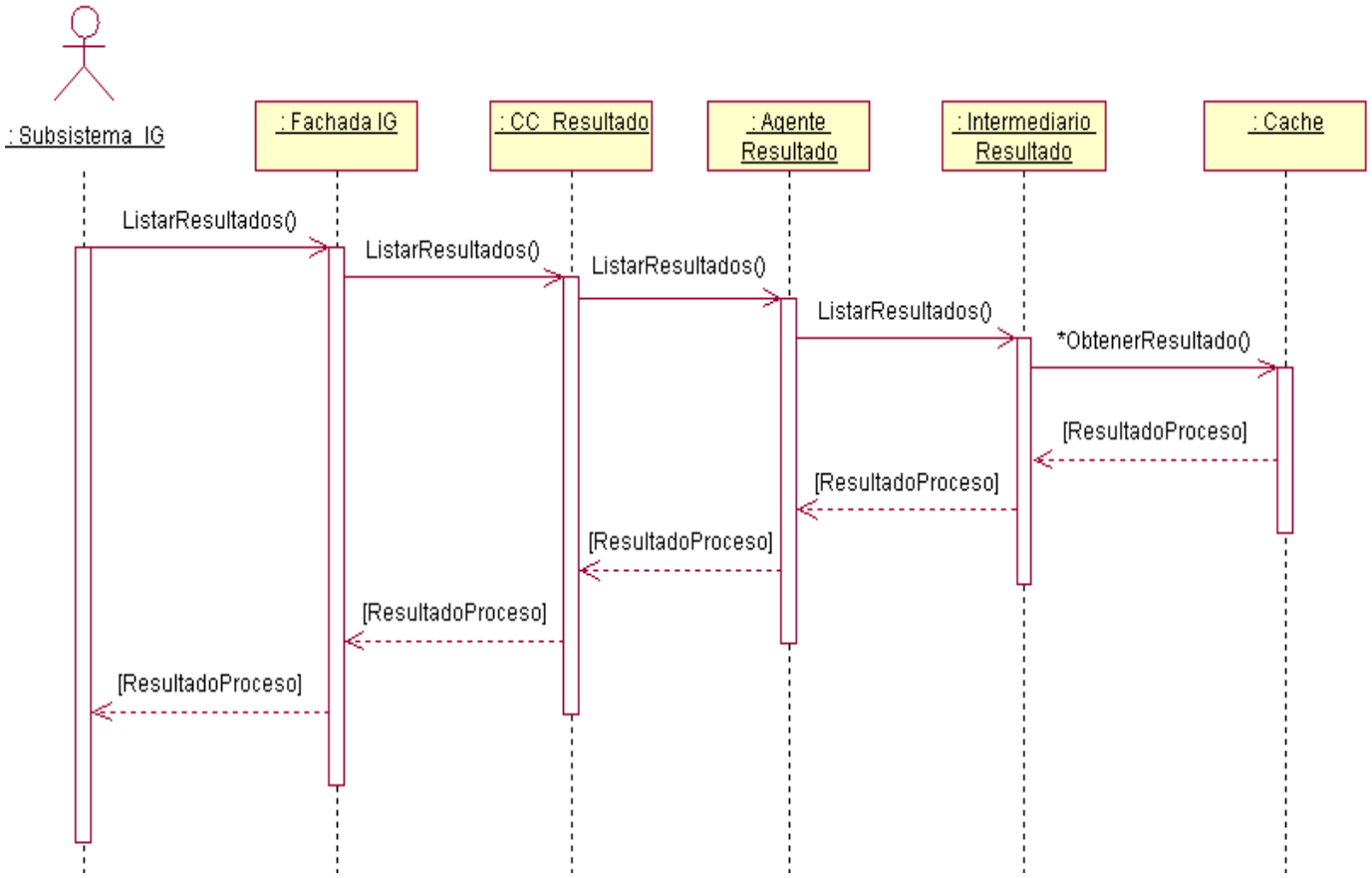

Ilustración 50: Diagrama de secuencia. Listar Resultados

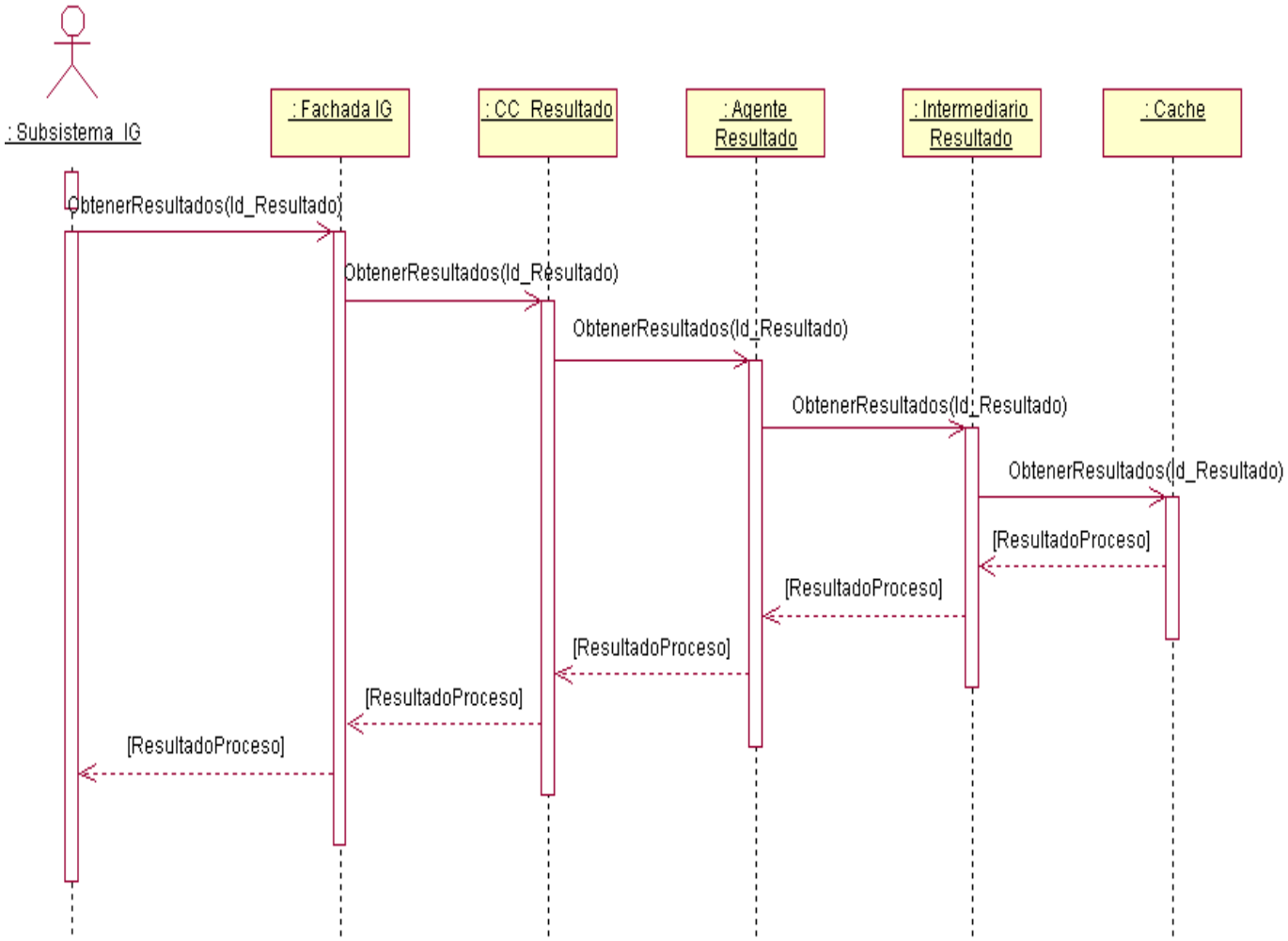

Ilustración 51: Diagrama de secuencia. Obtener Resultado

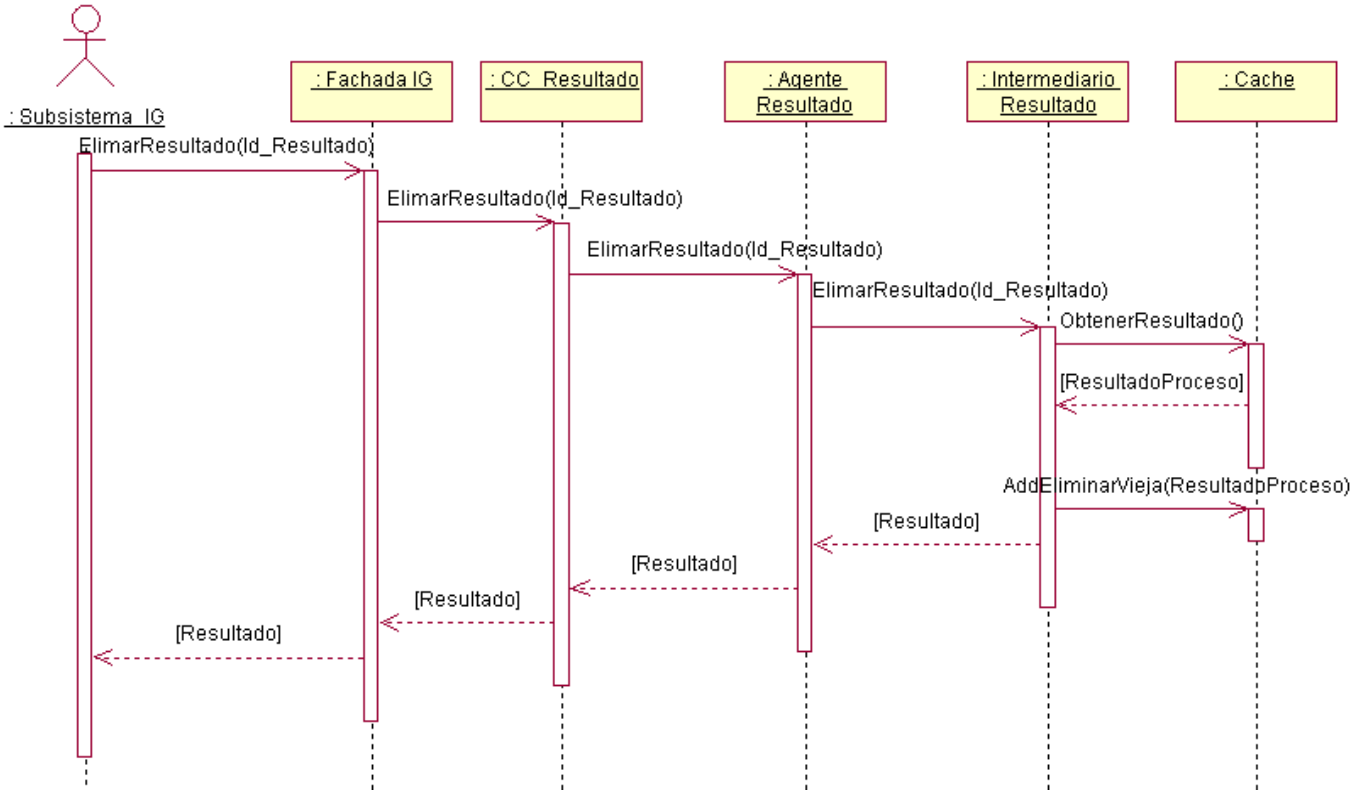

Ilustración 52: Diagrama de secuencia. Eliminar Resultado

2.5.2.3.3. Paquete de Persistencia

En este paquete se representó un esquema de persistencia muy necesario a la hora de almacenar y recuperar objetos. El mismo representa un conjunto cohesivo de clases que prestan servicios a la parte fundamental e invariable de un sistema lógico. Por ejemplo a la hora de crear las corrientes de un determinado proceso, estas van a estar conformadas por componentes los cuales van a estar materializados en un objeto caché. Así que este esquema permitirá leer un determinado fichero haciendo referencia a los objetos persistentes. Además se tiene la posibilidad de adicionar nuevas bibliotecas de componentes, donde para ello se emplearon patrones como el Identificador de Objetos (OID) y el Administrador de Caché, lo que permite que a la hora de guardar un objeto en memoria primero haya que buscarlo dado su identificador (OID) y si no está, se adiciona. También se contará con una clase que se encargue de materializar, desmaterializar y guardar un objeto en otro llamado caché (Clase intermediaria).

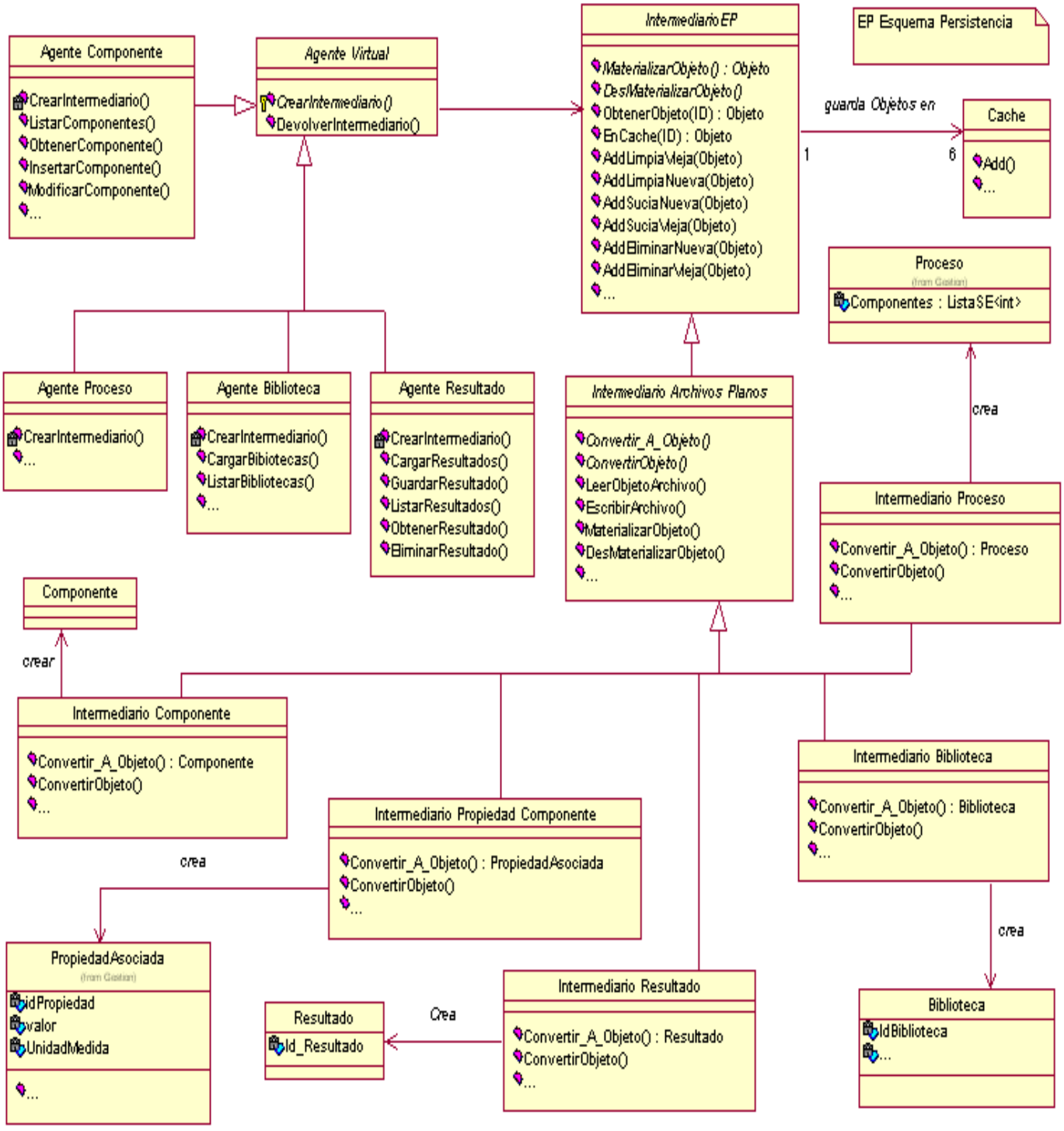

Ilustración 53: Esquema de persistencia

2.5.3. VISTA DE DESPLIEGUE

“Una vista de la arquitectura llamada vista de despliegue ilustra la distribución de procesos en un conjunto de nodos del sistema, incluida la distribución física de procesos y hebras. Esta vista es opcional y se emplea sólo si el sistema está distribuido entre más de un nodo.”(PROCESS, 2003)

Por ejemplo, en los casos en que existe un único servidor y muchos clientes, una vista de despliegue sólo es necesaria para delinear las responsabilidades del servidor y los clientes como una clase de nodos; no es necesario mostrar cada nodo cliente si todos tienen las mismas posibilidades.

En el caso particular del subsistema de modelos y propiedades, al cual se le quiere aplicar la arquitectura propuesta, puede decirse que no se considera un aspecto relevante mostrar la vista de despliegue, debido a que el subsistema antes mencionado conforma la lógica central de un simulador de procesos para la industria química, el cual constituye una aplicación de escritorio, por lo que todos los componentes corren sobre la misma computadora. Entonces sólo se tendrían en el diagrama de despliegue, elementos de procesamientos, o sea un nodo, y aunque no forma parte del subsistema de modelos y propiedades, también se podría contar con algún dispositivo para dar cobertura a los subsistemas de emisión de reportes y la interfaz gráfica.

Por lo que de forma general el diagrama a representar sólo lo conformarían una PC y una impresora, la cual se emplearía en la emisión de reportes o de las imágenes representativas de los diferentes diagramas de simulación.

Se coincide entonces con RUP en que el modelo de despliegue es opcional para sistemas con un único procesador, o sistemas simples con poca o sin ningún tipo de distribución del procesamiento. (Para una mejor comprensión ver en los Anexos el correspondiente a esta vista, o sea el Anexo 2)

2.5.4. VISTA DE IMPLEMENTACIÓN

"Esta vista es opcional**.** Se emplea sólo en los casos en que la implementación no esté dirigida estrictamente a partir del diseño, es decir, donde haya una distribución diferente de responsabilidades entre los paquetes correspondientes en los modelos de diseño e implementación. Si los paquetes de los modelos de diseño e implementación son idénticos, esta vista se puede omitir. "(PROCESS, 2003)

La vista de implementación suele capturar la enumeración de todos los subsistemas del modelo de implementación, los diagramas de componentes que ilustran la organización de los subsistemas en capas y jerarquías e ilustraciones de dependencias de importación entre subsistemas.

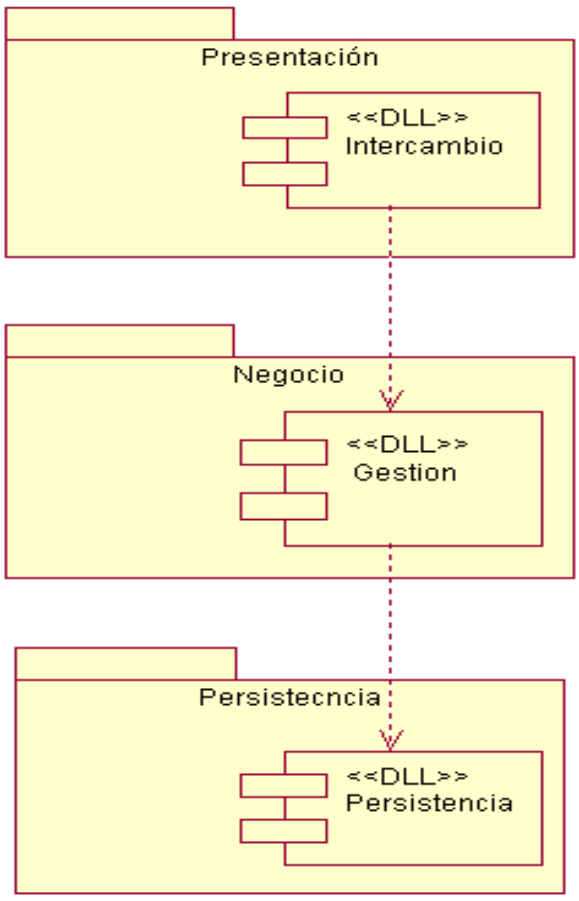

Ilustración 54: Diagrama de Componentes

En este caso existe una trazabilidad más que directa entre el diseño del subsistema y la implementación, donde los paquetes van a ser exactamente iguales, o sea, que en la implementación se tendría lo mismo que en el diseño pero en biblioteca de enlaces dinámicos (DLL), para facilitar una futura integración con otros subsistemas.

Por otra parte, la arquitectura a desarrollar se le quiere aplicar solamente al subsistema de modelos matemáticos y propiedades de un simulador de procesos para la industria química, o sea, para un paquete o componente del mismo. Además de que uno de los objetivos del flujo de trabajo de implementación es distribuir el sistema asignando componentes ejecutables a nodos en el diagrama de despliegue, por lo que al ser el simulador una aplicación de escritorio, todo estará centralizado en la misma unidad de procesamiento y el software estará situado en el hardware que lo contiene, debido a esto no habrá necesidad de distribuir el sistema. De ahí la simplicidad de esta vista para el caso en cuestión, como se mostró en la figura anterior.

2.6. Conclusiones del Capítulo

Con la realización del presente capítulo quedaron bien definidos los requerimientos del simulador, enmarcando las metas y restricciones arquitectónicas, lo que permitió identificar los elementos críticos o sensibles a tener en cuenta para desarrollar la arquitectura propuesta. La representación del sistema través de las 4+1 vistas definidas en la metodología seleccionada, posibilitó que se lograra tener una visión global del mismo. Todo esto hizo posible que se realizara un diseño de acorde a las principales funcionalidades del subsistema de modelos y propiedades, para proporcionar una mejor comprensión por parte de los desarrolladores, pensando siempre en una futura implementación.

CAPITULO III: EVALUACIÓN Y ANÁLISIS DE RESULTADOS

Al definir una arquitectura de software siempre surge la duda de cómo podemos estar seguros que la propuesta elegida es la correcta. Para ello, en el presente capítulo se pretende evaluar la arquitectura propuesta analizando los resultados obtenidos, de acuerdo a las técnicas y métodos de evaluación de arquitecturas de software.

Esta evaluación se inicia desde el momento en que quedan bien definidos los requisitos que debe cumplir el sistema para satisfacer las necesidades del usuario. La arquitectura debe estar preparada para desarrollar los casos de usos críticos y a medida que avance el proyecto permita incluir los restantes. Con ello se busca que la propuesta sea funcional y cumpla con los atributos de calidad definidos en los diferentes modelos de evaluación. Por lo que debe ser sometida a prueba, con el fin de que el sistema que la soporte sea robusto, modificable, seguro y con una performance aceptable.

3.1. ¿Por qué evaluar una arquitectura?

Muchos plantean que algún producto está listo cuando se encuentra libre de errores y cumple con todo lo requerido por el cliente, entonces mientras más rápido se localicen los problemas dentro de un proyecto de software, se estará en vías de lograr la calidad tempranamente. Además de que la evaluación de la arquitectura es la forma más económica de evitar desastres. Existen dos momentos para evaluar la arquitectura por decirlo de alguna manera, usted puede decidirse por la que más factible le sea, de acuerdo al proyecto que esté desarrollando:

Evaluación temprana: No es necesario que la arquitectura esté completamente especificada para efectuar la evaluación, y esto abarca desde las fases tempranas de diseño y a lo largo del desarrollo.

Evaluación tardía: Cuando ésta se encuentra establecida y la implementación se ha completado. Este es el caso general que se presenta al momento de la adquisición de un sistema ya desarrollado.

3.2. Cualidades para evaluar la arquitectura.

Según Bass existen dos categorías para la clasificación de los atributos de calidad(NUÑEZ, 2004):

Observables mediante la ejecución: aquellos atributos que se determinan del comportamiento del sistema en tiempo de ejecución

No observables en la ejecución: aquellos atributos que se establecen durante el desarrollo del sistema.

Al no estar implementado el subsistema de modelos matemáticos y propiedades, para la evaluación de su arquitectura sólo se pueden tratar los atributos de calidad no observables en vía de ejecución. Para esto se analizarán los planteados en el modelo de calidad ISO/IEC 9126.

3.2.1. Modelo de calidad ISO/IEC 9126

Diversos han sido los modelos de calidad a través de los años, los mismos marcan una pauta importante a la hora de determinar el nivel de complejidad de un determinado producto de software. Fueron establecidos de acuerdo al impacto de los atributos de calidad en lo que a evaluación de la arquitectura se refiere.

Variadas han sido las propuestas desde 1970 y durante los años siguientes hasta la actualidad, donde encontramos el modelo ISO/IEC 9126, en el cual estará centrado este capítulo en cuanto a modelos de evaluación se refiere, debido a que el mismo se basa en los atributos de la calidad que tienen un vínculo directo con la arquitectura.

Según lo planteado en la ISO 9126, la calidad de un software debe realizarse en base a sus atributos, tanto internos como externos. Es decir, la calidad interna, que trata la estructuración de las propiedades del software, influye directamente sobre la externa, o sea cualidades observables sin conocer como está construido el producto.

Estos atributos se expresan en un conjunto de características bien definidas en la norma ya mencionada:

- **Funcionalidad**: la capacidad del software de proveer las funciones que cumplen con las necesidades implícitas y explícitas cuando el mismo es utilizado bajo ciertas condiciones.

- **Fiabilidad**: la capacidad del software de mantener un nivel específico de rendimiento bajo determinadas condiciones de uso.
- **Usabilidad**: la capacidad del producto software de ser entendido, aprendido, usado y atractivo al usuario, cuando se usa bajo ciertas condiciones.
- **Eficiencia**: la capacidad del software de ofrecer el rendimiento apropiado con respecto a la cantidad de recursos utilizados, bajo condiciones prefijadas.
- **Mantenibilidad**: la capacidad del producto de ser modificado. Dichas modificaciones pueden incluir correcciones, mejoras o adaptaciones a cambios en el entorno y en los requisitos y especificaciones funcionales.
- **Portabilidad**: la capacidad del software de ser trasladado de un entorno (informático) a otro.

3.3. Técnicas de Evaluación

En la actualidad existen diversas técnicas que permiten realizar evaluaciones a la arquitectura, las mismas se clasifican en cualitativas y cuantitativas.

En las técnicas de evaluación cualitativas se pueden utilizar escenarios, cuestionarios o listas de verificación. Mientras que en las cuantitativas se emplean métricas, simulaciones, prototipos, experimentos o modelos matemáticos.

En varias ocasiones, debido al costo que implica realizar técnicas cuantitativas, los arquitectos de software optan por las cualitativas, decidiendo así entre situaciones de diseño. En el caso del subsistema tratado, se escogió como técnica, la evaluación basada en escenarios debido a que representa una utilización anticipada o deseada del sistema y constituyen una abstracción de los principales requerimientos del sistema. Por lo que los escenarios serán los medios descriptivos para especificar y evaluar atributos de calidad.

Se puede definir un escenario como una breve descripción de la interacción entre un interesado y el sistema. El mismo consta de tres partes: estímulo, ambiente y respuesta. El primero describe la acción a realizar por el involucrado en el desarrollo para iniciar el intercambio con el sistema. El ambiente representa lo que sucede en el sistema en esa ocasión y la respuesta da la medida de cómo debería responder el sistema ante el estímulo mediante la arquitectura. Permitiendo con esto validar enunciados sobre la calidad del sistema a través de una descripción arquitectónica.

3.4. Métodos de evaluación

El estudio realizado en cuanto a métodos de evaluación de arquitecturas de software se refiere ha arrojado los siguientes resultados:

1. En el Método de Análisis de la Arquitectura de Software (SAAM) la generación de escenarios está basada en la visión de los interesados, no provee una métrica clara sobre la calidad de la arquitectura evaluada y el equipo de evaluación confía en la experiencia de los arquitectos para proponer arquitecturas candidatas.
2. El ATAM, (Arquitectura de Acuerdo del Método de Análisis) obtiene su nombre no solo porque nos dice cuán bien una arquitectura particular satisface las metas de calidad, sino que también provee ideas de cómo esas metas de calidad interactúan entre ellas, como realizan concesiones mutuas (tradeoff). Fue diseñado con el fin de obtener las propuestas arquitectónicas utilizadas para alcanzar las metas de los atributos de calidad. Exponiendo los riesgos arquitectónicos que potencialmente prohíben a una organización conquistar las metas del negocio.
3. ADR (Revisión Activa de Diseño). “ADR es 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”. (NUÑEZ, 2004)
4. ARID (Revisión Activa de Diseño Intermedio). ARID es un método de bajo costo y gran beneficio, el mismo es conveniente para realizar la evaluación de diseños parciales en las etapas tempranas del desarrollo. ARID se considera un híbrido entre ADR y ATAM. Se basa en ensamblar el diseño de los stakeholders para articular los escenarios de usos importantes o significativos, y probar el diseño para ver si satisface los escenarios. Como resultado de la aplicación de dicho procedimiento se obtiene un diseño de alta fidelidad acompañado de una alta familiarización con el diseño de los stakeholders. Este método consta de nueve pasos agrupados en dos fases (Actividades Previas y Evaluación).
5. Losavio (2003): “Es un método para evaluar y comparar arquitecturas de software candidatas, que hace uso del modelo de especificación de atributos de calidad adaptado del modelo ISO/IEC 9126. La especificación de los atributos de calidad haciendo uso de un modelo basado en estándares

internacionales ofrece una vista amplia y global de los atributos de calidad, tanto a usuarios como arquitectos del sistema, para efectos de la evaluación "(NUÑEZ, 2004).

6. El Método de Evaluación para Arquitecturas de Software Basadas en Componentes (MECABIC), adapta diferentes elementos de algunos métodos de evaluación arquitectónica (ATAM, BOSCH, ARD, ARID, Losavio) y establece un conjunto de pasos para determinar la calidad de sistemas de software basados en componentes.

3.4.1. Método Propuesto

Visto el análisis realizado, el método seleccionado fue el MECABIC. Esta elección viene dada por las características del mismo, ya que con esto se propone un procedimiento para evaluar Arquitecturas de Software Basadas en Componentes (ASBC) teniendo como objetivo en este caso, de que sea aplicado en la arquitectura en capas, donde se integran los elementos más importantes de otros métodos. Empleando la técnica de evaluación basada en escenarios y utilizando como herramienta el árbol de utilidad.

Generación del árbol de utilidad

El MECABIC proporciona un árbol de utilidad inicial especifico a partir del cual seleccionan un conjunto de escenarios de interés, el mismo está basado en el modelo de calidad ISO 9126 para arquitecturas de software propuesto por Losavio. Se asume que al aplicar el método, las funcionalidades presentes en el sistema son las que necesitan los usuarios.

Característica	Sub-característica	Escenario
Funcionabilidad	Interoperabilidad	El sistema posibilita salvar los resultados en ficheros que pueden ser leídos por otros sistemas. Además de que el componente de acceso a datos puede modificarse, para aplicaciones que trabajen con bases de datos relacionales.(A,M)
	Precisión	Los resultados calculados tienen el tratamiento adecuado, ya que el sistema cuenta con un paquete en la capa de negocio

		para la manipulación de unidades de medida.(A,M)
	Seguridad	No existen restricciones de seguridad ya que no se maneja una base de datos relacional donde haya que proteger información. El único responsable de interactuar con la aplicación es el ingeniero químico y este usuario no opera directamente sobre el subsistema de modelos y propiedades, sino sobre la interfaz gráfica, la cual invoca las funciones que necesita.(M,B)
	Adecuación	El subsistema presenta los componentes de Gestión y Persistencia dentro de las capas de negocio y acceso a datos respectivamente. Los mismos proveen todo lo necesario para la simulación de procesos químicos en simuladores con características semejantes al que contiene la arquitectura que soporta el subsistema de modelos y propiedades.(A, M)
Mantenibilidad	Habilidad de cambio, estabilidad, prueba	El subsistema facilita la sustitución o adaptación de los módulos ya que la propia estructura en capas y el tipo secuencial modular del simulador lo permiten. Además de hacer extensible el subsistema posibilitando agregar nuevas bibliotecas personalizadas y adicionar módulos. Es posible verificar el estado de los módulos, si están listos para simularse o no.(A,M)
Portabilidad	Adaptabilidad	Con la estructuración en capas del sistema se posibilita migrar a Linux, realizando las modificaciones necesarias en el paquete Intercambio y dejando intacta el resto de la aplicación.(M,M)
	Capacidad de Instalación	La tecnología seleccionada fue el framework.Net, de ahí que se puede desarrollar aplicaciones e instalar el software en cualquier entorno que soporte el framework. (M,B)

Usabilidad	Dominio de Formación	El subsistema ofrece la posibilidad de personalizar bibliotecas y componentes, al dar la opción de formar nuevos mezclas y bibliotecas, operaciones que están contenidas dentro del paquete de Gestión y que le dan un uso más general al subsistema y al simulador, gracias a la gestión de estas funcionalidades. (M, B). Los subsistemas de Interfaz Gráfica y Análisis de Resultados en un contexto particular piden ayuda, y el sistema da ayuda para ese contexto. (A, M)
	Operaciones Normales	El subsistema presenta lo necesario para realizar la simulación de procesos dentro de lo normal y mantener un intercambio con los demás subsistemas.(A, M)
Eficiencia	Recursos Empleados	El subsistema no emplea procesadores demasiado potentes, ni con una capacidad de almacenamiento destacada como servidores, ni requiere de otro recurso de red.(M, B)

Tabla 23: Representación del árbol de utilidad propuesto por el método

Una vez definido el árbol de utilidad se priorizan los escenarios como un par (X, Y) en el cual X define el esfuerzo de satisfacer el escenario, y la Y indica los riesgos que se corren al excluirlos del árbol de utilidad. Los posibles valores para X e Y son A (Alto), B (Bajo) y M (Medio).

Para analizar los enfoques de arquitectura este método se basa en una serie de preguntas que arrojaron como resultado que los escenarios de alta prioridad están comprendidos en la Mantenibilidad, Funcionalidad y Usabilidad.

Sobre el alcance y repercusión de los cambios en los escenarios de importancia alta, se concluyó lo siguiente:

En cuanto a **funcionalidad** se refiere, el subsistema presenta un esquema de persistencia para archivos planos, el mismo posibilita realizar las modificaciones necesarias en el componente de acceso a datos para si se opta por una base de datos relacional, realizar el mapeo de las clases y la representación de las tablas. Este aspecto se registra como un punto de sensibilidad que afecta positivamente a la funcionalidad.

Los resultados calculados son otro escenario a analizar debido a la gran cantidad de operaciones que se llevan a cabo en la simulación de un proceso, para ello se diseñaron las clases necesarias para dar cobertura a esto y tener en cuenta la manipulación de unidades de medida. Sin embargo a la hora de convertir una determinada unidad de medida o calcular propiedades, pueden surgir algunos fallos que darían al traste con la convergencia y no se podría realizar la simulación, de ahí que el tratamiento de errores que no ha sido considerado hasta el momento debería ser visto como un punto de sensibilidad que afecta negativamente a la exactitud en la funcionalidad del sistema.
La adecuación del subsistema se encuentra bien representada en la vista lógica de la arquitectura, donde se estructuró el software en capas que contienen los paquetes con las clases necesarias para la simulación de un proceso químico. Por lo que se puede apreciar que el subsistema podría servir de lógica central a cualquier simulador con características semejantes al actual.

La **mantenibilidad** es otro atributo que presenta un escenario de alta prioridad, ya que al estar articulado el software en capas esto posibilita la modularidad, permitiendo realizar cambios sobre las unidades de operación o módulos. El tratamiento de errores se realizaría de una manera más fácil, ya que el simulador es del tipo modular secuencial, y estos cambios en un módulo no afectarían a todo el sistema ni a la arquitectura en general. Además de que con la posibilidad de formar nuevas mezclas y agregar bibliotecas personalizadas, se hace más extensible el sistema y le da un uso más general al simulador. Esto contribuye a ser un punto de sensibilidad que afecta positivamente a la mantenibilidad.

En reiteradas ocasiones se ha concebido a la **usabilidad** como una propiedad inherente a la capa de presentación. Se piensa que resolviendo los problemas en dicha capa, al estar separada de la lógica de la aplicación, no quedaría afectada la funcionalidad del software. En este sentido existen aspectos de mayor envergadura que no son tratados en este nivel, pues el subsistema de interfaz gráfica necesita de las funcionalidades contenidas en el paquete de Gestión, con el fin de mostrar la información relevante para el usuario y que está disponible en el subsistema de modelos y propiedades. Lo anterior afecta positivamente al atributo de calidad tratado en cuanto a su impacto en la arquitectura.

Como resultado de este análisis se registraron cuatro puntos de sensibilidad y un punto de desventaja. Identificándose como atributos de alta prioridad y que no constituyen riesgos la mantenibilidad, usabilidad y funcionalidad, aunque en este último se encuentra un punto no favorable en cuanto a la exactitud, pero

no hace que el atributo en general afecte negativamente a la arquitectura, ya que este problema se resolvería con la inclusión de un paquete de excepciones para el tratamiento de errores. Finalmente fueron clasificados como importantes pero con un grado de prioridad más bajo los atributos de calidad, eficiencia y portabilidad.

3.5. Comparando arquitecturas

En este epígrafe se han tenido en cuenta dos casos de estudio, donde fue seleccionado el mismo estilo arquitectónico que el propuesto para el subsistema de modelos y propiedades. El primer ejemplo es un video juego o simulador, mientras que el segundo constituye una aplicación de entrenamiento para la operación de subestaciones eléctricas de transmisión.

Video Juego o simulador

Se tiene un primer ejemplo de tipo video juego o simulador donde se pretende que un grupo de alumnos desarrollen un juego basado en la batalla entre hombres y dragones. Los hombres lanzan cuchillos y los dragones bolas de fuego. Los dragones se mueven en el área izquierda de la pantalla y los hombres en el lado derecho. En mitad de la batalla aparecen paredes móviles que se desplazan en el centro de la pantalla. El número de luchadores puede ser variable y dinámico. El planteamiento de solución resume que se estructuren las principales características de la aplicación en niveles, donde todos los componentes correrían sobre la misma PC, y el usuario interactuaría con una interfaz a través de varios dispositivos, dígase monitor y teclado. Además de manejarse un nivel para efectuar toda la programación y cálculos que implicaría simular la batalla, teniendo en cuenta otro nivel para la persistencia y el manejo de ficheros a la hora salvar y cargar partidas.

Según la situación planteada los alumnos representaron el modelo de domino y el diagrama de clases para la aplicación.

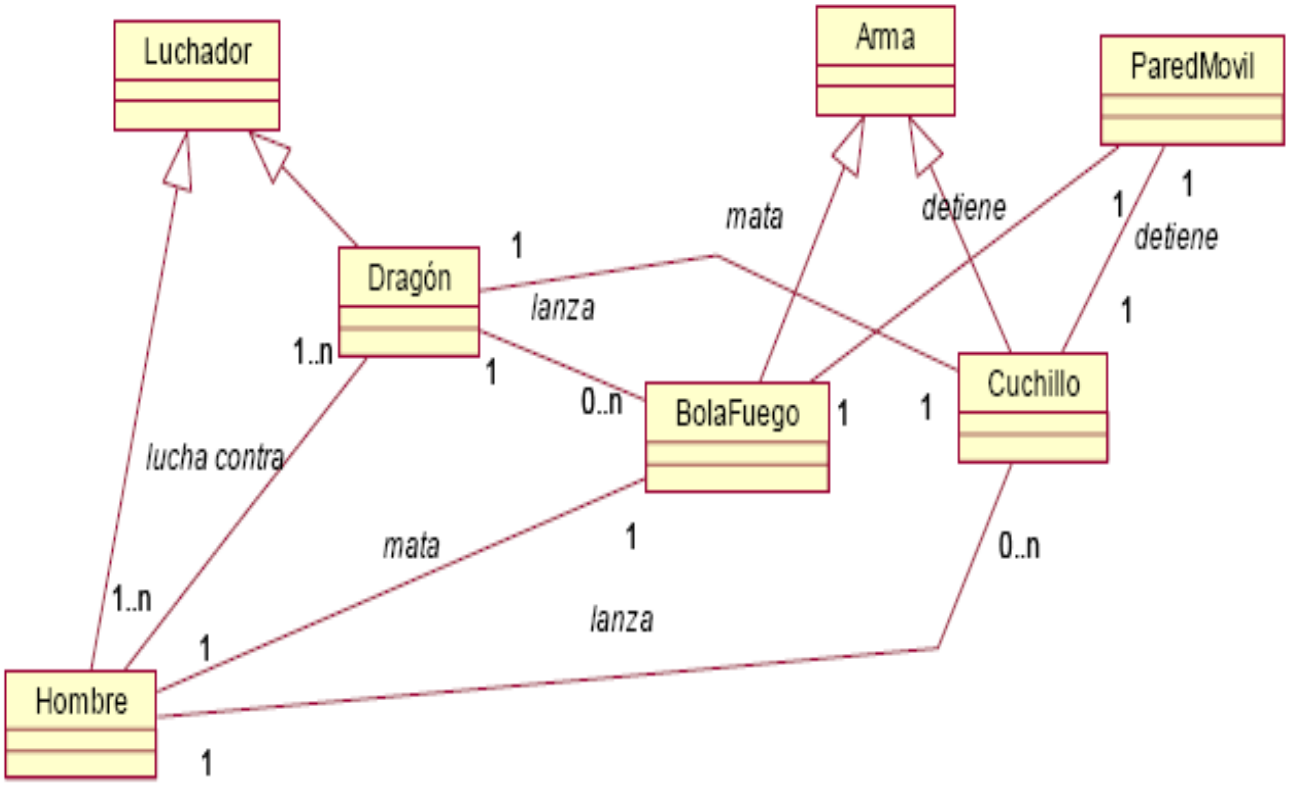

Ilustración 55: Modelo de Dominio. Caso de estudio Video Juego

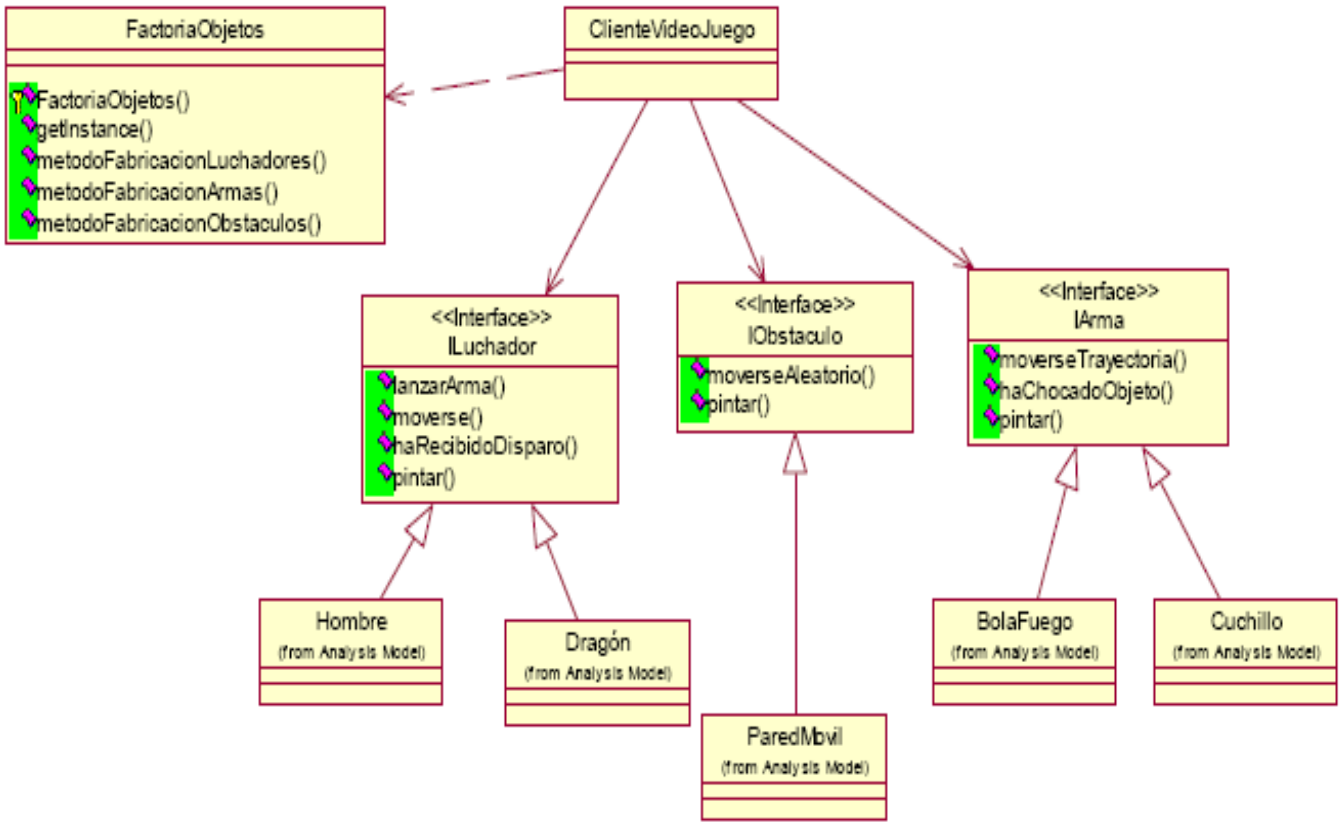

Ilustración 56: Diagrama de clases. Caso de Estudio Video Juego

Como se pudo apreciar en el diagrama de clases, este ejercicio es un buen ejemplo de la aplicación de patrones Singleton (GoF), Factoria Abstracta (GoF) y Métodos de Fabricación (GoF), los cuales incluyen las buenas prácticas de Polimorfismo (GRASP) y Fabricación Pura (GRASP), muchos de los cuales fueron empleados en la vista lógica de la propuesta de solución. Este caso de estudio al tener en cuenta la simulación, los cálculos para la ubicación de los caracteres en pantalla y el trabajo en la persistencia, hacen que sea un buen ejemplo para el subsistema de modelos matemáticos y propiedades. Además de que todos los componentes corren sobre el mismo ordenador, algo similar a lo que ocurre en el subsistema al que se le quiere aplicar la arquitectura propuesta.

Simulador de entrenamiento para la operación de subestaciones eléctricas de transmisión

La operación de subestaciones es un campo potencial para el desarrollo de sistemas enfocados a la optimización de sus procesos. Esto motivó la elaboración de un sistema de entrenamiento para el personal que atiende las subestaciones. Este sistema se fundamenta en tecnologías de la información y las comunicaciones, que permitan el desarrollo de nuevas alternativas para los procesos de formación, apoyadas en técnicas de simulación e inteligencia artificial, aportando de esta manera mecanismos de entrenamiento que estimulen el auto aprendizaje, la adquisición de habilidades y destrezas en el personal. La arquitectura candidata seleccionada para la implantación del sistema fue la de tres capas, ya que dada las características de la aplicación se define una capa de cliente, una de dominio de la aplicación y otra capa de datos.

En la ***capa de presentación*** se reúnen todos los aspectos del software que tienen que ver con las interfaces y la interacción con los diferentes tipos de usuarios. Los datos se solicitan de aquí inicialmente, donde las interfaces de usuario están representadas por navegadores web o clientes como aplicaciones java. Incluyendo el manejo y aspecto de ventanas, formato de los reportes, menús, gráficos y demás elementos visuales.

En el ***domino de la aplicación*** se automatizan todos los procesos de negocio realizados por el usuario, abarcando las tareas que forman parte de estos procesos y las reglas y restricciones que se aplican.

Para la ***manipulación de los datos*** se empleó una base de datos relacional. La aplicación se apoya además en las ventajas que ofrece java, lo que brinda la portabilidad, asegurando una ejecución igual en cualquier plataforma que cuente con la máquina virtual de este lenguaje.

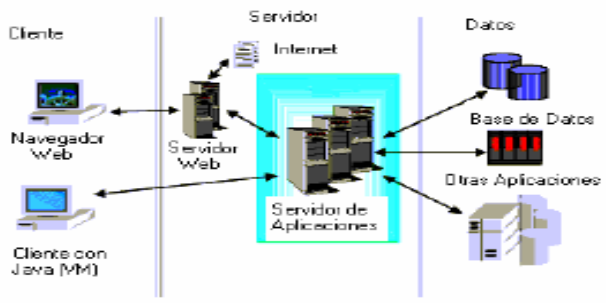

Ilustración 57: Arquitectura en capas. Caso de estudio Simulador de entrenamiento

El entorno de trabajo de la aplicación cuenta sobre todo con componentes visuales, con un menú de usuarios y otro de opciones. El sistema de simulación cuenta con dos módulos: Constructor y Simulador, que de igual forma son operados en su mayoría visualmente.

Como se puede apreciar este caso de estudio se adecua más bien al subsistema de interfaz gráfica y no al de modelos matemáticos y propiedades, debido a que la mayoría de las funcionalidades que brinda son para el trabajo con la presentación al usuario. Por otra parte esta arquitectura pudiera emplearse no en un subsistema del simulador, sino en el simulador como tal, y sobre todo para que sea más potente, presente mayor número de bibliotecas de componentes y gran diversidad de paquetes de propiedades, ya que emplea una base de datos relacional, a lo que se suma las ventajas que ofrece java como lenguaje.

3.6. Conclusiones

Debido a que la arquitectura del subsistema no se encuentra implantada, en este capítulo se realizó una evaluación de la solución propuesta para resolver el problema científico de esta investigación, mostrando el comportamiento de los atributos de calidad atendiendo a las vistas arquitectónicas desarrolladas. Para ello se tuvo en cuenta que el método seleccionado, se adaptara al estilo arquitectónico empleado y a las características del subsistema de modelos y propiedades. Finalmente se realizó la caracterización de dos casos de estudio, donde el elemento principal fue la adecuación de la arquitectura al subsistema estudiado, constatando la identificación del primer ejemplo con el aspecto antes expuesto.

CONCLUSIONES GENERALES

La arquitectura propuesta cumple con los requerimientos funcionales y no funcionales definidos para la aplicación y la descripción de la misma mediante las vistas definidas por RUP garantizaron que se tuviera una visión de todos los modelos del sistema y de aquellos elementos arquitectónicamente significativos durante el Proceso de Desarrollo, aunque solo fue necesario la descripción de la vista de casos de uso, la lógica, y brevemente la de implementación, debido a las características del sistema.

El estilo arquitectónico en Capas permitió que el sistema se estructurara de forma tal que permitiera la descomposición de la complejidad estructural y que el mantenimiento no sea complejo.

El análisis cualitativo permitió establecer una estrategia para lograr la calidad requerida de la arquitectura.

Con la investigación realizada se logró además profundizar en los conocimientos sobre arquitectura de software y todo lo relacionado con la misma. Desde los diferentes estilos y patrones arquitectónicos que existen, hasta los conceptos de diseño y las técnicas y métodos para la evaluación de arquitecturas de software.

RECOMENDACIONES

Luego de la realización del presente trabajo se recomienda:

- Implementar la arquitectura propuesta.
- Desarrollar el código en C#, realizando pruebas sobre la plataforma Mono, para la ejecución de la aplicación en Linux.
- Realizar la integración del subsistema de modelos y propiedades con lo demás subsistemas del simulador.
- Refinar la arquitectura a medida que se implemente el paquete de modelos y propiedades del simulador, siempre que sea necesario y esté debidamente justificado.

CITAS REALIZADAS

- C. Alexander, S. I, M. SILVERSTEIN, M. JACOBSON, I. FIKSDAHL-KING. *A Pattern Language.* Oxford University Press, 1977.
- ANÓNIMO. *Modelos iterativos e incrementales. Introducción Ingeniería de software.* Disponible en: http://jpozo334.blogspot.es/1193700420/modelos-incrementales-e-iterativos/
- ARIAS, G. A. B. G. E. N. P. J. *Arquitectura de Proyectos de IT. Evaluación de Arquitecturas.* Facultad Regional de Buenos Aires - Departamento de Sistemas. Universidad Tecnológica Nacional, 2005.
- BON, F. B. G. Y. F. C. *Curso de C#. La plataforma .NET* Disponible en: http://elvex.ugr.es/decsai/csharp/dotnet/index.xml
- CASANOVAS, J. *Usabilidad y arquitectura del software.* 2004, Disponible en: http://www.desarrolloweb.com/articulos/1622.php
- CORDÓN, N. *La simulación de procesos ayuda a las empresas a ser más innovadoras. En determinadas áreas, España tiene un papel muy destacado a nivel internacional.* 2008, vol. Nº:1189, Pág.: 10 p. Disponible en: http://www.idg.es/computerworld/articulo.asp?id=191293
- DAVID GARLAN, M. S. *Introducción a la Arquitectura de Software. Avances en Ingeniería de Software and Knowledge Engineering* World Scientific Publishing Company, de I]. Disponible en: http://translate.google.com.cu/translate?hl=es&sl=en&u=http://www.cs.cmu.edu/afs/cs/project/vit/ftp/pdf/intro_softarch.pdf&ei=KgOfSc-aJpPHtgec4PGEDQ&sa=X&oi=translate&resnum=1&ct=result&prev=/search%3Fq%3DAn%2Bintroduction%2Bto%2BSoftware%2BArchitecture%26hl%3Des%26lr%3Dlang_es%26sa%3DG
- FERNÁNDEZ, S. G. *Arquitectura de un Nodo Virtual de Procesos.* Universidad de las Ciencias Informáticas, 2008.
- BUSCHMANN, FRANK, M., HANS ROHNERT, SOMMERLAD PETER, STAHL MICHAEL. *Pattern-Oriented Software Architecture-A System of Patterns.* New York: NY: John Wiley and Sons, Inc, 1996.

- GARAVITO, L. A. *OPTIMIZACION DE PROCESOS - SIMULACION & MODELAMIENTO* Colombia: Disponible en: http://www.virtual.unal.edu.co/cursos/economicas/2008551/lecciones/cap2-4-1.htm
- GNUGNOLI, H. M. Y. G. *DEFINICIONES DE SIMULACIÓN.* Universidad Nacional de Colombia, 2004.
- INFANTE, A. Modelos de Desarrollo. En *Universidad Centro Occidental "Lisandro Alvarado". 2003*.
- JUAREZ, A. *Arquitectura de Software* Disponible en: http://www.alfrek.net/blog/2008/05/arquitectura-de-software/
- KRUCHTEN, P. *Architectural Blueprints—The "4+1" View Model of Software Architecture* Rational Software Corp. ed. IEEE Software 12, Disponible en: http://materias.fi.uba.ar/7572/
- LEÓN, E. *Tutorial Visual Paradigm for UML* Disponible en: http://www.slion2000.blogspot.com
- FUENTES SUÁREZ, MAINOLDIS, A. R. *Propuesta de arquitectura para sistemas de gestión hospitalaria.* Universidad de las Ciencias Informáticas, 2007.
- NUÑEZ, E. C. F. C. G. *Arquitecturas de Software.* 2004.
- P. CLEMENTS, L. B., R. KAZMAN. *Software Architecture in Practice.* Editado por: Series, S. 2003.
- PARRA, R. A. M. *Definición de Simulación* Disponible en: http://docencia.50webs.com/simula01.htm
- PIZARRO, P. *Qué es un arquitecto de software.* 2006, Disponible en: http://arquitectura-de-software.blogspot.com/2006/12/qu-es-un-arquitecto-de-software.html
- PRESSMAN, R. *Ingeniería del Software: Un Enfoque Práctico.* McGraw Hill ed. Madrid, España: 2006. 601 p.
- Ingeniería del Software: Un Enfoque Práctico. En: PROCESS, R. U. 2003.
- REYNOSO, C. B. *Introducción a la Arquitectura de Software* Disponible en: http://www.willydev.net/descargas/prev/IntroArq.pdfU

- SÁNCHEZ, X. E. L. *Proyecto de Software Educativo para mantener habilidades adquiridas por personas con necesidades educativas especiales de tipo intelectual y con carácter permanente.* Universidad Central "MARTA ABREU" DE LAS VILLAS, 2007.
- SCENNA, N. J. *Modelado, Simulación y Optimización de Procesos Químicos.* 1999. pag-213 p.
- SHANNON, R. *DEFINICIONES DE SIMULACIÓN.* Universidad Nacional de Colombia, 2006.
- SOLUTIONS, C. *Word Magic Software, Inc.* Texas, Huston: 2008, Disponible en: http://www.wordmagicsoft.com/dictionary/es-en/proceso.php
- VERA MESTANZA, A. N., PIZARRO PINEDA, MARCOS LISTER *Simulación y Análisis del Sistema de Destilación de la Destilería CHICLAYO S.A.C empleando el Simulador de Procesos HYSYS.* Facultad de Ingeniería Química e Industrias Alimentarías. UNIVERSIDAD NACIONAL PEDRO RUIZ GALLO 2009.
- CORRALES VALDÉS, YURISNEL, J. C. S. L. *Desarrollo de la arquitectura del Sistema para la Simulación de Procesos en la Industria Química Cubana.* Habana: Universidad de las Ciencias Informáticas (UCI), Disponible en: http://biblioteca.uci.cu/sbd/biuci/index.html

BIBLIOGRAFÍA

- LUYBEN, W.L... Process Modeling, Simulation and Control for Chemical Engineers. McGraw-Hill (1990).
- ASPEN PLUS, Manuales del programa.
- Cruselles, Ernesto J. Cruselles, José L. Melús Moreno. *Secuencias seudo aleatorias para telecomunicaciones.* Publicado por Ediciones UPC, 1996. ISBN 8483011646, 9788483011645,280 páginas.
- Cerrada, J. A., & Collado, M. (2000). *Introducción a la ingeniería del software.* Ramón Areces. Publicado por Editorial Ramón Areces, 2000. ISBN 8480044179, 334 páginas.
- Acebes L., Alves R., Merino A., de Prada César. "*Un Entorno de Modelado Inteligente y Simulación Distribuida de Plantas de Proceso".* Revista iberoamericana de automática e informática industrial (RIAI), ISSN [en línea]. 1697-7912, Vol. 1, Nº 2, 2004, págs. 42-48.
- Reynoso, C. B. (2004). Introducción a la Arquitectura de Software. Consultado en Enero 23, 2009. Disponible en: http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp
- Hernández Orallo, *Enrique. "El Lenguaje Unificado de Modelado".* [Consultado en: 18/2/2009]. Págs. 2-6. Disponible en: http://www.disca.upv.es/enheror/pdf/ActaUML.PDF.
- G. Booch, J. Rumbaugh y I. Jacobson, "El Lenguaje Unificado de Modelado ", Addison Wesley, 1999
- I. Jacobson, G. Booch, J. Rumbaugh , "El Proceso Unificado de Desarrollo", Addison Wesley, 2000
- Sitio de Descargas de Software *.Visual Paradigm for UML (ME) - (Paradigma Visual para UML (ME)) (Visual Paradigm for UML (ME)) 6.0. Disponible en:*

http://www.freedownloadmanager.org/es/downloads/Paradigma_Visual_para_UML_(M%C3%8D)_14720_p/

- COOS BU, Raúl. Simulación. Un Enfoque Práctico. México. Editorial Limusa. 1986.
- SHANON, Robert E. Simulación de Sistemas. México. Editorial Trillas. 1988.
- Pattern-Oriented Software Architecture: A System of Patterns" Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal.
- Scenna Nicolás J., "Modelado, Simulación y Optimización de Procesos Químicos". Editorial de la Universidad Tecnológica Nacional, 1999. Rosario, Argentina. Cap. VI - Pág. 213.
- Casanovas, Josep. "Usabilidad y arquitectura del software". *Alzado.org,* 2004. Madrid, España. [En línea]. Disponible en: http://Usabilidad y arquitectura de software - alzado_org.mht
- Olmedilla Arregui, Juan José. *Revisión Sistemática de Métricas de Diseño Orientado a Objetos.* Facultad de Informática. Trabajo tutelado de Investigación. Universidad Politécnica de Madrid, 2005.
- Astudillo, H., Visconti, M. *Fundamentos de Ingeniería de Software.* Departamento de Informática. Universidad Técnica Federico Santa María.
- Douglas JM (1988). *Diseño Conceptual de Procesos Químicos.* McGraw-Hill. ISBN 0-07-017762-7.
- *Garret, J.J. (2002)*. Un vocabulario visual para describir arquitectura de información y diseño de interacción. Traducción: Javier Velazco (marzo 2002). Disponible en: http://www.jjg.net/ia/visvocab/spanish.html
- Clements Paul, Kazman Rick, Klein Mark. "Evaluating Software Architectures: Methods and Case Studies". ISBN 0-201-70482-X.

ANEXOS:

Anexo 1

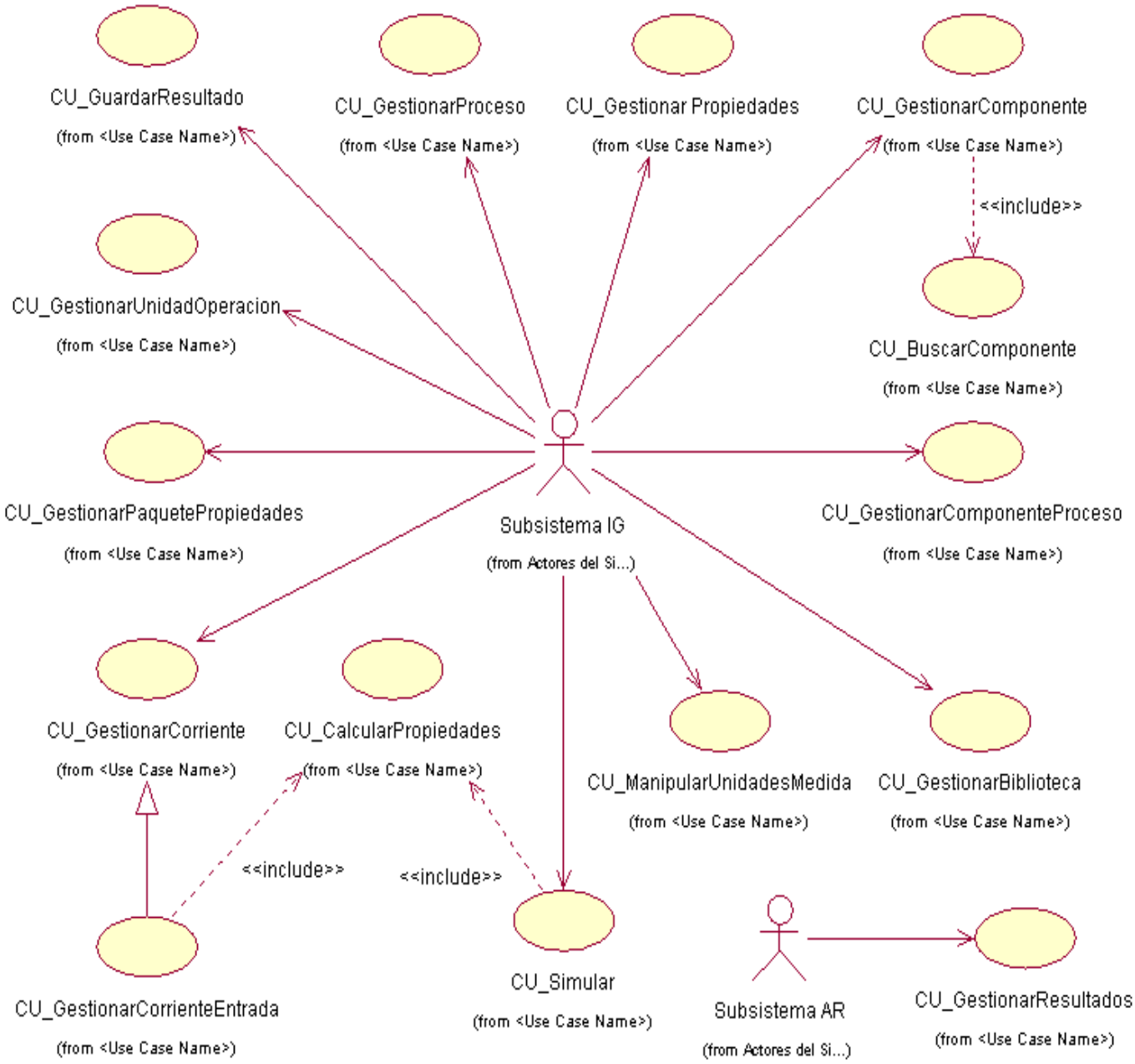

Ilustración 58: Diagrama de Casos de Uso del Subsistema de Modelos y Propiedades

Anexo 2

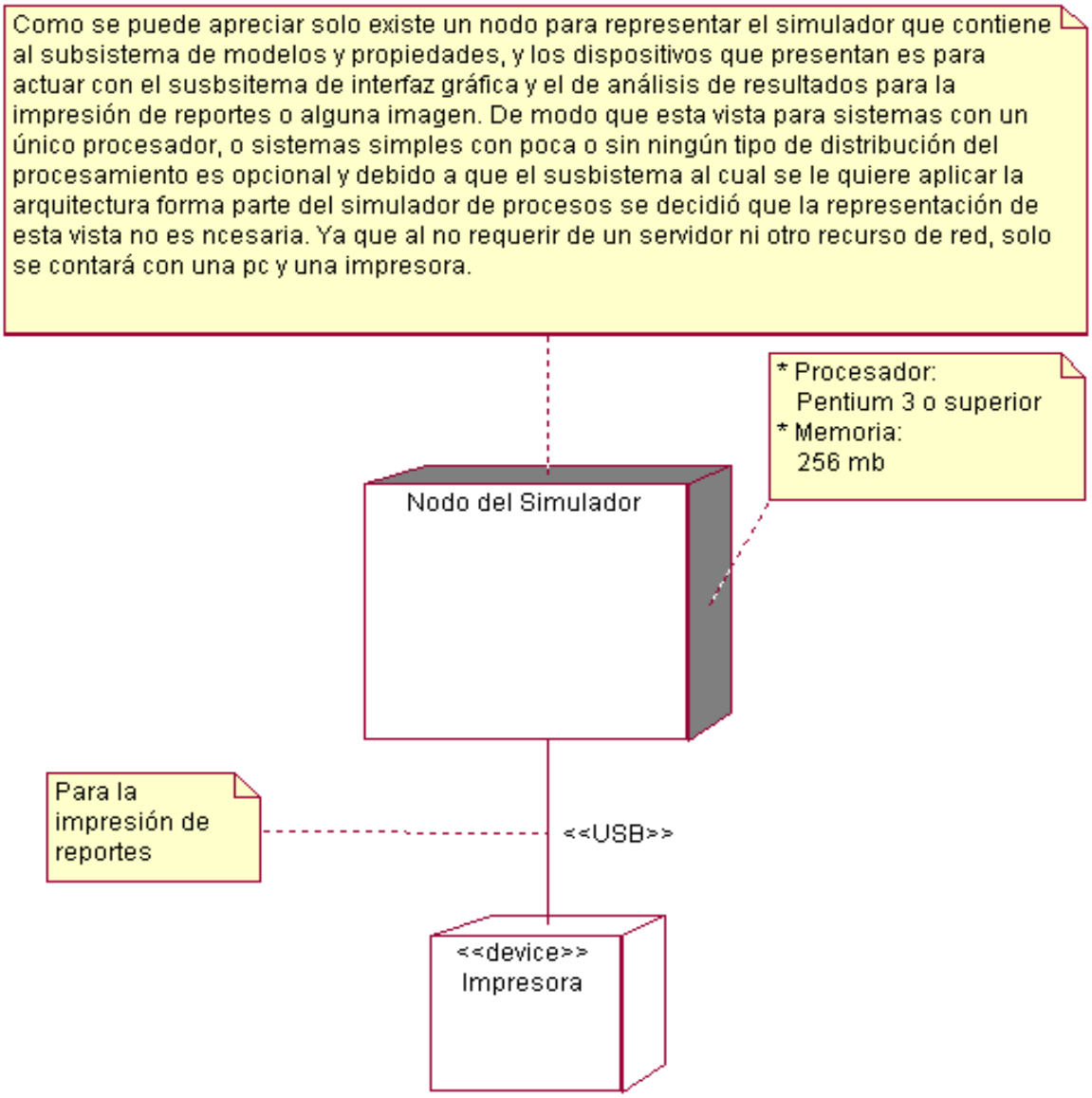

Ilustración 59: Información del diagrama de despliegue

GLOSARIO

En el presente trabajo se han expuesto algunos conceptos relacionados con algo tan complejo como la simulación de procesos industriales. Debido a esto y con el fin de esclarecer cualquier duda que pudiera surgir en los lectores, se ha escrito el siguiente glosario de términos en forma resumida, con los aspectos que se consideran esenciales de acuerdo al tema tratado en el documento.

Proceso: Conjunto de operaciones químicas y/o físicas en las cuales están involucrados determinados equipos o dispositivos, así como componentes químicos. Un proceso representa las mezclas, separación, calentamiento y enfriado de estos componentes a través de los equipos presentes. El proceso tiene como objetivo obtener un conjunto de productos o sustancias que se diferencian del conjunto inicial en su composición o estado.

Modelo Matemático: Sistema de ecuaciones que simula el funcionamiento real de un equipo determinado, tiene asociado un conjunto de variables de entrada, que deben ser especificadas, esto es datos conocidos a través de los cuales por medio de la resolución del modelo matemático se obtienen un conjunto de variables de salidas o variables desconocidas.

Componente químico: Sustancia química la cual tiene asociado un conjunto de propiedades físico-químicas.

Flujos: Representan las corrientes de entrada o salida a los equipos.

Propiedades físico-químicas: Conjunto de propiedades ya sean físicas o químicas que representan las características de un componente químico determinado.

Unidad de operación, Equipo, o módulo: Representan a los equipos que son usados en la realidad en los distintos procesos. Tienen asociado un conjunto de modelos matemáticos, en dependencia de la exactitud de la representación deseada para simular su funcionamiento o de las características específicas del proceso.

Módulo: Modelo matemático específico, que se diferencia de otros modelos que tienen las mismas ecuaciones en las variables independientes, seleccionadas para cubrir los grados de libertad.

Objeto Persistente*:* Objeto instanciado en memoria que debe ser almacenado en un medio no volátil.

Esquema de persistencia*:* Conjunto reutilizable de clases que presentan servicios a los objetos persistentes.

Materialización: Es el acto de transformar una representación no orientada a objetos como los datos o registros a objetos.

Desmaterialización: Acto contrario a Materializar.

Polling: en computación hace referencia a una operación de consulta constante, generalmente hacia un dispositivo de hardware, para crear una actividad sincrónica sin el uso de interrupciones, aunque también puede suceder lo mismo para recursos de software.

Publisher: La entidad responsable de hacer que el recurso se encuentre disponible en la red en su formato actual, por ejemplo la empresa editora, un departamento universitario u otro tipo de organización.

Data Mining: Es un término genérico que engloba resultados de investigación, técnicas y herramientas usadas para extraer información útil de grandes bases de datos.

Vista: Descripción simplificada (abstracción) de un modelo, vista desde una perspectiva dada o posición de ventaja que omite entidades no relevantes para esta perspectiva.