Universidad de las Ciencias Informáticas
Facultad 3

Título: Sistema para la descarga y procesamiento automatizado de patentes: Predictor.

Rol de arquitecto

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autores: Mileisys Plasencia Crespo

Sergio Jesús García de la Puente

Tutor: Rolando Camacho Pupo

Consultante: Ridosbey Milian Iglesias

Junio de 2007

AGRADECIMIENTOS

A mis padres por haberme dado la vida y confiar siempre en que saldría adelante, gracias a ellos hoy puedo realizar un sueño importante en mi vida.

A Lázaro por todo su cariño y apoyo durante estos años de estudio.

A la persona con la que he compartido trabajos, seminarios, proyectos de curso en mi carrera y ahora tengo el placer de compartir mi trabajo de diploma, a quien ha sabido ser mi amigo, mi hermano, mi padre y el mejor esposo, a ti mil gracias, por cada día darme tanto amor y cariño.

A mi hermanito querido por su ternura y espero un día poder estar en tus agradecimientos, porque yo sé que vas a ser un magnífico profesor.

A mis abuelos que yo se que están orgullosos de mi porque he cumplido sus expectativas.

A Tata, Isbel, Dianelys, Rosa y Papo que han sido un eslabón fundamental en mi formación, gracias por su preocupación constante, su cariño y comprensión.

A Maritza, Cecil y Jesús por ser los mejores suegros del mundo.

A mi tutor Rolando Camacho por su preocupación y apoyo en la realización de este trabajo.

A mis compañeros de aula y mis amigos, en especial a aquellos que cuando más los necesité estuvieron a mi lado y me hicieron reír.

A mis compañeros del proyecto con los que siempre aprendí algo nuevo, en especial a Luisito.

A mis profesores y a todas aquellas personas que contribuyeron a mi formación como ingeniera en esta maravillosa universidad.

Mileisys.

AGRADECIMIENTOS

A mi esposa, por ser la persona que más me ha ayudado directamente en lograr esta meta de hoy, mis padres jugaron un papel importantísimo e indispensable, pero realmente, sin ella este sueño quizás me hubiese costado un poco más de esfuerzo. Su comprensión, solidaridad, dedicación, abnegación, constancia y amor, son fruto de esta gran alegría.

A mi madre por toda su confianza y amor hacia mí, por ser incondicional en todo momento y por su siempre comprensión cuando le traía un problema.

A mi padre, mi faro y guía, mi patrón, mi gran motor impulsor, el que nunca ha dejado de darme energía, incluso en los momentos difíciles.

A mis abuelitas, que siempre han confiado en mí y siempre me han dado su bendición, ya ven, no las defraudé.

A mi tío Sergio, mi otro patrón y motivo de inspiración, quien también ha confiado siempre en mí y ha disfrutado de mis logros, enorgulleciéndose por cada aspecto relevante de mi vida.

A Cecil, por su especial apoyo, por su amor para conmigo y mi esposa, y por sus lágrimas que siempre son espontáneas y llenas de amor, es por esto que eres tan especial.

A mi hermano Arielito, de quien siempre he recibido apoyo, puntos de vistas y criterios inteligentes.

A mis demás tíos y mis primos, quienes siempre me han deseado suerte y han mostrado preocupación por mis exámenes y eventos docentes.

A mi suegra y mis suegros, por apoyar y entender a mí esposa, cuando tenía que dedicarme especial atención; por darnos apoyo a los dos para nuestra formación y ánimo para salir adelante.

Especial agradecimiento a nuestro Tutor Ing. Rolando Camacho Pupo, quien ha dado muestras de incondicionalidad, aportándonos ideas muy valiosas, reflexiones interesantes y recomendaciones inteligentes, convirtiéndose en nuestro principal crítico y compañero, quien nos ha tratado con sumo respeto y mostrando especial interés en nuestro trabajo.

A todos mis profesores, los que confiaron y creyeron que podía llegar a la cima, incluso viéndome en circunstancias complejas y a los que no lo creían, tienen una enseñanza más, la confianza en el

ser humano es lo último que se pierde. Especial agradecimiento en mi formación a Tony, Teddy, Carlos y Yoan.

A mis especiales amigos desde primer año, quienes siempre me han tratado con atención esmerada y desinteresada, muchas gracias a Pedrito, Alain, Nani (Darlenis), Laura, Maidi, Yanelys, Hernyk, Dionis, Vilma, Damaris, Alexis, Luisito y en fin a todos mis compañeros de grupo en todos estos años de estudio intenso.

Y el especial agradecimiento, al amor que existe entre mi compañera de tesis y yo, ingrediente indispensable para llegar a donde llegamos hoy.

Sergio J.

DEDICATORIA

A nuestro Comandante en Jefe por ser el principal impulsor de esta gran idea que se llama UCI y que nos forma como Ingenieros al servicio de la Revolución.

A nuestros padres y hermanos.

A nuestras familias.

A todos los amigos de la UCI y los que no son de la UCI también.

A nuestro eterno amor….

Mileisys y Sergio J.

RESUMEN

En el trabajo de diploma se define la arquitectura del Sistema para la descarga y procesamiento automatizado de patentes “Predictor”, para lo cual se tienen en cuenta diferentes aspectos:

Se realiza un estudio del estado del arte de los estilos arquitectónicos más utilizados hoy en día y de los principales modelos y tendencias de la arquitectura, se analizan los diferentes software de descarga y análisis de información existentes, especificando las ventajas y desventajas de cada uno de ellos y el por qué no pueden ser utilizados en la consultoría Delfos.

Se definen las herramientas y la plataforma de desarrollo así como la metodología que guiará el proceso de desarrollo del sistema Predictor.

Se define la línea base de la arquitectura y se realiza la descripción de la misma mediante las diferentes vistas arquitectónicas propuestas por RUP (Vista arquitectónica del Modelo de Casos de Uso, Vista arquitectónica del Modelo de Análisis, Vista arquitectónica del Modelo de Diseño, Vista arquitectónica del Modelo de Despliegue, Vista arquitectónica del Modelo de Implementación)

Se definen los patrones de diseño utilizados y se fundamenta la utilización del patrón arquitectónico en capas.

En la actualidad en Cuba no existe ningún software que realice la descarga y procesamiento de la información relacionada con las patentes publicadas en Internet, por lo que este sistema tendrá un impacto significativo especialmente para los especialistas de la consultoría Delfos.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Desde la antigüedad la humanidad ha ido en pos del desarrollo y avance en todos los sectores de la sociedad, para lo cual ha sido necesario organizar y establecer métodos formales para legalizar cada uno de los descubrimientos que han ido surgiendo a lo largo del proceso de desarrollo, con el objetivo de tener registrado el autor del descubrimiento, la fecha de creación, entre otros aspectos. Producto de esta escasez de métodos legales para registrar estos eventos surgió la necesidad de certificar o emitir derechos de autoría de todo aquello que constituya una innovación o quiera ser comercializado y es lo que se conoce hoy como patente (1).

Una patente es un conjunto de derechos exclusivos garantizados por un gobierno o autoridad al inventor de un nuevo producto (material o inmaterial) susceptible de ser explotado industrialmente para el bien del solicitante de dicha invención durante un espacio limitado de tiempo (generalmente veinte años desde la fecha de solicitud), con la finalidad de prohibir a cualquier otra persona fabricar, utilizar o vender el producto, procedimiento o método patentado.

El uso de patentes como indicador de innovación ha sido exhaustivamente estudiado y ha alcanzado un gran nivel de madurez en la actualidad debido a que las patentes son documentos que contienen información muy valiosa desde el punto de vista legal, técnico y comercial. Las patentes son de gran importancia, no solo para determinar indicadores de innovación de un país determinado, sino para establecer estrategias de negocios de las compañías tecnológicas y políticas de desarrollo en países menos desarrollado. Se considera que el 80% de la información que aparece en las patentes no aparece publicado en otras fuentes (1), por lo que constituyen una fuente ideal para la vigilancia tecnológica y la inteligencia competitiva.

La Consultaría del Ministerio de informática, con su nombre abreviado Delfos, Centro Coordinador del Sistema de Información del Ministerio de la Informática y las Comunicaciones de Cuba, se encarga de ejercer la vigilancia tecnológica permanente en diferentes temáticas de interés para el desarrollo de las tecnologías de la informática y las comunicaciones (TIC). El trabajo de los especialistas de esta consultoría puede incluir tanto el acceso a Bases de Datos de Patentes (BDP) como a otras Bases de Datos (BD) bibliográficas de tipo científico-tecnológico. Estas BD pueden estar situadas en un puesto de una red local o en un sitio web de Internet. En caso de que las BDP estén publicadas en Internet el especialista se conecta al sitio web correspondiente y descarga las páginas de cada uno de los documentos de patentes que le interese.

Las BDP presentes en Internet pueden ser comerciales o gratuitas. En el caso de las de acceso libre permiten que cualquier persona interesada que tenga una computadora conectada a Internet pueda acceder a los documentos que contienen las patentes publicadas. Esas BD se encuentran en línea y no están limitadas a las fronteras nacionales. Entre ellas se pueden mencionar las BD de Espacenet (Esp@cenet), World Wide Web (WWW), United States Patent and Trademark Office (USPTO), State Intellectual Property Office (SIPO) de China, Patent Abstract of Japan (PAJ) a las cuales se conectan sistemas informáticos, implementados para la descarga y análisis de las patentes, artículos científicos y otro tipo de información. Estos documentos, son procesados y sometidos a un análisis de información para elaborar productos y servicios de Inteligencia empresarial, algunos de los estudios son: tendencias, mercado y perfiles estratégicos.

Actualmente en la Consultoría Delfos los especialistas hacen manualmente la descarga de muchas patentes publicadas en Bases de Datos de Internet, lo cual es un proceso lento y engorroso, luego a dicha información ya descargada, se le aplican diferentes tipos de filtros para homogeneizarla y extraer datos bibliográficos de las patentes, hasta el momento este proceso se realiza de forma manual y a través de un proceso muy complicado, lo cual trae consigo demoras y en ocasiones, se producen incumplimientos por parte de los especialistas encargados de realizar dichas tareas; por esta razón se hace inminente realizar una aplicación con una arquitectura confiable que resuelva esta situación.

Formulación del problema

La no aplicación de una arquitectura robusta y confiable para el desarrollo del sistema “Predictor”, afecta el correcto desempeño del proyecto y compromete el futuro mantenimiento, flexibilidad y reusabilidad del producto una vez implantado.

Objeto de Estudio

La arquitectura en el ciclo de desarrollo de software.

Campo de Acción

La arquitectura en el Sistema para la descarga y procesamiento automatizado de patentes “Predictor”.

Hipótesis

Si se define una línea base coherente con las funcionalidades del sistema y es implementada durante el desarrollo del mismo se puede lograr una mayor confiabilidad en la calidad del producto final y se garantiza la total facilidad de mantenimiento del sistema.

Metodología de la Investigación

Para realizar la parte investigativa de este trabajo se siguieron los siguientes pasos:

- Recopilar información necesaria para el proyecto.
    - Análisis de la bibliografía disponible.
    - Búsqueda de textos disponibles en Internet.
    - Entrevistas con los especialistas de la Consultoría Delfos.

Seleccionar y organizar datos necesarios para el proyecto.

Objetivo General

Definir la arquitectura que se usará para la implementación del sistema “Predictor”.

Objetivos Específicos

- Definir la metodología y las herramientas para el desarrollo.
- Definir la Línea Base de la arquitectura.
- Realizar la descripción de la arquitectura.
- Definir el patrón arquitectónico.
- Evaluar el diseño arquitectónico.

Para dar respuesta a estos objetivos se investigó acerca del estado del arte de los modelos y estilos arquitectónicos más usados hoy en día, indagando en una propuesta de patrón arquitectónico acorde con los sistemas de análisis de información, se le expuso al cliente cuáles son los principales requerimientos de la arquitectura seleccionada, se hizo uso de bibliografía confiable para indagar más sobre el tema de los estilos y vertientes arquitectónicas.

En la actualidad con el avance de las tecnologías en el campo de la informática, los procesos de desarrollo de software se hacen cada vez más complicados; un componente dentro de este proceso que constituye una solución a muchos de los problemas es el desarrollo de arquitecturas de software que ayuden a la descomposición de la complejidad de los proyectos informáticos.

La clave de una buena arquitectura de software está basada en la correcta selección del estilo arquitectónico a usar. Es amplia la gama de estilos existentes, han sido escritos aproximadamente de 20 a 30 estilos, una cifra bastante modesta, que contrasta si se tiene en cuenta que el dominio de los mismos dificulta en demasía la elección de uno en especifico con vistas a ser aplicado al proyecto en curso. En la actualidad existen varios modelos y tendencias de la arquitectura dentro de las cuales se encuentran:

- Arquitectura como etapa de ingeniería y diseño orientada a objetos.
- Arquitectura estructural, basada en un modelo estático de estilos, ADLs y vistas.
- Estructuralismo arquitectónico radical.
- Arquitectura basada en patrones.
- Arquitectura procesual.
- Arquitectura basada en escenarios.

Y dentro de los estilos arquitectónicos más usados hoy en día se encuentran::

- Arquitecturas basadas en eventos.
- Arquitecturas orientadas a servicios. (SOA)
- Arquitecturas basadas en recursos.
- Arquitecturas en capas.
- Arquitecturas basadas en componentes.
- Arquitecturas orientadas a objetos.

La necesidad de este trabajo de diploma radica en proporcionar la línea base de la arquitectura del sistema “Predictor”, además de la descripción de la arquitectura mediante las diferentes vistas arquitectónicas del sistema, logrando de esta forma implementar un sistema robusto que responda a las necesidades de los clientes.

El trabajo está dividido en introducción, tres capítulos, conclusiones, recomendaciones, bibliografía y anexos, los capítulos están distribuidos de la siguiente forma:

Capitulo I: Fundamentación Teórica. En este capítulo se realiza el estudio del estado del arte de los principales sistemas de descarga y análisis de patentes existentes. Se hace una breve reseña sobre lo que es arquitectura de software, además de un análisis crítico de los principales modelos y tendencias actuales de la arquitectura, así como de los principales estilos arquitectónicos. Se fundamenta la utilización de tecnologías y herramientas en el desarrollo de este proyecto.

Capitulo ll: Se mencionan los requisitos funcionales y no funcionales que son fundamentales para definir la línea base de la arquitectura, se presenta la descripción arquitectónica de cada uno de los modelos del sistema. Se justifica la utilización de los patrones utilizados dentro de los que se encuentra el patrón arquitectónico en capas, se define la estrategia de integración de los diferentes componentes y se especifica la configuración de los puestos de trabajo según los roles del equipo de desarrollo.

Capitulo lll: En el capítulo 3 se hace una valoración de la arquitectura mediante la aplicación de métricas que permiten evaluar el diseño arquitectónico, se describe el impacto que tuvo en el sistema Predictor y los beneficios que este reportó después de instalado en la Consultoría Delfos.

1 FUNDAMENTACIÓN TEÓRICA.

1.1 Introducción

En la actualidad el estudio sobre las patentes se ha convertido en un tema importante para aquellos países o empresas en desarrollo. Gracias a la información que brindan se pueden tomar decisiones importantes que posibilitan establecer pautas, estrategias y políticas a seguir para lograr insertarse en el mercado mundial. Por ello las patentes constituyen una fuente importantísima de información sobre la competencia, lo que las hace una herramienta principal de la vigilancia tecnológica. Por otro lado conocer qué patentes están vigentes en el mercado es fundamental antes de ponerse a desarrollar nuevos productos, con una considerable inversión que luego podría verse bloqueada.

Para la realización de estos estudios se han desarrollado diferentes sistemas que realizan el análisis y procesamiento de la información sobre las patentes.

A continuación se explican algunos de estos sistemas especializados en la descarga y procesamiento de patentes; así como temas de interés a la hora de llevar a cabo el diseño arquitectónico del sistema que se propone. Se tratan temas relacionados con las herramientas definidas para el desarrollo, algunas de las metodologías más usadas hoy en día, entre otros.

1.2 Sistemas de descarga y análisis de información de patentes.

Muchas oficinas de patentes permiten ya descargar gratuitamente textos completos de sus patentes. Entre ellos la USPTO de Estados Unidos, la EPO Europea, entre otras. En particular Esp@cenet, el servicio de descargas de la EPO se ha convertido en una fuente de información muy utilizada en este campo.

En base a las posibilidades de búsqueda que todo ello ofrece, en el mercado han ido apareciendo diversos sistemas que permiten buscar, descargar y analizar las patentes de forma automática. Si bien la descarga, clasificación, agrupación y relación entre patentes de contenidos similares es relativamente popular, la visualización de dichos resultados no está muy extendida.

Entre los sistemas más conocidos se encuentran:

Matheo Patent: Hasta el momento es el que más se asemeja al sistema Predictor. Realiza la búsqueda, recuperación y análisis de patentes de las bases de datos de la Oficina de Patentes de

los EE.UU. (USPTO) y de la Oficina Europea de Patentes (EPO). Posee una interfaz amigable y consta de diversas funcionalidades para la recuperación y análisis de patentes.

PatentHunter: Realiza la búsqueda, descarga y otras funcionalidades como por ejemplo, mostrar el documento en formato pdf, HTML y permite imprimirlo.

BizInt Smart Charts: Carga la información obtenida de Bases de Datos de patentes como WPI, CA/CAplus, MicroPatent, Delphion y otras. Permite cargar esa información en una hoja electrónica, reformatearla y hacerle análisis estadísticos.

Aureka: En su versión ThemeScape los temas se representan visualmente en mapas con aspecto cartográfico, identificando los conceptos predominantes y sus relaciones; además se pueden comparar compañías, competidores o tecnologías. En su versión Citation Tree construye un árbol de citas a partir de una patente seleccionada.

Delphion-Text Clustering: Es una función en línea que extrae los términos más relevantes del texto de los resultados de una búsqueda, analiza sus relaciones y los presenta en una mapa.

Bibexcel: Permite combinar la información extractada de diferentes campos de un registro, realiza conteos por frecuencia, co-ocurrencias de diversos elementos, y emparejamiento bibliográfico. Además de estas tareas, cuenta con un procedimiento para encontrar enlaces de citas entre diferentes documentos de un conjunto determinado.

PatentLab-II: Se utiliza sólo para analizar datos descargados de la base de datos Thomson Delphion. Posibilita la agrupación por familias de patentes, para evitar las duplicidades en el análisis en los conteos de los diferentes campos tales como año de publicación, entidad solicitante o titular, inventor, temáticas, etc.

VantagePoint: Software para el análisis de información científico-técnica y de patentes desarrollado por Search Technologies. Permite la visualización de relaciones mediante matrices de co-ocurrencia, mapas tecnológicos y la creación de tesauros para la reducción de datos. Además, realiza análisis estadísticos multidimensionales para identificar grupos y relaciones entre conceptos, autores, países.

Los sistemas antes mencionados brindan ventajas y facilidades para el análisis de la información sobre patentes de Internet, pero debido a la situación actual que tiene el país y las leyes económicas que existen contra Cuba, los especialistas de la consultoría Delfos se ven

imposibilitados de utilizarlos debido a que la mayoría de estos son propietarios y hay que pagar la licencia para su uso.

Algunos de ellos son libres, o sea no se paga por el uso del software, pero es necesario pagar por la información que se descargue de las bases de datos a las que acceden; siendo este caso muy similar al primero.

Existen otros aspectos que impiden el uso de algunos de estos software por los especialistas de Delfos y es que la mayoría de los sistemas se conectan a bases de datos específicas y no permiten realizar búsquedas en distintas fuentes; los análisis y gráficos son predeterminados y es necesario ajustarse a las especificaciones que brindan, siendo obligatorio hacer el análisis de la información como viene definido. Estos dos aspectos se deben a que los sistemas son desarrollados por empresas para uso específico y por tanto no se ajustan a las necesidades de Delfos.

En Cuba existen otras organizaciones que realizan el estudio de patentes, pero ninguna cuenta con una herramienta que automatice completamente el proceso, dependen de otros sistemas para la realización de los análisis.

1.3 Breve reseña sobre la arquitectura de software

Al escuchar la terminología arquitectura, rápidamente se piensa en la más general de las arquitecturas, la de un edificio y vienen a la mente diferentes características, entre ellas la estructura física que pueden tener estas construcciones. Según el significado que brinda el Diccionario de la Real Academia Española la arquitectura es: “el *arte de proyectar y construir edificios*” (2). Es decir, en todo el proceso de construcción de edificios, el arquitecto será el responsable de ver como integra los distintos componentes de la construcción para formar un todo consistente, este buscará la forma, apoyado en la arquitectura de hacer que los edificios encajen en su entorno y con los adyacentes a su vecindad; este proceso termina cuando cumple con las perspectivas que dieron origen a la obra y además con la satisfacción de sus clientes; pero es la estética la que la define como un arte, cuando llega el momento de la integración de los colores y materiales para la fachada externa, así como el diseño de las instalaciones eléctricas, del tipo de suelo, de la colocación de los tapices, entre otros conjuntos de materiales.

La Real Academia Española plantea que arquitectura para el perfil de informática es la: “e*structura lógica y física de los componentes de un computador*” (2), la cual está más bien centrada en toda la

estructura y composición de la parte dura de los ordenadores (hardware). Muchas figuras importantes de la Ingeniería del Software actual han dado su apreciación respecto a la AS.

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

En una definición tal vez demasiado amplia, David Garlan (3) establece que la AS constituye un puente entre el requerimiento y el código, ocupando el lugar que en los gráficos antiguos se reservaba para el diseño. La definición “oficial” de AS se ha acordado que sea la que brinda el documento de IEEE Std 1471-2000, adoptada también por Microsoft, donde enfoca que:

La Arquitectura de Software es la organización fundamental de un sistema encarnada en sus componentes, las relaciones entre ellos y el ambiente y los principios que orientan su diseño y evolución. (3)

Bass por otro lado también gestiona su perspectiva de la AS y plantea:

La arquitectura de software de un sistema de programa o computación es la estructura de las estructuras del sistema, la cual comprende los componentes visibles externamente, y las relaciones entre ellos (4).

Roger Pressman (4) enfatiza que la arquitectura de software no se puede ver como un sistema operacional, sino como una representación que le brinda una serie de capacidades al Ingeniero de Software, entre ellas:

1. Analizar la efectividad del diseño para la consecución de los requisitos fijados.
2. Considerar las alternativas arquitectónicas en una etapa en la cual hacer cambios en el diseño es relativamente fácil.
3. Reducir los riesgos asociados a la construcción del software.

En este mismo texto, Pressman hace referencia a Jerrold Grochow y a su planteamiento de la congruencia existente entre la arquitectura y los componentes en las representaciones arquitectónicas, reflejando su cita donde plantea: “la arquitectura de un sistema constituye un

amplio marco que describe su forma y estructura – sus componentes y como estos encajan juntos – ". Es válido señalar que en las descripciones arquitectónicas los componentes pueden ser tan simples como complejos, desde un módulo del programa o sistema hasta las inclusiones de las bases de datos y software intermedio que permitan las configuraciones de redes entre clientes y servidores. De igual manera, las relaciones entre los componentes pueden ser tan sencillas como una llamada de procedimiento de un módulo a otro, o tan complicadas como el protocolo de acceso a bases de datos.

Bass, Clements y Kazman (5) argumentaron en tres razones el porqué de la importancia de la AS:

- Las representaciones de la arquitectura de software facilitan la comunicación entre todas las partes interesadas en el desarrollo de un sistema basado en computadora.
- La arquitectura destaca decisiones tempranas de diseño que tendrán un profundo impacto en todo el trabajo de ingeniería del software que sigue, y es tan importante en el éxito final del sistema como una entidad operacional.
- La arquitectura constituye un modelo relativamente pequeño e intelectualmente comprensible de cómo está estructurado el sistema y de cómo trabajar juntos sus componentes.

1.4 Análisis crítico de los principales modelos y tendencias de la Arquitectura.

En la década de los noventa del siglo pasado es definida la AS como un dominio, aunque separado del marco global de la ingeniería y del marco puntual del diseño. Realmente no existen estudios reconocidos que dicten modalidades de AS, pero en la actualidad se pueden distinguir unas seis corrientes. Algunas distinciones pueden encontrarse en ciertas bibliografías, pero una bibliografía sobre corrientes y alternativas prácticamente no existe.

Lo que si está claro es que pronunciar una clasificación de estrategias no admite una solución simple y determinista. Los practicantes de la AS ocasionalmente se mueven de una táctica a otra o evolucionan de puntos de vistas más genéricos a más particulares, o realizan diferentes trabajos operando marcos distintos. En las comunidades se han enfocado grandes debates metodológicos entre los métodos pesados y ligeros, pero lo cierto es que las discusiones entre las distintas posturas rara vez se han manifestado como choques frontales entre ideologías irreconciliables, por

lo que a menudo hay que leer entre líneas para intuir que una afirmación cualquiera es una crítica a otra manera de ver las cosas, o trasunta una toma definida de posición.

Las seis corrientes que más se destacan son:

1.4.1 Arquitectura como etapa de ingeniería y diseño orientada a objetos.

Es el modelo liderado por Craig Larman, James Rumbaugh, Ivar Jacobson, Grady Booch, ligados estrechamente al mundo del UML y Rational, sobre todo a estos tres últimos se le conocen como “Los Tres Amigos”, por ser principales precursores de esta corriente específica. Algunos autores han entrado en conflicto con sus doctrinas, pues dudan que se pueda tratar como una postura arquitectónica, exponiendo sus conflictos en cuanto al proceso de desarrollo, quizás porque ven en ella un corto alcance de la arquitectura, exponen además su desacuerdo con el modelado y sus diagramas, quizás algo denso y excesivo respectivamente para una descripción de la arquitectura y realzan un contundente desacuerdo a que sea asumido el UML como lenguaje por excelencia y subestimen la consistencia de los Lenguajes de Descripción Arquitectónicos (ADLs); señalan además la forma en que se utiliza la terminología de estilos, pues consideran que hacen acoplamientos tales que tienden a confundirlos con los conceptos de patrones arquitectónicos y de diseños (3).

En esencia esta modalidad define todo un proceso de desarrollo dirigido por casos de uso (CU), iterativo e incremental y centrado en la arquitectura, por tanto, sus fundadores describen este modelo de forma tal que la arquitectura se centre en elementos estructurales significativos del sistema, como subsistemas, clases, componentes, etc. De igual forma los CU son los encargados de darle al sistema la funcionalidad y uso aspirados, alcanzando a la vez objetivos de rendimiento razonables. La arquitectura es descrita por las conocidas 4 + 1 vistas del proceso que fueron desarrolladas de una forma iterativa e incremental durante las fases de inicio, elaboración y construcción, utilizando los CU significativos implementando la línea base y enarbolando la consigna de que la arquitectura debe de ser completa y lo suficientemente flexible como para incorporar nuevas funciones si así lo necesitara el cliente, los desarrolladores o el propio sistema, así como capacitada para soportar la reutilización del software existente (6).

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

Es considerada la corriente fundacional y por excelencia de la disciplina. Sus representantes y precursores: Mary Shaw, Paul Clements, David Garlan, Robert Allen, Gregory Abowd, John Ockerbloom; son todos académicos, más específicamente en su mayoría de la Carnegie Mellon University (CMU) – Software Engineering Institute (SEI), en Pittsburgh. Esta corriente, tiene como peculiaridad que es extremadamente dominante en la academia y en la práctica industrial sus categorías y herramientas son poco conocidas. La corriente se resalta en el alto nivel de abstracción que utiliza para hacer el diseño arquitectónico, en ningún momento se hacen referencias a lenguajes de programación o piezas de código, ni mucho menos se involucran los términos de clases o de objetos.

1.4.3 Estructuralismo arquitectónico radical.

Este modelo estructuralista es un desprendimiento del visto anteriormente, establecido principalmente en la zona europea y utiliza el UML a diferencia del anterior como lenguaje de modelado. En el seno de este movimiento hay al menos dos tendencias, una que excluye de plano la relevancia del modelado orientado a objetos para la AS y otra que procura definir nuevos metamodelos y estereotipos de UML como correctivos de la situación.

1.4.4 Arquitectura basada en patrones.

Esta corriente aparece por los mediados de la década de los noventa del siglo pasado y no está muy vinculada al modelado apoyado en UML, ni a la metodología CMM. Esta variante reconoce los textos de patrones descrito por Buschmann principalmente entre otros autores y conocido como la serie POSA y de manera secundaria reconocen el texto de la Banda de los Cuatro (Gang of Four <GOF>). En esta manifestación de la AS prevalece cierta tolerancia hacia modelos de procesos tácticos, no tan macroscópicos, y eventualmente se expresa cierta simpatía por las premisas de la programación extrema. El diseño consiste en identificar y articular patrones preexistentes, que se definen en forma parecida a los estilos de arquitectura.

1.4.5 Arquitectura procesual.

En los comienzos de este siglo, en la CMU y con centro en el SEI algunos de los arquitectos más reconocidos de la primera y segunda generación: Rick Kazman, Len Bass, Paul Clements, Felix Bachmann, Fabio Peruzzi, Jeromy Carrière, Mario Barbacci, Charles Weinstock; están llevando a cabo un modelo de ciclo de vida y técnicas de licitación de requerimientos, tormenta de ideas (brainstorming), diseño, análisis, selección de alternativas, validación, comparación, estimación de calidad y justificación económica especificas para la arquitectura de software. Toda esta información se puede encontrar concretamente en el SEI, pero no se mezcla jamás con la de CMM, a pesar de redefinirla de un extremo a otro. Otras variantes dentro de la corriente procesual caracterizan de otras maneras las etapas del proceso: extracción de arquitectura, generalización, reutilización.

1.4.6 Arquitectura basada en escenarios.

Es la corriente más nueva y predominante en Europa, con centro en Holanda. Aquí se recupera el nexo existente entre la AS con los requerimientos y la funcionalidad del sistema, lo cual ha sido casi olvidado en la arquitectura estructural clásica. Los fundadores o bien los que han establecido esta modalidad, es decir, han aportado sus teorías, así como los simpatizantes de esta corriente se adentran al mundo de la arquitectura procesual. En este tipo de corriente se usan diagramas de CU modelados por UML como herramienta informal u ocasional, dado que los CU son uno de los escenarios posibles, pero estos a su vez no están orientados a objeto.

Reynoso (3) expone que han existido intentos de creaciones de otras variantes de AS, tales como una arquitectura adaptativa inspirada en ideas de la programación genética o en teorías de la complejidad, la auto-organización, el caos y los fractales, una arquitectura centrada en la acción que recurre a la inteligencia artificial heideggeriana o al postmodernismo, y una arquitectura epistemológicamente reflexiva que tiene a Popper o a Kuhn entre sus referentes; todos estos criterios sustentados por los autores de variados textos que plantean además que consideran omitir todas estas variantes, ya que ni la masa crítica es notoria ni su mensaje parece sustancial.

1.5 Análisis crítico de los principales estilos arquitectónicos.

El hombre en el transcurso de su historia, ha creado métodos de conductas estándares, los cuales han cambiado en épocas y generaciones, en la mayoría con patrones de conductas fuertes, donde

el honor y las convicciones morales han plasmado un sello determinante, también en el paso de la humanidad, vicios como el proxenetismo, la corrupción y otros males sociales han crecido en algunas generaciones y superadas por otras, todas heredadas a través de los años, como legado tangible de sus predecesores, en sus normas o patrones de conductas personales y sociales; así es la vida de las personas, es un traspaso de vivencias, experiencias, iniciativas y acciones sustentadas por patrones y/o estilos coherentes a su época y el momento en que están circunscritos los individuos. Según (2), define la palabra estilo como “modo, manera, forma de comportamiento”, así como “uso, práctica, costumbre, moda”; por lo tanto cuando hacemos de una acción un uso periódico o en busca de una solución a un problema y esta se convierte en moda o estándar, podemos decir que se ha convertido en un estilo y/o patrón. En los finales de la década de los ‘60 y en el transcurso de los ‘70 y parte del ‘80, nuestro país se encontraba enfrascado en grandes obras de carácter social, tales como instituciones educacionales y de salud pública, además de sectores priorizados de la población, así que se comienzan a llevar a cabo proyectos constructivos de prefabricado de hormigón armado, ya que conllevaban a un aumento de las capacidades de producción de la construcción y reducía considerablemente el tiempo de ejecución de las obras, a éste tipo de tecnología de prefabricado se le comenzó a llamar Sistema Constructivo Girón (7), es por ello que cuando hacemos alusión a una construcción Girón nos es muy fácil recrear en nuestras mentes una construcción de este tipo o enseguida nos vienen imágenes de aquella escuelita o policlínico de nuestra localidad. El Sistema Constructivo Girón posee características específicas en cuanto a la habilitación de sus cimientos, la utilización de vigas y las losas para establecer las estructuras de los niveles de las edificaciones, así como los paneles que le darán forma a la fachada y delimitar sus interiores, es por ello que a un constructor no se le hace muy complicado imaginarse que estructura le tocará emprender, porque *el estilo arquitectónico es también para él un patrón de construcción por el cual se guiará en todo momento* (4).

En nuestra rama, el software construido para sistemas basados en computadoras también cuenta con diversos estilos arquitectónicos. Cada estilo describe una categoría del sistema que contiene:

1. Un conjunto de componentes (ej. Una base de datos, módulos computacionales) que realizan una función requerida por el sistema.

2. Un conjunto de conectores que posibilitan la comunicación, la coordinación y la cooperación entre los componentes.

3. Restricciones que definen como se pueden integrar los componentes que forman el sistema.

4. Modelos semánticos que permiten al diseñador entender las propiedades globales de un sistema para analizar las propiedades conocidas de sus partes constituyentes (4).

Una vez que la ingeniería de requisitos define las características y las restricciones del sistema que ha de ser construido, se escoge el patrón arquitectónico (estilo) o la combinación de patrones (estilos) que mejor encajan con las características y restricciones. En muchos casos, puede ser apropiado más de un patrón y se podrían diseñar y evaluar estilos arquitectónicos alternativos. Los patrones de las arquitecturas se centran en estructuras e interacciones de grano más grueso, entre subsistemas e incluso entre sistemas.

1.5.1 Arquitecturas basadas en eventos.

Las arquitecturas basadas en eventos, también son conocidas como arquitecturas de invocación implícita, integración reactiva o difusión selectiva. En la actualidad existen estrategias de programación basada en eventos, sobre todo referida a interfaces de usuario, y hay además eventos en los modelos de objetos y componentes, pero no es a eso a lo que se refiere principalmente el estilo, aunque esas consideraciones no están del todo excluidas.

El patrón de diseño que más estrechamente está relacionado con este estilo es el que se conoce con el nombre de Observer, este término se hizo popular a principios de los ochenta en Smalltalk y que en el mundo de Java se le conoce como Modelo de delegación de eventos.

Este tipo de arquitecturas históricamente han estado vinculadas con sistemas basados en actores, daemons y redes de conmutación de paquetes (publicación-suscripción). Los conectores de estos sistemas incluyen procedimientos de llamada tradicionales y vínculos entre anuncios de eventos e invocación de procedimientos. La idea dominante en la invocación implícita es que, en lugar de invocar un procedimiento en forma directa (como se haría en un estilo orientado a objetos) un componente puede anunciar mediante difusión uno o más eventos. Un componente de un sistema puede anunciar su interés en un evento determinado asociando un procedimiento con la manifestación de dicho evento (8).

Un ejemplo clásico en ambientes de Microsoft es el servicio de Notificación de SQL Server, cuando el evento se anuncia, el sistema invoca todos los procedimientos que se han registrado para él, de este modo, el anuncio de un evento implícitamente ocasiona la invocación de determinados procedimientos en otros módulos.

Muchos son los sistemas que utilizan esta arquitectura. Este estilo es utilizado en ambientes de integración de herramientas, en sistemas de gestión de base de datos para asegurar las restricciones de consistencia, en interfaces de usuario para separar la presentación de los datos de los procedimientos que gestionan datos y en editores sintácticamente orientados para proporcionar verificación semántica incremental.

Un ejemplo de este tipo de estilo es C2 o Chiron-2, este tipo de arquitectura está constituida por componentes que se comunican a través de buses y la comunicación entre los componentes está basada en eventos.

Las reglas de Microsoft para el uso del modelo de eventos señalan que este estilo puede utilizarse cuando:

- Se desea manejar independientemente y de forma aislada diversas implementaciones de una “función” específica.
- Las respuestas de una implementación no afectan la forma en que trabajan otras implementaciones.
- Todas las implementaciones son de escritura solamente o de dispararse-y-olvidar, tal que la salida del proceso de negocios no está definida por ninguna implementación, o es definida sólo por una implementación de negocios específica.

Entre las principales ventajas de este estilo arquitectónico se encuentran:

- Se optimiza el mantenimiento haciendo que procesos de negocios que no están relacionados sean independientes.
- Se alienta el desarrollo en paralelo, lo que puede resultar en mejoras de performance.
- Es fácil de empaquetar en una transacción atómica.
- Se puede agregar un componente registrándolo para los eventos del sistema; se pueden reemplazar componentes.

Y como desventajas están que:

- El estilo no permite construir respuestas complejas a funciones de negocios.
- Un componente no puede utilizar los datos o el estado de otro componente para efectuar su tarea.

Cuando un componente anuncia un evento, no tiene idea sobre qué otros componentes están interesados en él, ni el orden en que serán invocados, ni el momento en que finalizan lo que tienen que hacer. Pueden surgir problemas de performance global y de manejo de recursos cuando se comparte un repositorio común para coordinar la interacción.

1.5.2 Arquitecturas orientadas a servicios. (SOA)

Recientemente ha sido que estas arquitecturas llamadas SOA han recibido un tratamiento intensivo en el campo de exploración de los estilos,

Alrededor de los Web services, que dominan el campo de un estilo SOA, se han generado las discusiones que usualmente acompañan a toda idea exitosa, al principio hasta resultaba difícil encontrar una definición aceptable y consensuada. El grupo de tareas de W3C, por ejemplo, demoró un año y medio en ofrecer la primera definición canónica apta para consumo arquitectónico, y que se cita a continuación:

Un Web Service es un sistema de software diseñado para soportar interacción máquina-a-máquina sobre una red. Posee una interfaz descripta en un formato procesable por máquina. Otros sistemas interactúan con el Web Service de una manera prescripta por su descripción utilizando mensajes SOAP, típicamente transportados usando Hypertext Transfer Protocol (HTTP[1]) con una serialización en e**X**tensible **M**arkup **L**anguage (XML[2]) en conjunción con otros estándares relacionados a la Web (8).

En la literatura clásica referida a estilos, las arquitecturas basadas en servicios se ajustan con lo que Garlan & Shaw definen como el estilo de procesos distribuidos. Otros autores hablan de Arquitecturas de Componentes Independientes que se comunican a través de mensajes. Según esta perspectiva, existen dos variantes de este estilo:

- Participantes especificados (named): Estilo de proceso de comunicación. El ejemplar más conocido sería el modelo cliente-servidor. Si el servidor trabaja sincrónicamente, retorna control al cliente junto con los datos; si lo hace asincrónicamente, sólo retorna los datos al cliente, el cual mantiene su propio hilo de control.

[1] Es el protocolo usado en cada transacción de la Web (WWW).

[2] Es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

- Participantes no especificados (unnamed): Paradigma publish/subscribe, o estilo de eventos.

Existen varios textos consagrados a analizar los Web Services basados en XML en términos de arquitectura de software en general y de estilos arquitectónicos en particular

Es fundamental destacar la forma en la cual este estilo redefine los viejos modelos de ORPC propios de las arquitecturas orientadas a objetos y componentes.

La descripción, publicación, descubrimiento, localización e invocación de los Web Services se puede hacer en tiempo de ejecución, de modo que los servicios que interactúan pueden figurarse la forma de operar de sus contrapartes, sin haber sido diseñados específicamente caso por caso.

Desde el punto de vista arquitectónico, se puede hacer ahora una breve caracterización de este estilo:

- Un servicio es una entidad de software que encapsula funcionalidad de negocios y proporciona dicha funcionalidad a otras entidades a través de interfaces públicas bien definidas.

- Los componentes del estilo (o sea los servicios) están débilmente acoplados. El servicio puede recibir requerimientos de cualquier origen. La funcionalidad del servicio se puede ampliar o modificar sin rendir cuentas a quienes lo requieran.

- Los componentes que requieran un servicio pueden descubrirlo y utilizarlo dinámicamente mediante UDDI y sus estándares sucesores. En general (aunque hay alternativas) no se mantiene persistencia de estado y tampoco se pretende que un servicio recuerde nada entre un requerimiento y el siguiente.

Un servicio puede incluir muchas interfaces y poseer propiedades tales como descripción de protocolos, puntos de entrada y características del servicio mismo. Algunas de estas notaciones son provistas por lenguajes declarativos basados en XML, como WSDL (Web Service Description Language).

En el futuro próximo, el modelo de programación de Longhorn permitirá agregar a la ya extensa funcionalidad de los servicios una nueva concepción relacional del sistema de archivos, soporte extensivo de shell y prácticamente toda la riqueza del API de Win32 en términos de código manejado (8).

1.5.3 Arquitecturas basadas en recursos.

La literatura especializada tiende a considerar a Representational State Transfer o REST una variante menor de las arquitecturas basadas en servicios, Fielding considera que REST resulta de la composición de varios estilos más básicos, incluyendo repositorio replicado, cache, cliente-servidor, sistema en capas, sistema sin estado, máquina virtual, código a demanda e interfaz uniforme (8). REST define recursos identificables y métodos para acceder y manipular el estado de esos recursos, un ejemplo lo constituye la World Wide Web, donde las urls identifican los recursos y HTTP es el protocolo de acceso. El argumento de Fielding es que HTTP mismo, con su conjunto mínimo de métodos y su semántica simplísima, es suficientemente general para modelar cualquier dominio de aplicación, de esta manera, el modelado tradicional orientado a objetos se hace innecesario y es reemplazado por el modelado de entidades tales como familias jerárquicas de recursos abstractos con una interfaz común y una semántica definida por el propio HTTP.

1.5.4 Arquitecturas en capas.

Los sistemas o arquitecturas en capas constituyen uno de los estilos que aparecen con mayor frecuencia mencionados en las diferentes literaturas. El estilo en capas constituye una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior, instrumentando una vieja organización estratigráfica que se remonta a las concepciones formuladas por Edsger Dijkstra en la década de 1960. En la práctica, las capas suelen ser entidades complejas, compuestas por varios paquetes o subsistemas. Los patrones de uso común relativos a este estilo son: Facade, Adapter, Bridge y Strategy. Dentro de las principales restricciones de este estilo se encuentran algunas limitaciones que exigen a cada capa operar solo con capaz adyacentes y que los elementos de una capa solo pueden entenderse con otros elementos de la propia capa, se supone que si esta exigencia se debilita el estilo deja de ser puro y pierde parte de su capacidad heurística, también se pierde la posibilidad de reemplazar una capa sin que se afecten las demás, disminuye la flexibilidad del conjunto y se complica su mantenimiento. Algunos ejemplos de este estilo son muchos de los protocolos de comunicación en capas, el más característico es el modelo OSI con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación, aunque este estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos, así como en las especificaciones relacionadas con XML.

El número mínimo de capas es dos y es en ese umbral donde la literatura arquitectónica sitúa a veces al sub-estilo cliente-servidor como el modelo arquetípico del estilo en capas y el que se encuentra con mayor frecuencia en las aplicaciones en red.

Las ventajas del estilo en capas son obvias:

- Soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los implementadores la partición de un problema complejo en una secuencia de pasos incrementales.
- Admite optimizaciones y refinamientos.
- Proporciona amplia reutilización. Al igual que los tipos de datos abstractos, se pueden utilizar diferentes implementaciones o versiones de una misma capa en la medida que soporten las mismas interfaces de cara a las capas adyacentes, esto conduce a la posibilidad de definir interfaces de capa estándar a partir de las cuales se pueden construir extensiones o prestaciones específicas.

Dentro de las desventajas que tiene este estilo se encuentra:

- Muchos problemas no admiten un buen mapeo en una estructura jerárquica. Incluso cuando un sistema se puede establecer lógicamente en capas, consideraciones de performance pueden requerir acoplamientos específicos entre capas de alto y bajo nivel.
- A veces es extremadamente difícil encontrar el nivel de abstracción correcto; por ejemplo, la comunidad de comunicación ha encontrado complejo mapear los protocolos existentes en el framework ISO.
- Los cambios en las capas de bajo nivel tienden a filtrarse hacia las de alto nivel, en especial si se utiliza una modalidad relajada.
- La arquitectura en capas ayuda a controlar y encapsular aplicaciones complejas, pero complica no siempre razonablemente las aplicaciones simples.

A pesar de que los documentos de arquitectura de Microsoft que en algún momento impulsaron con fuerza el diseño en capas y en los últimos años se han inclinado hacia las arquitecturas basadas en servicios; la separación y especialización de interfaces de usuario, reglas de negocios y estructuras de datos (o sus equivalentes arquitectónicos en otros dominios) conservan todavía plena relevancia. Se han elaborado numerosos patrones de arquitectura y diseño derivados de este estilo, como Layered Application y Three-layered Services Application.

A pesar de la versatilidad de REST o de SOAP, las arquitecturas en capas distan de ser un estilo viejo y anticuado.

1.5.5 Arquitecturas basadas en componentes.

Los sistemas de software basados en componentes se basan en principios definidos por una ingeniería de software específica, en un principio, hacia 1994, se planteaba como una modalidad que extendía o superaba la tecnología de objetos.

Dentro de las tantas definiciones que se han dado de componentes se encuentra la de Clemens, Alden, Szyperski; que proporciona una que es bastante operativa:

Un componente de software, es una unidad de composición con interfaces especificadas contractualmente y dependencias del contexto explícitas.

El hecho que sea una unidad de composición y no de construcción quiere decir que no es preciso confeccionarla: se puede comprar hecha, o se puede producir para que otras aplicaciones la utilicen en sus propias composiciones.

También se puede definir un componente como un artefacto diseñado y desarrollado de acuerdo ya sea con CORBA Component Model (CCM), JavaBeans y Enterprise JavaBeans en J2EE.

En un estilo como este:

- Los componentes son las unidades de modelado, diseño e implementación.
- Las interfaces están separadas de las implementaciones, y las interfaces y sus interacciones son el centro de incumbencias en el diseño arquitectónico.

En cuanto a las restricciones de este estilo, puede admitirse que una interfaz sea implementada por múltiples componentes. Usualmente, los estados de un componente no son accesibles desde el exterior. Que los componentes sean locales o distribuidos es transparente en la tecnología actual.

La evaluación dominante del estilo de componentes subraya su mayor versatilidad respecto del modelo de objetos, pero también su menor adaptabilidad comparado con el estilo orientado a servicios.

1.5.6 Arquitecturas orientadas a objetos.

Nombres alternativos para este estilo han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este estilo son los objetos, o más bien instancias de los tipos de dato abstractos.

Para David Garlan y Mary Shaw (8), los objetos representan una clase de componentes que ellos denominan managers, debido a que son responsables de preservar la integridad de su propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos.

Las características fundamentales de las arquitecturas Orientadas a Objetos son:

- Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.
- Las interfaces están separadas de las implementaciones. En general la distribución de objetos es transparente, y en el estado de arte de la tecnología apenas importa si los objetos son locales o remotos. El mejor ejemplo de OO para sistemas distribuidos es Common Object Request Broker Architecture (CORBA), en la cual las interfaces se definen mediante Interface Description Language (IDL); un Object Request Broker media las interacciones entre objetos clientes y objetos servidores en ambientes distribuidos.

En cuanto a las restricciones del estilo, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases. Los objetos interactúan a través de invocaciones de funciones y procedimientos.

Ventajas de las arquitecturas Orientadas a Objetos:

- Se puede modificar la implementación de un objeto sin afectar a sus clientes.
- Es posible descomponer problemas en colecciones de agentes en interacción.
- Un objeto es ante todo una entidad reutilizable en el entorno de desarrollo.

Desventajas:

- Para poder interactuar con otro objeto a través de una invocación de procedimiento, se debe conocer su identidad.

- Se presentan problemas de efectos colaterales en cascada: si A usa B y C también lo usa, el efecto de C sobre B puede afectar a A.

En los estudios arquitectónicos de estilos, y posiblemente por efecto de un énfasis que es más basado en las relaciones que en las cosas que componen un sistema, el modelo de objetos aparece relativamente subordinado en relación con la importancia que ha tenido en otros ámbitos de la ingeniería de software, en los que la orientación a objeto ha sido y sigue siendo dominante. Pero entre los estilos, se lo sitúa al lado de las arquitecturas basadas en componentes y las orientadas a servicios, ya sea en paridad de condiciones o como precedente histórico (8).

1.6 Tendencias y tecnologías actuales

La tendencia actual en el mundo del software lleva a la construcción de sistemas mucho más grandes y complejos, todo esto debido a que los ordenadores son cada día más potentes y también al uso creciente de Internet, cuando nos enfrentamos a proyectos de gran envergadura se hace necesaria la utilización de una metodología que guie el proceso de desarrollo, dirija las tareas de cada desarrollador por separado y del equipo como un todo y especifique los artefactos que deben desarrollarse a lo largo del desarrollo del sistema.

Dentro de las principales metodologías de desarrollo de software se encuentran: XP, MSF y RUP, etc, cada una con sus ventajas y desventajas y características especificas que le permiten ser aplicadas en distintos entornos de desarrollo.

1.6.1 Metodología eXtreme Programming (XP)

Es una de las metodologías de desarrollo de software más exitosas en la actualidad, es utilizada en proyectos de corto plazo fundamentalmente, consiste en una programación rápida o extrema, cuya particularidad es tener como parte del equipo, al usuario final, pues es uno de los requisitos para llegar al éxito del proyecto.

La metodología XP propone:

- Empieza en pequeño y añade funcionalidad con retroalimentación continua.
- El manejo del cambio se convierte en parte sustantiva del proceso.
- El costo del cambio no depende de la fase o etapa.
- No introduce funcionalidades antes que sean necesarias.

- El cliente o el usuario se convierte en miembro del equipo.

1.6.2 Microsoft Solution Framework (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas.

Características de la metodología MSF:

- Adaptable**:** es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un lugar específico.
- Escalable**:** puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- Flexible**:** es utilizada en el ambiente de desarrollo de cualquier cliente.

Tecnología Agnóstica: porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

1.6.3 Rational Unified Process (RUP)

El Proceso Unificado es un Proceso de Desarrollo de Software que contiene un conjunto de actividades necesarias para transformar los requisitos de un usuario en un sistema de software. Más que un simple proceso; es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas software, para diferentes áreas de aplicación, tipos de organizaciones, niveles de actitud y tamaños de proyecto. Está basado en componentes, lo cual quiere decir que el sistema software en construcción está formado por componentes software interconectados a través de interfaces bien definidas. Utiliza el *Lenguaje Unificado de Modelado* (Unified Modeling Languaje, UML) para preparar todos los esquemas de un sistema software.

1.6.3.1 Características del Proceso Unificado

Los verdaderos aspectos definitorios del Proceso Unificado, y que lo convierten en único, se resumen en tres frases clave - dirigido por CU, centrado en la arquitectura, e iterativo e incremental.

Dirigido por los casos de uso:

Teniendo en cuenta que la razón de ser de un sistema es brindar servicios a los usuarios, RUP define caso de uso como el conjunto de acciones que debe realizar un sistema para dar un resultado de valor a un determinado usuario y los utiliza tanto para especificar los requisitos funcionales del sistema, como para guiar todos los demás pasos de su desarrollo, dígase diseño, implementación y prueba.

Centrado en la arquitectura:

La arquitectura es una vista del diseño completo con las características más importantes, dejando a un lado los detalles. Esta no sólo incluye las necesidades de los usuarios e inversores, sino también otros aspectos técnicos como el hardware, sistema operativo, sistema de gestión de base de datos, protocolos de red; con los que debe coexistir el sistema. En otras palabras, la arquitectura representa la forma del sistema, la cual va madurando en su interacción con los CU hasta llegar a un equilibrio entre funcionalidad y características técnicas.

Iterativo e incremental:

La alta complejidad de los sistemas actuales hace que sea factible dividir el proceso de desarrollo en varios mini-proyectos. Cada uno de estos mini-proyecto se les denomina iteración y pueden o no representar un incremento en el grado de terminación del producto completo. En cada iteración los desarrolladores seleccionan un grupo de CU, los cuales se diseñan, implementan y prueban. La planificación de iteraciones hace que se reduzcan los riesgos de los costes de un solo incremento, mantener la motivación del equipo pues puede ver avances claros a corto plazo y que el desarrollo pueda adaptarse a los cambios en los requisitos.

El desarrollo del software según RUP se divide en 4 fases obteniendo al final de cada una un objetivo específico.

- **Inicio**: Determinar la visión del proyecto.
- **Elaboración**: Definir la arquitectura óptima para el sistema.
- **Construcción**: Obtener una capacidad operacional inicial.
- **Transición**: Obtener el release del proyecto.

Una de las principales ventajas de esta metodología es que, en cada ciclo de iteración, se hace exigente el uso de artefactos, siendo por este motivo, una de las metodologías más importantes para alcanzar un grado de certificación en el desarrollo del software.

1.6.3.2 Notación UML

UML (Unified Modeling Language, Lenguaje Unificado de Modelación)

Es un lenguaje de modelado visual que se usa para especificar, visualizar, construir y documentar artefactos del sistema de un software. Se usa para entender, diseñar, configurar, mantener y controlar la información sobre los sistemas a construir.

UML capta la información sobre la estructura estática y el comportamiento dinámico de un sistema. Un sistema se modela como una colección de objetos discretos que interactúan para realizar un trabajo que finalmente beneficia a un usuario externo. El lenguaje de modelado pretende unificar la experiencia pasada sobre técnicas de modelado e incorporar las mejores prácticas actuales en un acercamiento estándar.

Además tiene las siguientes características:

- Permite modelar sistemas utilizando técnicas orientadas a objetos.
- 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.

UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los CU, centrado en la arquitectura, interactivo e incremental, como RUP.

1.6.4 Fundamentación de la Plataforma de Desarrollo y Lenguaje de programación, Gestor de Base de Datos y otras herramientas.

En un proceso de desarrollo de software las herramientas son esenciales en el soporte del mismo, es casi imposible desarrollar software sin utilizar un proceso soportado por herramientas, ya que estas ayudan a automatizar procesos repetitivos, mantener las cosas estructuradas, gestionar grandes cantidades de información y para guiarnos a lo largo de un camino de desarrollo (6).

Un proceso con poco soporte por herramientas debe sostenerse sobre gran cantidad de trabajo manual y será por tanto menos formal. Sin un soporte por herramientas que automaticen la consistencia a lo largo del ciclo de vida, será difícil mantener actualizados los modelos y la implementación y el ciclo iterativo e incremental será más difícil, todo esto puede disminuir la productividad del equipo, se tendrían que hacer manualmente todas las comprobaciones y habrá fisuras en los artefactos desarrollados, además de que todo esto trae consigo más tiempo de desarrollo. (6).

Kazman (1996) plantea que la necesidad del diseño y el análisis de las arquitecturas de software ha llevado al deseo de la creación de herramientas CASE para soportar el proceso de desarrollo

Para el desarrollo del sistema se realizó un estudio de las posibles herramientas a utilizar en su construcción. Teniéndose en cuenta la tendencia actual y las novedades de cada una de ellas. Las herramientas utilizadas son:

1.6.4.1 Visual Studio .NET 2005 y C#.

En esta sección se trata modestamente de justificar el entorno de desarrollo y lenguaje de programación que se utilizó para la implementación de este sistema. Para esto se tuvieron en cuenta varios aspectos que se mencionan a continuación:

- No existió por parte del cliente un requerimiento sobre la definición de una plataforma específica.
- La necesidad de una plataforma de trabajo productiva con una biblioteca de clases, que incluyera componentes para el trabajo con hilos y componentes de conexión a internet, así como determinadas estructuras de datos básicos para la construcción del sistema.
- La necesidad de un lenguaje dentro de la plataforma que soportara los conceptos de POO necesarios, para el desarrollo de la misma.

- El conocimiento sobre dicha tecnología o plataforma que se escogiera, para ahorrar tiempo de estudio y capacitación.

Por todas estas razones se seleccionó la Plataforma .NET y Lenguaje de programación C# para el desarrollo del sistema cumpliendo con los aspectos mencionados anteriormente, y específicamente el lenguaje C#, dentro de la plataforma por el conocimiento del mismo, tener las estructuras de programación necesarias para poder implementar lo planteado en la fase de análisis y diseño de este sistema, como son las interfaces de implementación, los eventos y delegados, listas genéricas, entre otros.

Visual Studio 2005, es la suite de herramientas de Microsoft para crear e implementar software seguro y eficaz para la plataforma Microsoft .NET. Creado para satisfacer las necesidades de desarrollo de software más exigentes, Visual Studio 2005 mejora y optimiza a su predecesor, y se convierte en la herramienta más productiva para el diseño, desarrollo, depuración y despliegue de aplicaciones para Windows, Web y dispositivos móviles

C# es un lenguaje que reúne las mejores características de los principales lenguajes de programación existentes hasta el momento. Algunas particularidades de C# lo convierten en un lenguaje fácilmente legible, sencillo, orientado a objetos e intuitivo. Características como modernidad, portabilidad del código, orientación a componentes y gestión automática de memoria constituyen eslabones importantes a la hora de utilizar este lenguaje de programación.

1.6.4.2 Framework .NET 2.0

La arquitectura .NET (.NET Framework) es el modelo de programación de la plataforma .NET para construir y ejecutar los servicios .NET. El objetivo de esta arquitectura es la de reducir la complejidad en el desarrollo de este tipo de aplicaciones, permitiendo a los desarrolladores centrarse en escribir la lógica específica del servicio a desarrollar.

El Framework .Net es una infraestructura sobre la que se reúne todo un conjunto de lenguajes y servicios que simplifican enormemente el desarrollo de aplicaciones. Mediante esta herramienta se ofrece un entorno de ejecución altamente distribuido, que permite crear aplicaciones robustas y escalables. Los principales componentes de este entorno son:

- Lenguajes de compilación
- Biblioteca de clases de .Net
- CLR (Common Language Runtime).

1.6.4.3 SQL Server 2000

En la actualidad, las compañías demandan una clase diferente de solución de base de datos. El rendimiento, la escalabilidad y la confiabilidad son esenciales, aparte de estas cualidades fundamentales, SQL Server 2000 proporciona agilidad a sus operaciones de análisis y administración de datos al permitir a las organizaciones adaptarse rápida y fácilmente para obtener ventaja competitiva en un entorno de cambios constantes.

SQL Server 2000 es un potente motor de bases de datos de alto rendimiento capaz de soportar millones de registros por tabla con una interfase intuitiva y con herramientas de desarrollo integradas como Visual Studio 6.0 o .NET, además incorpora un modelo de objetos totalmente programable (SQL-DMO) con el que podemos desarrollar cualquier aplicación que manipule componentes de SQL Server, es decir, hacer aplicación para crear bases de datos, tablas, DTS, backups, etc., todo lo que se puede hacer desde el administrador del SQL Server y podemos hacerlo no solo en Visual C++ sino también en Visual Basic, ASP y por supuesto en .NET.

SQL Server 2000 ha obtenido importantes galardones en pruebas de referencia por su escalabilidad y velocidad.

1.6.4.4 Data Access Application Blocks (DAAB)

Para el manejo de la persistencia de los datos se propone el Data Access Application Block, componente de Microsoft que contiene código de acceso a datos optimizado y provee una forma simple, ordenada y limpia de acceder a los datos del Servidor SQL. DAAB ayuda a llamar los procedimientos almacenados y ejecutar comandos de texto SQL. Entre sus funciones está devolver los objetos SqlDataReader, DataSet, XmlReader o valores únicos (scalar).

1.6.4.5 Subversion 1.3.0

Software de sistema de control de versiones[3], que ayuda a que los desarrolladores tengan un seguimiento de la gestión de cambios y configuraciones durante el proceso de desarrollo de un producto, como por ejemplo controlar todos los cambios que se efectúan en los ficheros de código fuente de producto, etc.

[3] Es la gestión de versiones (revisiones) de todos los elementos de configuración que forman la línea base de un producto o una configuración del mismo.

1.6.4.6 Rational Rose Enterprise Edition 2003

Es una herramienta para “modelado visual”, que forma parte de un conjunto más amplio de herramientas que juntas cubren todo el ciclo de vida del desarrollo de software. Permite completar una gran parte de las disciplinas (flujos fundamentales) del Proceso Unificado de Rational (RUP) e incluye un conjunto de herramientas de ingeniería inversa y generación de código que allanan el camino hasta el producto final.

Es una herramienta con plataforma independiente que ayuda a la comunicación entre los miembros del equipo, a monitorear el tiempo de desarrollo y a entender el entorno de los sistemas. Una de las grandes ventajas de Rose es que utiliza la notación estándar en la arquitectura de software (UML), la cual permite a los arquitectos de software y desarrolladores visualizar el sistema completo utilizando un lenguaje común, además los diseñadores pueden modelar sus componentes e interfaces en forma individual y luego unirlos con otros componentes del proyecto.

1.6.5 Fundamentación de la metodología utilizada.

En el desarrollo del sistema “Predictor” se hizo uso de las herramientas de la metodología RUP (*Rational Unified Process*) para facilitar el desarrollo del sistema, y la notación UML, debido al alcance que se espera del producto, la necesidad de tener la documentación bien detallada de cada uno de los artefactos que se generen y para facilitar el futuro desarrollo y posible ampliación del sistema con nuevos requerimientos. Un factor importante que influyó en la decisión por esta metodología fue el previo conocimiento que poseían los integrantes del equipo de desarrollo y demás personas que colaboraron en el diseño e implementación del sistema. Aunque el proyecto no es complejo, el tiempo de desarrollo ha sido a largo plazo ya que se han ido agregando requisitos por parte de los usuarios finales, manteniendo al sistema en constante cambio, por lo que se puede concluir que esta metodología es la que más se asemeja para el desarrollo de proyectos de este tipo.

1.7 Métricas de diseño arquitectónico.

Las métricas de diseño de alto nivel se concentran en las características de la arquitectura del programa, con énfasis en la estructura arquitectónica y en la eficiencia de los módulos. Estas métricas son de caja negra en el sentido que no requieren ningún conocimiento del trabajo interno de un módulo en particular del sistema.

1.7.1 Métrica de Card y Glass

Card y Glass definieron tres medidas de la complejidad de diseño del software, referenciadas por Pressman (4) la cual se basa en la complejidad estructural, complejidad de datos y complejidad del sistema:

1. La *complejidad estructural*, S(i), de un módulo i se define de la siguiente manera: donde $S(i) = f_{out}^2(i)$ donde $f_{out}(i)$ es la expansión[4] del módulo i.

2. La *complejidad de datos*, D(i) proporciona una indicación de la complejidad en la interfaz interna de un módulo i y se define como: $D(i) = \frac{v(i)}{[f_{out}(i)+1]}$ donde v(i) es el número de variables de entrada y salida que entran y salen del módulo i.

3. La *complejidad del sistema*, C(i), se define como la suma de las complejidades estructural y de datos, y se define como: $C(i) = S(i) + D(i)$

En materia de evaluación a medida que crecen los valores de complejidad, la complejidad arquitectónica o global del sistema también aumenta.

1.7.2 Métrica de Henry y Kafura

Henry y Kafura definen una métrica de complejidad que emplea la expansión y la concentración y se expresa con la siguiente fórmula:

$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2$$

Donde la $longitud(i)$ es el número de sentencias en lenguaje de programación para el módulo i y $f_{in}(i)$ la concentración del módulo i.

La concentración y la expansión no sólo incluyen el número de conexiones de control del módulo (llamadas al módulo), sino que también incluyen el número de módulos del que el módulo i recoge (concentración) o actualiza (expansión) datos.

[4] La expansión $f_{out}^2(i)$ indica el número de módulos inmediatamente subordinados al módulo i, es decir, el número de módulos que son invocados directamente por el módulo i.

1.8 Conclusiones

El uso de las tecnologías para el desarrollo de sistemas de gestión depende en gran medida del tipo de sistema que se desea desarrollar y de la entidad a la que está destinada. La selección de la tecnología a utilizar depende de un estudio de los requerimientos del sistema a desarrollar.

Después de haber hecho un análisis profundo en este capítulo acerca de la problemática, de haber planteado el objetivo general y los específicos, de hacer un estudio de algunos software existentes relacionados con la propuesta, además de hacer una valoración acerca de las herramientas más recomendadas para el desarrollo de un sistema como el que se propone, se concluye que el mismo será desarrollado con el gestor de base de datos SQL Server 2000, el entorno de desarrollo Visual Studio .NET 2005 con C# como lenguaje y el Framework .NET 2.0, se utilizará además la metodología RUP con UML como lenguaje de modelado, además del Subversion 1.3.0 para la gestión de cambios y configuraciones.

2 DESCRIPCIÓN Y ANÁLISIS DE LA SOLUCIÓN PROPUESTA.

2.1 Introducción

Las necesidades actuales que tiene toda organización para el logro de sus objetivos, demandan la construcción de grandes y complejos sistemas de software que requieren de la combinación de diferentes tecnologías y plataformas de hardware y software para alcanzar un funcionamiento acorde con dichas necesidades. Lo anterior, exige de los profesionales dedicados al desarrollo de software poner especial atención y cuidado al diseño de la arquitectura, bajo la cual estará soportado el funcionamiento de sus sistemas (9).

La arquitectura de software constituye un conjunto de decisiones significativas acerca de la organización de un sistema, incluye la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía esta organización. La arquitectura del software no sólo se interesa por la estructura y el comportamiento, sino también por las restricciones y compromisos de uso, funcionalidad, funcionamiento, flexibilidad al cambio, reutilización, comprensión, economía y tecnología, así como por aspectos estéticos (4).

El desarrollo de la arquitectura de software es una de las etapas fundamentales y, en muchos casos, la más importante en el desarrollo de software, pues es aquí donde los profesionales aportan todos sus conocimientos, creatividad y experiencia para crear la mejor propuesta de solución que se dará al cliente que cumpla con los requerimientos funcionales y no funcionales establecidos para el sistema en desarrollo, así como sus preocupaciones principales de lo que esperan del sistema (9).

El desarrollo de arquitecturas de software ayuda en la:

- Comprensión del sistema.
- Organización del desarrollo.
- Fomenta la reutilización.
- Evolución del sistema.

En este capítulo se presentan algunos conceptos relacionados con la arquitectura del software; se define la línea base de la arquitectura para el sistema “Predictor” y se presenta la descripción de la arquitectura mediante las 4 + 1 vistas arquitectónicas definidas por RUP, se justifica la utilización de los diferentes patrones tanto de diseño como el patrón arquitectónico en capas y se define la estrategia de comunicación entre los diferentes componentes del sistema.

2.2 Los requisitos en la arquitectura.

Los CU juegan un papel fundamental en la concepción de la arquitectura de un sistema, en las primeras etapas de desarrollo de un software el esfuerzo es mayor en la captura de los requisitos y en la comprensión de lo que quiere el cliente. Diseñar la arquitectura basándose en los requisitos garantiza que ésta sea funcional. Los CU son formas de expresar los requisitos funcionales del sistema que se quiere desarrollar. Cuando se concibe la arquitectura se toman los CU más significativos y a partir de ellos se define un sistema que soporte dichas funcionalidades. En esta sección se darán a conocer los requisitos funcionales del sistema “Predictor” y en el epígrafe 2.3 se describirán los CU arquitectónicamente significativos. Además se listarán los requisitos no funcionales, que también juegan un papel fundamental en la arquitectura.

2.2.1 Requisitos Funcionales

El Sistema debe ser capaz de:

1. R1 - Controlar la información a la que accede cada usuario.
2. R2 - Configurar la conexión a Internet.
3. R3 - Hacer búsquedas en cualquiera de las bases de datos disponibles en Internet que contengan información sobre patentes.
4. R4 - Realizar búsqueda básica.
5. R5 - Realizar búsqueda avanzada.
6. R6 - Descargar la información acorde con el criterio de búsqueda que emplee el usuario.
7. R7- Realizar el procesamiento de la información que se haya encontrado en las bases de datos de Internet.

8. R8 - Guardar la información obtenida de la descarga, en el directorio seleccionado por el usuario.
9. R9 - Permitir al administrador del sistema registrar los datos de los usuarios.
10. R10 - Permitir al administrador del sistema actualizar los datos de los usuarios.
11. R11 - Permitir al administrador del sistema eliminar los datos de los usuarios.
12. R12 - Permitir al administrador del sistema adicionar a la aplicación una nueva base de datos de Internet.
13. R13- Permitir al administrador del sistema actualizar la información de las bases de datos.
14. R14 - Permitir al administrador del sistema eliminar la base de datos de Internet que elija.

2.2.2 Requisitos no Funcionales

Requerimientos no funcionales de Diseño e implementación.

1. En la implementación del sistema se utilizará un estándar de codificación.
2. Se utilizarán diferentes herramientas para desarrollo del sistema como son: Rational Rose 2003, MS SQL Server 2000, Visual Studio.NET 2005
3. Las bibliotecas de clases que se emplearán serán: The Base Class Libraries (BCL) y msdn2.microsoft.com/en-us/netframework/aa569603.aspx.

Requerimientos no funcionales de Software

1. El sistema operativo sobre el que se trabajará será Windows XP o superior.
2. En la PC donde se encuentre el Servidor de Bases de Datos debe tener instalado el Microsoft SQL Server 2000.
3. Las PCs donde se ejecute el Sistema debe tener instalado el Framework .NET 2.0

Requerimientos no funcionales de Hardware

1. Es necesario que se cuente con una red local donde existan al menos dos computadoras. En una se ejecutará el Sistema y en la otra se encontrará el Servidor de Base de Datos por razones de seguridad.

2. En la computadora donde esté el Sistema debe haber conexión a Internet.

3. Es recomendado para el Sistema Operativo una PC PENTIUM III o superior con 300 MHz como mínimo, con 128 MB de RAM, y mínimo 20 GB de disco duro.

4. La comunicación entre las máquinas donde estará el sistema y el Servidor de Base de Datos será a través del protocolo ADO.

Requerimientos no funcionales de Portabilidad

1. El software podrá ser usado sólo bajo los sistemas operativos Windows XP o superior.

Requerimientos no funcionales de Seguridad.

1. El sistema deberá tener el control de la información a la que accede cada usuario, no se le denegará el acceso a la aplicación a ningún usuario.

Requerimientos no funcionales de Rendimiento

1. Rápido acceso de búsqueda a la información en tiempos relativamente cortos, dependiendo de la conexión a Internet.

Requerimientos no funcionales de Soporte

1. Para garantizar el soporte a los clientes de la Consultoría Delfos, existirá la posibilidad de que ellos emitan sus quejas y sugerencias a los desarrolladores del Sistema, ya sea por correo electrónico o por cualquier otra vía de comunicación por un período de 1 mes.

Requerimientos no funcionales de Usabilidad

1. El Sistema contará con un Manual de Usuario que servirá para guiar a los especialistas en el manejo del Sistema.

Requerimientos Adicionales

El usuario podrá hacer cambios en cuanto a los colores de la interfaz configurando ésta a su gusto.

2.3 Línea Base de la Arquitectura.

Al finalizar la fase de elaboración se deben tener desarrollados los modelos del sistema que representan los CU más importantes y su realización desde el punto de vista de la arquitectura,

además ya se tiene que tener decidido los estándares con qué se cuentan, qué software del sistema, qué middleware[5] se puede utilizar, que sistemas heredados se pueden reutilizar y qué necesidades de distribución se tienen, se tiene también una primera versión de los modelos de CU, de análisis, de diseño, de implementación y despliegue; a todo este compendio de modelos se le denomina *línea base de la arquitectura* (4).

La línea base de la arquitectura es un conjunto de artefactos revisados y aprobados que representa un punto de acuerdo para la posterior evolución y desarrollo, y solamente puede ser modificada a través de un procedimiento formal, como la gestión de cambios y configuraciones.

La arquitectura de un software está condicionada por los CU que se definan para soportar el sistema y estos a su vez se convierten en directores de la arquitectura que se va a definir, es por esto que la correcta definición de los CU y determinar aquellos que serán importantes desde el punto de vista arquitectónico permite tener una visión del tipo de arquitectura que se desea utilizar, además de poder definir aspectos tales como el lenguaje de programación que se utilizará, la metodología, etc.

La arquitectura se desarrolla mediante iteraciones, fundamentalmente durante la fase de elaboración, cada iteración comienza con los requisitos, análisis, diseño, implementación y pruebas, pero siempre centrados en los CU arquitectónicamente significativos y en otros requisitos no funcionales.

Conocidos los requisitos funcionales del sistema (Ver epígrafe 2.2.1) se pueden definir los CU arquitectónicamente significativos. A continuación se hace una breve descripción de cada uno de ellos para que quede clara su importancia.

2.3.1 Caso de uso Configuración de la Conexión a Internet

Tiene la responsabilidad de configurar el tipo de conexión que se va a seguir para realizar las descargas, guardando si se usa o no proxy. En caso de usarse, se registra el número IP, el puerto, usuario y contraseña. Toda esta información es guardada en un fichero que es cargado en el momento de la descarga para verificar si la conexión es correcta. Como la información sobre las

[5] Capa intermedia que ofrece bloques de construcción reutilizables (paquetes o subsistemas) a marcos de trabajo y servicios independientes de la plataforma, para cosas como computación con objetos distribuidos, o interoperabilidad en entornos heterogéneos.

patentes está publicada en Internet, es por esta razón que este CU es considerado crítico en el sistema.

2.3.2 Caso de uso Búsqueda de Patentes

Este CU es el encargado de realizar la búsqueda de las patentes en bases de datos publicadas en Internet, este tipo de búsqueda puede ser Básica o Avanzada, a continuación se explica mejor la funcionalidad de cada una de ellas.

2.3.3 Caso de uso Búsqueda Básica

La función de este CU es buscar y descargar las patentes de las bases de datos seleccionadas según un criterio simple, que puede estar en el título o el resumen de las patentes. Después de estar guardadas en un directorio seleccionado por el usuario, automáticamente se realiza el análisis de cada uno de los ficheros para confeccionar los listados de patentes encontradas y crear los ficheros Procite.txt y Excel.txt. Este CU junto con otros que se analizarán posteriormente constituye la razón de ser del sistema.

2.3.4 Caso de uso Búsqueda Avanzada

Este CU es el encargado de buscar y descargar información de la base de datos seleccionada según los criterios de la búsqueda. Se diferencia de la búsqueda básica en que al formar las urls se le pasan más parámetros. Esta búsqueda es más específica que la búsqueda simple, pues se tienen en cuenta la fecha de publicación, el solicitante y el criterio de búsqueda.

2.3.5 Caso de uso Procesamiento de Patentes

Este CU es sumamente importante, debido a que genera los ficheros finales con los cuáles los especialistas realizan los análisis de la información descargada. Se puede inicializar de dos maneras, una por el especialista, especificando la ruta donde está la información que desea procesar; y otra por el mismo sistema cuando termina de descargar la información, que automáticamente la procesa. Es aquí donde se lleva a cabo el proceso de búsqueda y selección de los campos de las patentes con los ficheros que contienen el código *HTML* de las páginas bajadas de Internet, para así conformar el listado de las patentes.

2.4 Descripción de la Arquitectura

Un aspecto importante dentro de la arquitectura de software es que debe describirse a través de las diferentes vistas definidas por RUP, que representan las perspectivas del diseño; según la notación UML las 4 + 1 vistas fundamentales son: la vista de CU, como la perciben los usuarios, analistas y encargados de las pruebas; la vista del análisis; la vista del diseño que comprende las clases, interfaces y colaboraciones que forman el vocabulario del problema y su solución; la vista de implementación que incluye los componentes y archivos sobre el sistema físico; y la vista de despliegue que comprende los nodos que forman la topología de hardware sobre la que se ejecuta el sistema.

Según Rumbaugh, Jacobson y Booch (6) la *descripción de la arquitectura*:

> Se desarrolla habitualmente de forma concurrente, a menudo incluso antes que las actividades que obtienen las versiones del modelo que son parte de la línea base de la arquitectura. El papel de la descripción de la arquitectura es guiar al equipo de desarrollo a través del ciclo de vida del sistema, no sólo por las iteraciones del ciclo actual, sino por todos los ciclos que vengan.

La descripción de la arquitectura es importante y a la vez necesaria en el desarrollo de un software:

- Constituye una expresión del sistema y de su evolución.
- Permite una mejor comunicación entre los actores.
- Permite la evaluación y comparación de arquitecturas de una manera consistente.
- Facilita la planificación, gestión y ejecución de las actividades del desarrollo del sistema.
- Es una forma de verificar la correcta implementación de acuerdo con la descripción arquitectural.

A continuación se presentan las vistas arquitectónicas de los diferentes modelos del sistema, las cuales representan los CU, realizaciones de CU, paquetes del análisis, subsistemas de diseño y nodos que son significativos desde el punto de vista de la arquitectura.

2.4.1 Vista arquitectónica del modelo de casos de uso

La vista de la arquitectura del modelo de CU representa los actores y CU (o escenarios de esos CU) más importantes desde el punto de vista de la arquitectura, o sea aquellos que describen alguna funcionalidad crítica o impliquen algún requisito funcional importante que deba desarrollarse pronto dentro del ciclo de vida del software. La siguiente figura muestra la vista arquitectónica del modelo de CU del sistema “Predictor”.

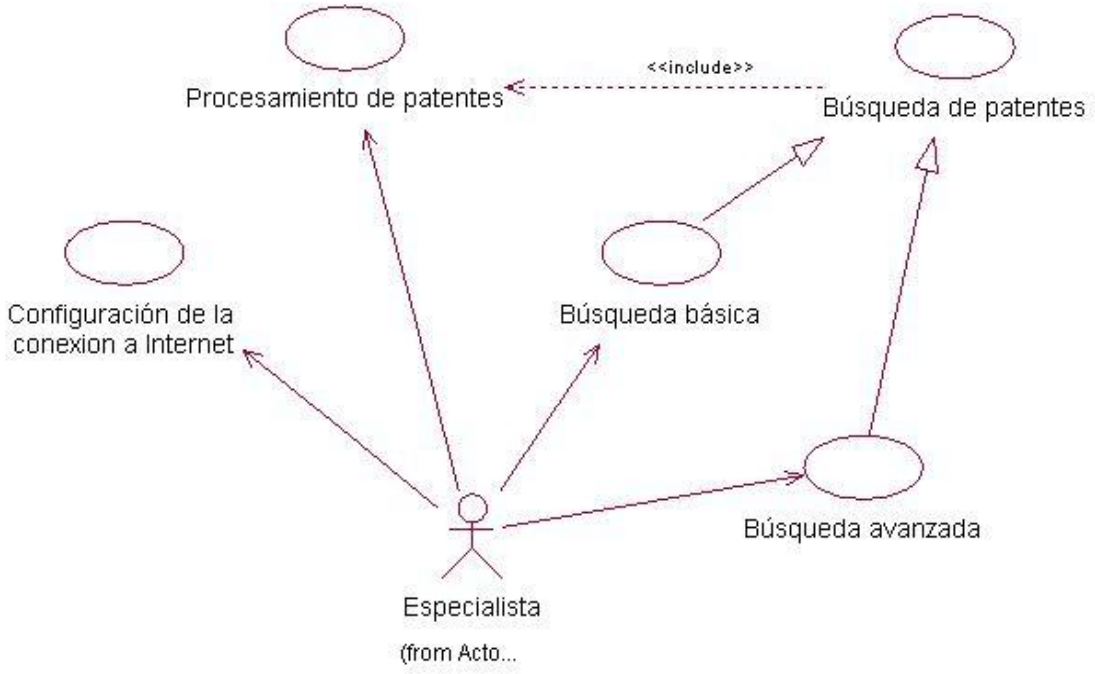


Figura 1 Vista arquitectónica del Modelo de Casos de Uso del Sistema.

La justificación de los CU arquitectónicamente significativos se mostró en el epígrafe 2.3; no obstante para obtener mayor claridad sobre cada uno de ellos se puede remitir a los **¡Error! No se encuentra el origen de la referencia.** 1 – 5 donde se profundiza más mediante su descripción textual.

2.4.2 Vista arquitectónica del modelo de análisis

Otra de las vistas que contiene la descripción de la arquitectura es la vista del modelo de análisis que muestra sus artefactos más significativos para la arquitectura; dentro de esos artefactos se encuentran:

1. La descomposición del modelo de análisis en paquetes de análisis y sus dependencias.

2. Las clases fundamentales del análisis como las clases de entidad que encapsulan un fenómeno importante del dominio del problema; las clases interfaz que encapsulan interfaces de comunicación importantes y mecanismos de interfaz de usuario; las clases de control que encapsulan importantes secuencias con una amplia cobertura.

3. Realizaciones de CU que describen cierta funcionalidad importante y crítica; que implican muchas clases del análisis, o aquellas que se centran en un CU importante que debe ser desarrollado al principio del ciclo de vida del producto.

El análisis se estructuró en varios paquetes para que fuera más fácil el desarrollo y comprensión de los modelos de análisis, dentro de los paquetes significativos arquitectónicamente se encuentran: Paquete de Configuración de la Conexión a Internet, el Paquete Búsqueda Básica, Búsqueda Avanzada y Procesamiento de Patentes. A continuación se muestran las realizaciones de estos paquetes que tienen trazas con CU que presentan una funcionalidad crítica en el sistema y por tanto se desarrollan en la primera versión.

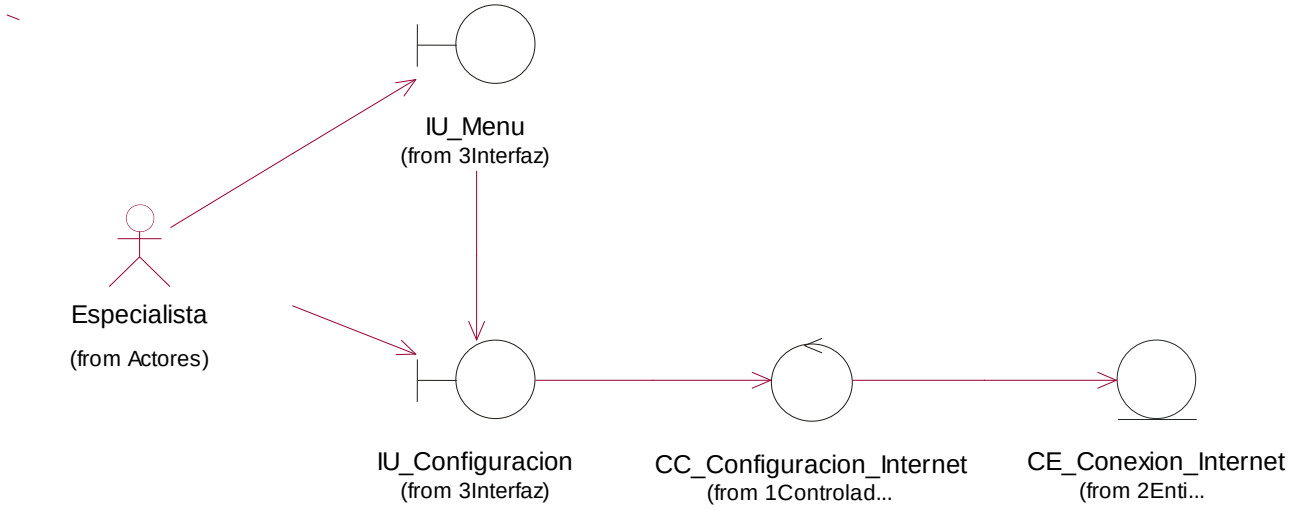


Figura 2 Vista del Modelo de Análisis. Paquete: Configuración de la Conexión a Internet.

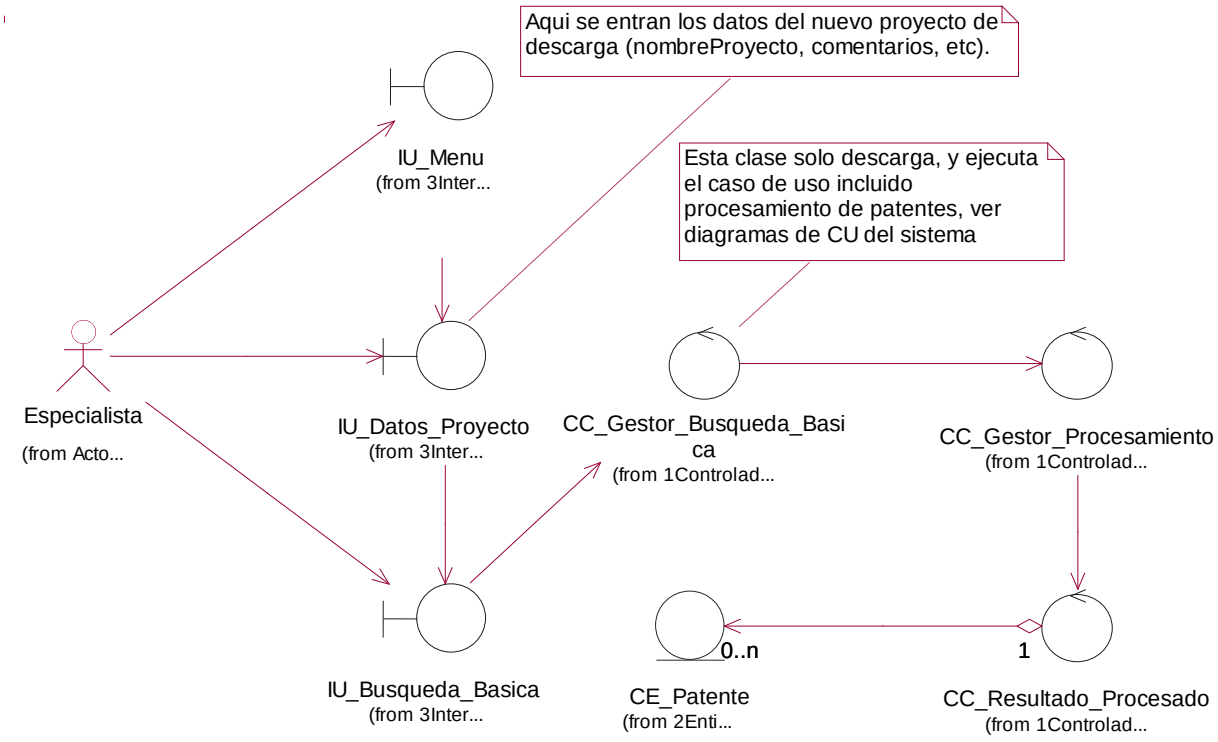


Figura 3 Vista del Modelo de Análisis. Paquete: Búsqueda Básica.

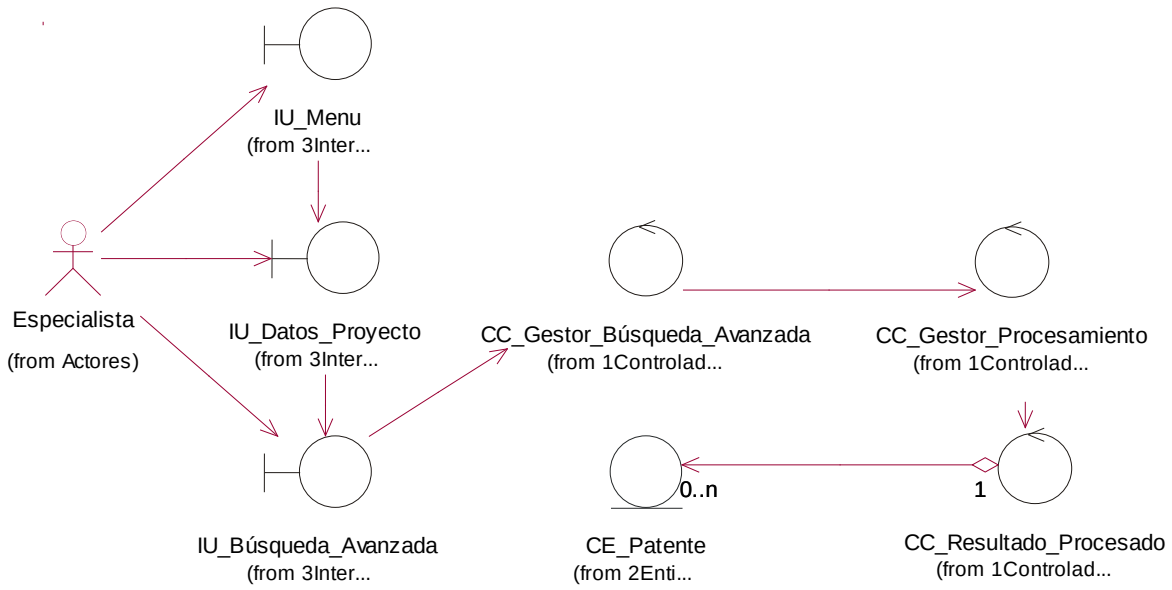


Figura 4 Vista del Modelo de Análisis. Paquete: Búsqueda Avanzada.

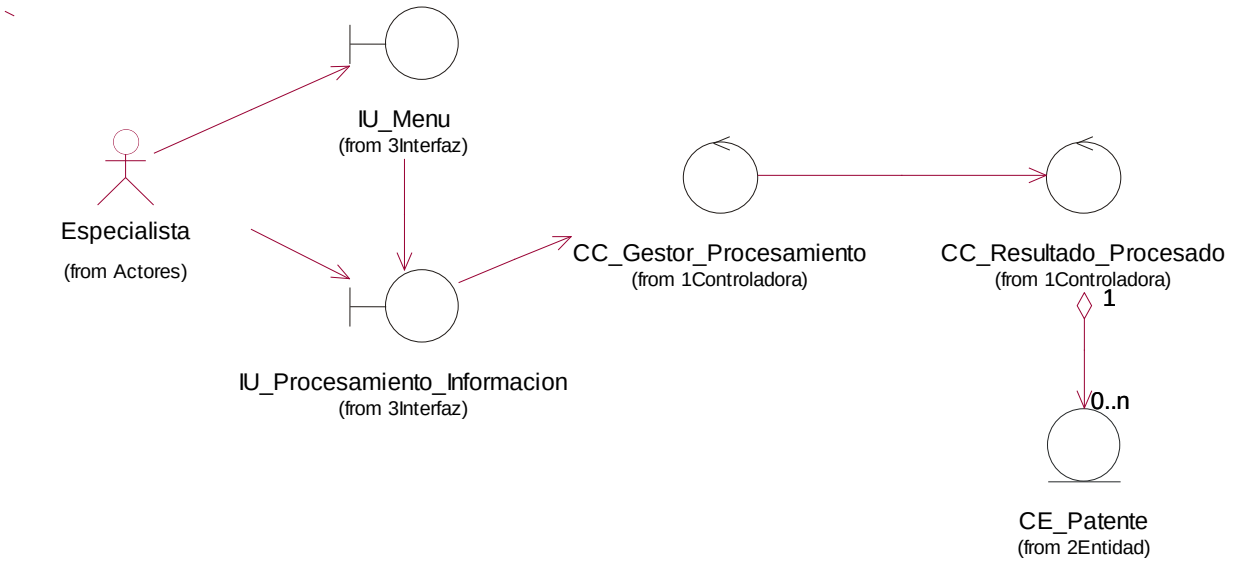


Figura 5 Vista del Modelo de Análisis. Paquete: Procesamiento de Patentes.

2.4.2.1 Justificación de las Clases del Análisis arquitectónicamente significativas.

2.4.2.1.1 Paquete: Configuración de la conexión

IU_Configuracion: Interfaz de usuario correspondiente al módulo de configuración de la conexión a internet.

CC_Configuracion_Internet: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio.

CE_Conexion_Internet: Clase entidad que representa una conexión a Internet, posee atributos como: IP del Proxy, puerto, usuario y contraseña, entre otros.

2.4.2.1.2 Paquete: Búsqueda Básica

IU_Datos_Proyecto: Interfaz de usuario correspondiente a la entrada de los datos de un proyecto de descarga o búsqueda de información de Internet, para el control y la identificación de las búsquedas.

IU_Busqueda_Basica: Interfaz de usuario correspondiente a la búsqueda básica de patentes de Internet, muestra los detalles del proceso de descarga.

CC_Gestor_Busqueda_Basica: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio específicamente con el módulo o subsistema de búsqueda básica.

CC_Gestor_Procesamiento: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio específicamente con el módulo o subsistema de procesamiento de HTML.

CC_Resultado_Procesado: Clase de control, que se encarga de la gestión de las patentes procesadas.

CE_Patente: Clase que representa una patente, posee los atributos de una patente, como número de la patente, autor, fecha de publicación, entre otros.

2.4.2.1.3 Paquete: Búsqueda Avanzada

IU_Datos_Proyecto: Interfaz de usuario correspondiente a la entrada de los datos de un proyecto de descarga o búsqueda de información de Internet, para el control y la identificación de las búsquedas.

IU_Busqueda_Avanzada: Interfaz de usuario correspondiente a la búsqueda avanzada de patentes de Internet, muestra los detalles del proceso de descarga.

CC_Gestor_Búsqueda_Avanzada: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio específicamente con el módulo o subsistema de búsqueda avanzada.

CC_Gestor_Procesamiento: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio específicamente con el módulo o subsistema de procesamientoHTML.

CC_Resultado_Procesado: Clase de control, que se encarga de la gestión de las patentes procesadas.

CE_Patente: Clase que representa una patente, posee los atributos de una patente, como: número de la patente, autor, fecha de publicación, entre otros.

2.4.2.1.4 Paquete: Procesamiento de Patentes

IU_Procesamiento_Informacion: Interfaz de usuario correspondiente al procesamiento de patentes sin pasar por el proceso de descarga de las mismas.

CC_Gestor_Procesamiento: Clase de control, que se encarga de la gestión de los mensajes entre la capa de presentación y la capa de negocio específicamente con el módulo o subsistema de procesamientoHTML.

CC_Resultado_Procesado: Clase de control, que se encarga de la gestión de las patentes procesadas.

CE_Patente: Clase que representa una patente, posee los atributos de una patente, como: número de la patente, autor, fecha de publicación, entre otros.

2.4.3 Vista arquitectónica del Modelo de Diseño

La vista del modelo de diseño muestra sus artefactos más relevantes desde el punto de vista de la arquitectura, dentro de los que se encuentran:

1. La descomposición del modelo de diseño en subsistemas, sus interfaces y las dependencias entre ellos.
2. Clases del diseño fundamentales, por ejemplo aquellas que tienen trazas con clases del análisis significativas.
3. Realizaciones de CU del diseño que describan alguna funcionalidad importante y crítica, que impliquen muchas clases del diseño, etc.

Los subsistemas de diseño son una forma de organizar los diferentes artefactos del modelo de diseño en piezas más adaptables. En el diseño del sistema Predictor se definieron 6 subsistemas de los cuales sólo se consideran importantes para la arquitectura los siguientes: *Subsistema de Presentación, Subsistema Configuración, Subsistema Comunes, Subsistema Descarga y Subsistema ProcesamientoHTML*

El modelo de diseño describe la realización física de los CU centrándose en como los requerimientos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además el modelo de

diseño sirve de abstracción de la implementación del sistema, y es de ese modo utilizado como una entrada fundamental de las actividades de implementación (6).

En el presente trabajo el modelo de diseño (Ver figura 6) está compuesto por un sistema de diseño que contiene cada uno de los subsistemas de diseño que son considerados arquitectónicamente significativos.

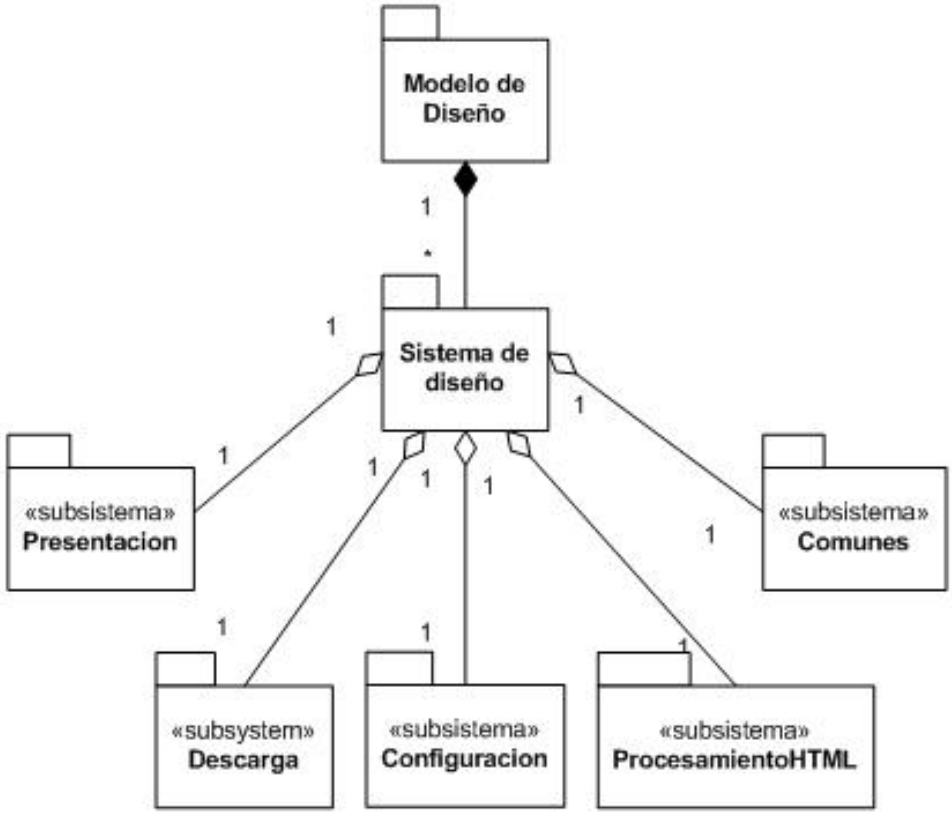


Figura 6 Vista Arquitectónica del Modelo de Diseño del Sistema.

A continuación se explica la funcionalidad de cada uno de estos subsistemas, se muestra el diagrama de las clases que lo conforman y se describen las clases más importantes para la arquitectura.

2.4.3.1 Subsistema Comunes

Este subsistema es de servicio, en él se encuentran todas las clases e interfaces que son usadas por otros subsistemas para realizar diferentes funcionalidades. Debido a la importancia que tiene para comprender los demás subsistemas se decidió que fuese el primero en presentarse.

En este subsistema no todas las clases se relacionan entre sí, pues la función de este es brindar apoyo y facilitar el trabajo de los demás subsistemas. En Comunes se modelan las clases que son usadas por más de un subsistema que tienen funcionalidades específicas como por ejemplo: Configuración, ProcesamientoHTML o Descarga, para que de esta forma queden estos subsistemas menos acoplados y dependan lo menos posible unos de los otros.

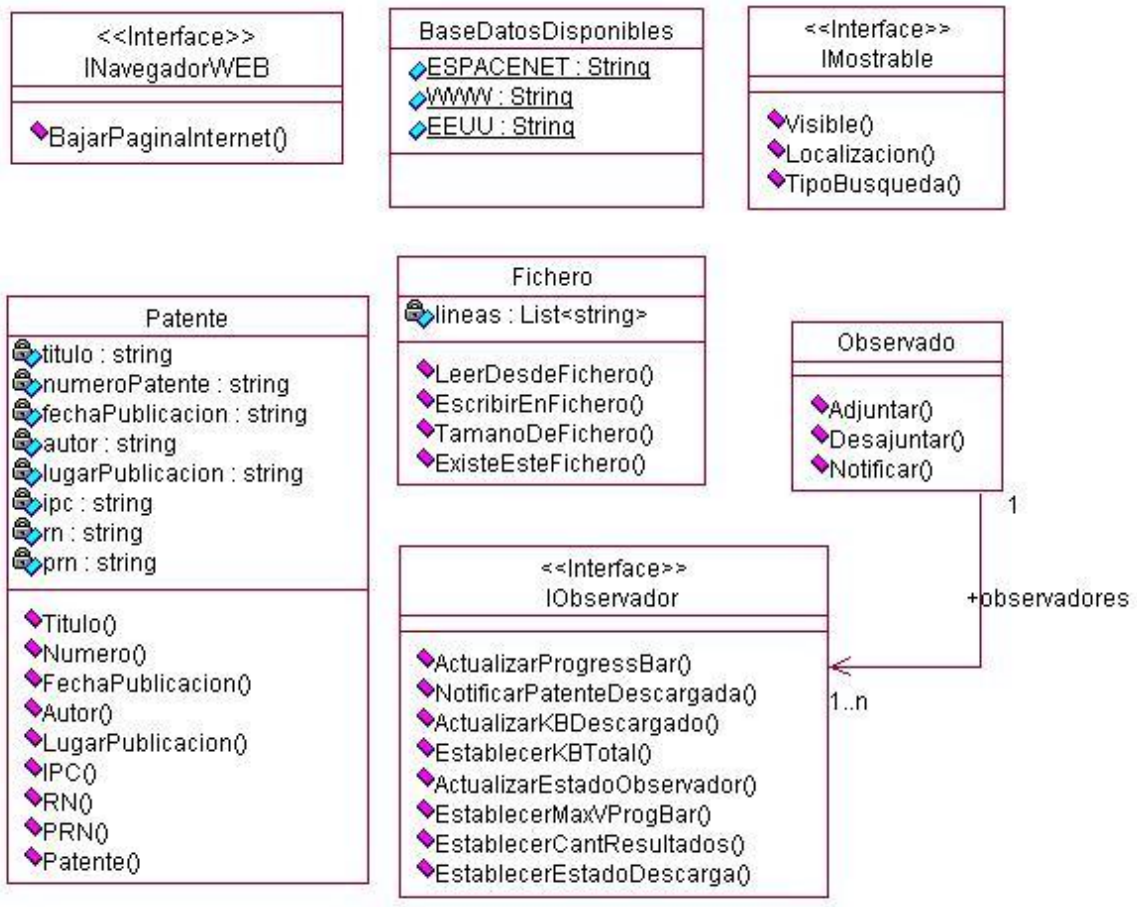


Figura 7 Diagrama de clases del diseño. Subsistema Comunes.

A continuación se explican brevemente cada una de las clases arquitectónicamente significativas para el Subsistema Comunes:

- **Fichero**: Es la que permite el trabajo con los ficheros. Se usa en la descarga y procesamiento. Posee funciones que son las encargadas de leer, escribir, verificar si existe y devolver el tamaño del fichero seleccionado.
- **Patente**: Es la clase más importante y el motivo fundamental para la realización del proyecto, posee como atributos los parámetros de interés para luego realizar los análisis. Al realizar las descargas se procesan cada uno de los ficheros descargados para extraer de ellos objetos de

esta clase, y luego conformar los ficheros finales con listas de esos mismos objetos. Esta clase es importantísima para todos los subsistemas.

- **Observado**: El objetivo de esta clase es usar el patrón Observer. Hace función del observado abstracto, del cual hereda el observado concreto MotorSolicitudes, perteneciente al Subsistema Descarga.
- **IObservador**: Es una interface que completa el patrón Observer. Es el observador abstracto. A ella la implementan los controles de usuario que realizan las descargas (IUBusquedaBasica e IUBusquedaAvanzada), quienes tienen una barra de progreso (ProgressBar) que se actualiza a medida que MotorSolicitudes va descargando los archivos.
- **BaseDatosDisponibles**: Tiene como atributos todas las bases de datos de la que se descarga información, sus nombres y como valores las direcciones electrónicas. Es usada en el momento de seleccionar la fuente de búsqueda, para entonces conformar la dirección electrónica, en esta clase es donde se agregan las nuevas bases de datos de las que se desea que se descargue información.
- **IMostrable**: Es una interface que tiene como métodos si está visible o no, el tipo de búsqueda y la localización, los que son sobre escritos de forma diferente según el tipo de búsqueda que se realice.
- **INavegadorWEB**: Es una interface implementada por los controles que realizan las búsquedas. Cada uno implementa de manera distinta el método que baja las páginas de Internet, definido en esta interface.

2.4.3.2 Subsistema Presentación

Este subsistema se encuentra en la Capa de Presentación en la arquitectura del sistema, sólo están en el aquellos componentes que permiten la interacción con el cliente como son las ventanas y formularios para capturar o mostrar datos.

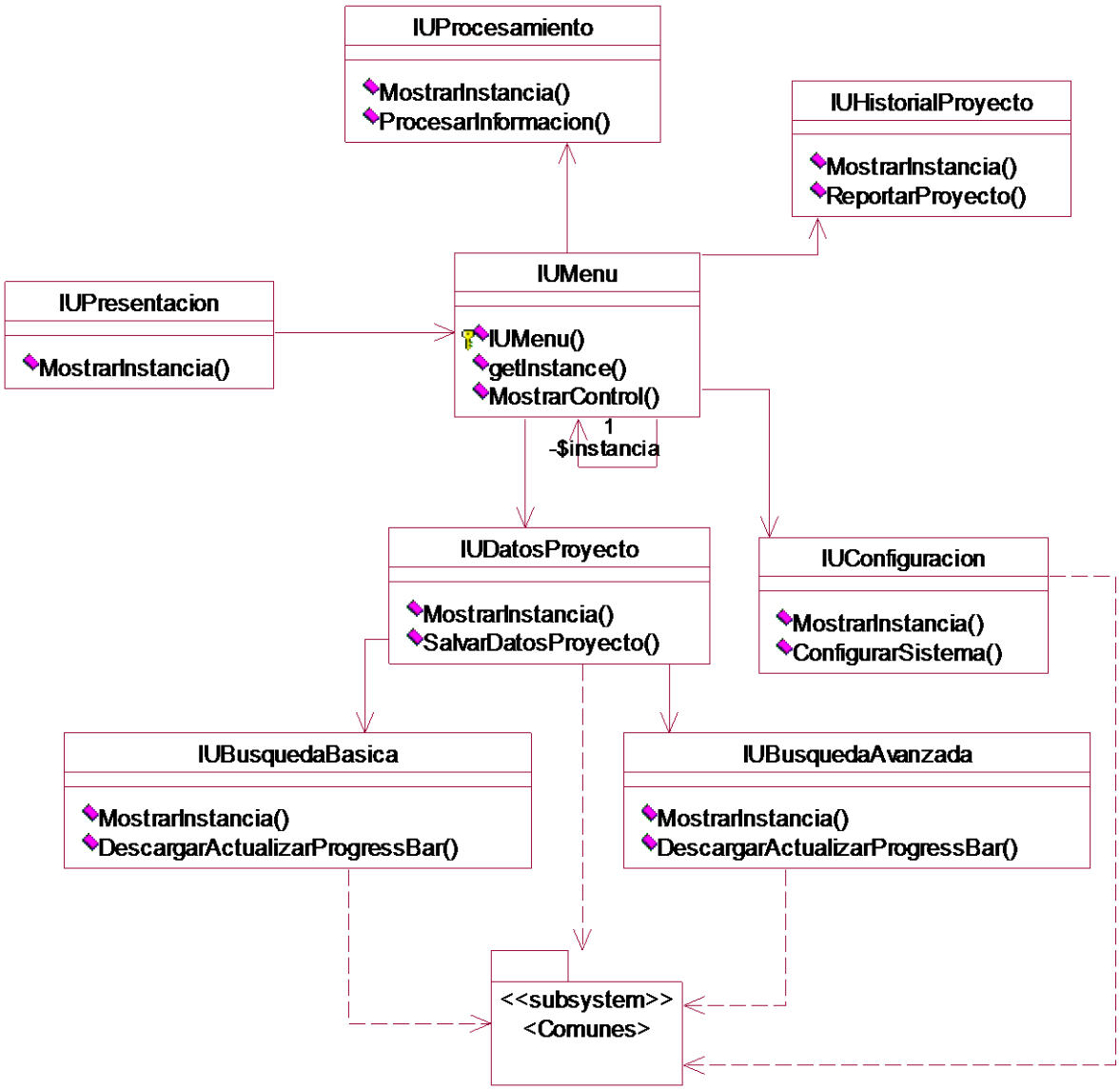


Figura 8 Diagrama de clases del diseño. Subsistema Presentación.

A continuación se describen las clases arquitectónicamente significativas para el Subsistema Presentación:

- ***IUPresentación****:* Es un formulario splash que carga al iniciar la aplicación con una imagen representativa del software y la empresa para la cuál se está trabajando. Tiene un Timer que es el encargado de controlar el tiempo durante el cuál va a estar activa, y desaparece después de un tiempo determinado, dándole paso a la IUMenu.
- ***IUMenu****:* Es la forma principal de la aplicación, contiene un menú a la izquierda con las distintas funcionalidades del sistema, en los eventos On_Click de cada uno de los componentes del

menú se invocan los distintos controles de usuarios para capturar o mostrar información. Mediante esta clase es que se pueden realizar todos los CU del sistema.

- ***IUConfiguración**:* Este control de usuario se invoca desde un evento de IUMenu, tiene como función brindar la posibilidad de configurar el tipo de conexión con la que se va a trabajar para realizar la descarga. Posee componentes como TextBox, CheckBox y Botones encargados de capturar los datos de configuración y enviarlos hacia otros subsistemas para su procesamiento y posteriormente ser usados para la realización de la descarga.
- ***IUDatosProyecto**:* Brinda la posibilidad de asignarle un nombre a la descarga que se desea realizar, así como otros aspectos de interés para el usuario, como puede ser, comentario y descripción. Aquí es donde se selecciona el directorio donde se va a guardar la descarga mediante un OpenDialog. Después de culminadas estas operaciones se guardan los cambios realizados y automáticamente se pasa a la interfaz de descarga, la cuál puede ser Básica o Avanzada, que se explican a continuación.
- ***IUBusquedaBasica**:* La función principal de esta clase es capturar los datos para poder realizar la búsqueda básica. Se seleccionan las Bases de Datos de las que se desea ver la información y se introduce el término a buscar, dando la opción de que esté en el título o en el resumen de la patente. Además brinda la posibilidad al usuario de saber cuántas patentes fueron encontradas y el progreso de la búsqueda mediante la barra de progreso antes mencionada, cuando se termina de bajar toda la información el sistema lo notifica para luego pasar a otras interfaces.
- ***IUBusquedaAvanzada**:* Al igual que la IUBusquedaBasica, su función es capturar los datos para la búsqueda, pero en este caso se capturan más parámetros para realizar la búsqueda, como son: fecha de publicación, solicitante de la patente, clasificación internacional de la patente y el término de búsqueda. Las bases de datos se configuran igual que en la básica, y brinda la misma posibilidad de ver cuánto ha avanzado la descarga que se esté realizando mediante la barra de progreso.
- ***IUHistorialProyectos**:* La función de esta clase es mostrar las descargas realizadas por el usuario que esté trabajando, permitiendo seleccionar cualquiera y reportar todo lo referente a ella. También permite buscar cualquier otra descarga que se encuentre guardada en la PC.
- ***IUProcesamiento**:* Su función es permitir al usuario seleccionar un directorio donde se encuentre una descarga previamente realizada para llevar a cabo el procesamiento y obtener las patentes encontradas en esa descarga.

2.4.3.3 Subsistema de Configuración.

Pertenece a la capa de negocio. Su función es configurar la conexión a Internet para realizar las descargas, para esto contiene las clases que permiten llevar a cabo dicho proceso.

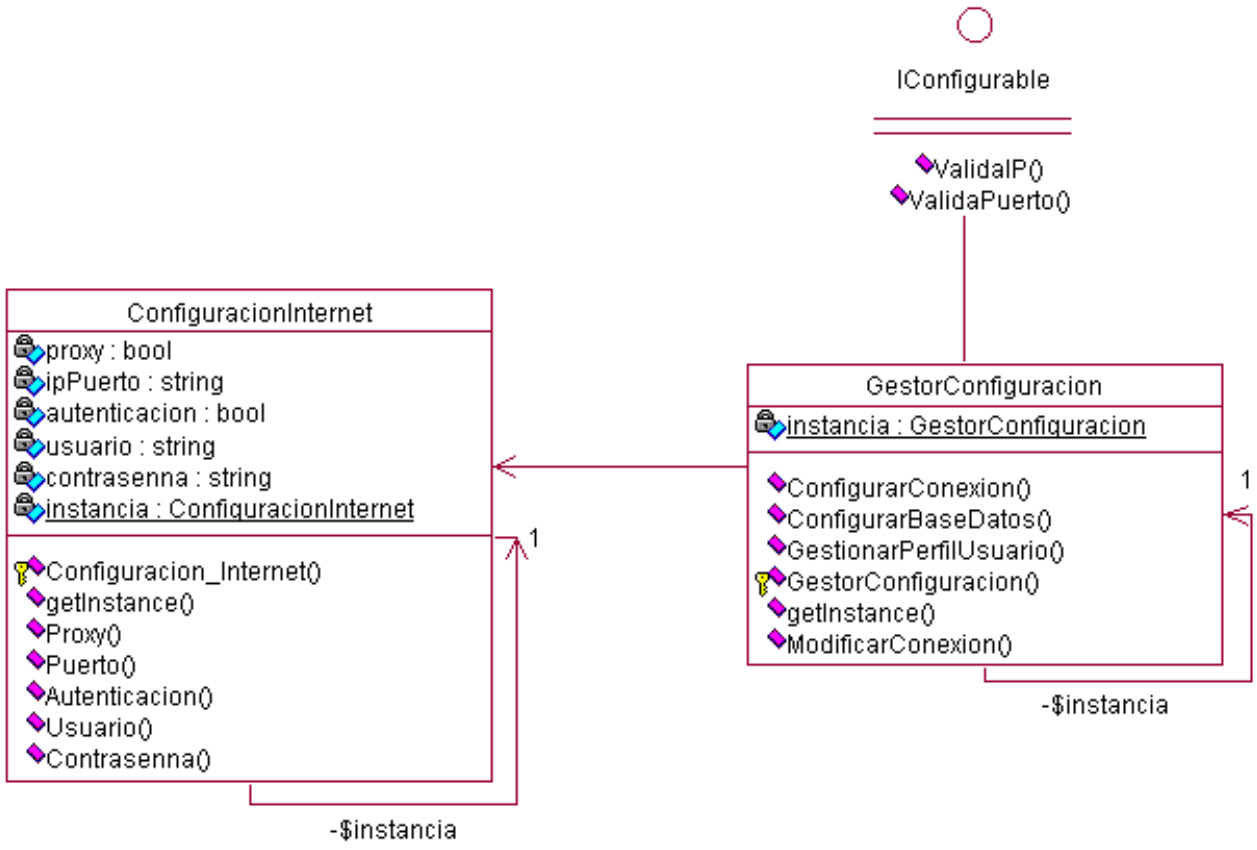


Figura 9 Diagrama de clases del diseño. Subsistema Configuración.

A continuación se describen las clases arquitectónicamente significativas para el Subsistema Configuración:

- ***IConfigurable***: Es una interface que tiene dos métodos que permiten validar el IP del Proxy y el Puerto para realizar la conexión.

- ***GestorConfiguracion****:* Tiene una instancia de ella misma que permite que exista sólo un objeto de ella en todo el sistema. Su función es gestionar el proceso de configurar la conexión.

Captura los datos introducidos mediante la interfaz de usuario y configura la conexión creando un objeto ConfiguracionInternet.

- ***ConfiguracionInternet****:* Su función es guardar los datos de la conexión. Implementa el patrón Singleton, de esa manera se puede acceder al objeto creado una vez configurada la conexión, desde la descarga, y se garantiza que sólo exista una conexión a Internet.

2.4.3.4 Subsistema Descarga

Este es el subsistema principal, su función es descargar la información según el tipo de búsqueda y los criterios deseados. Su funcionamiento depende de otros dos subsistemas, necesita los datos del tipo de conexión que se va a usar, que se deben haber capturado con anterioridad por el Subsistema Configuración. Con esa información y los criterios de búsqueda, se realiza la descarga de la base de datos seleccionada y se almacena esa información en el directorio escogido por el usuario. Una vez terminada la descarga, se procede a filtrar la información que está contenida en los ficheros y conformar el listado de patentes obtenidas, esta tarea corresponde al Subsistema ProcesamientoHTML, quien crea los ficheros con todas las patentes encontradas.

El subsistema funciona de la siguiente manera: según la búsqueda a realizar, avanzada o simple, es que se instancia la clase BusquedaBasica o BusquedaAvanzada, se crea un objeto ParametrosSolicitud y se instancia la clase MotorSolicitudes, quien con los parámetros a buscar inicia la descarga usando la configuración de Internet antes hecha. Crea un hilo de ejecución de la descarga y gestión de ella dando la posibilidad de ponerlo en diferentes estados: ejecución, pausa o cancelación. Se construye la *url* de la página de la base de datos seleccionada y descarga la página con los resultados. Selecciona los vínculos existentes en ella y va conformando las *url* y descargando cada una. Una vez descargadas todas las páginas encontradas se pasa al procesamiento de los ficheros, tarea que realiza el Subsistema ProcesamientoHTML.

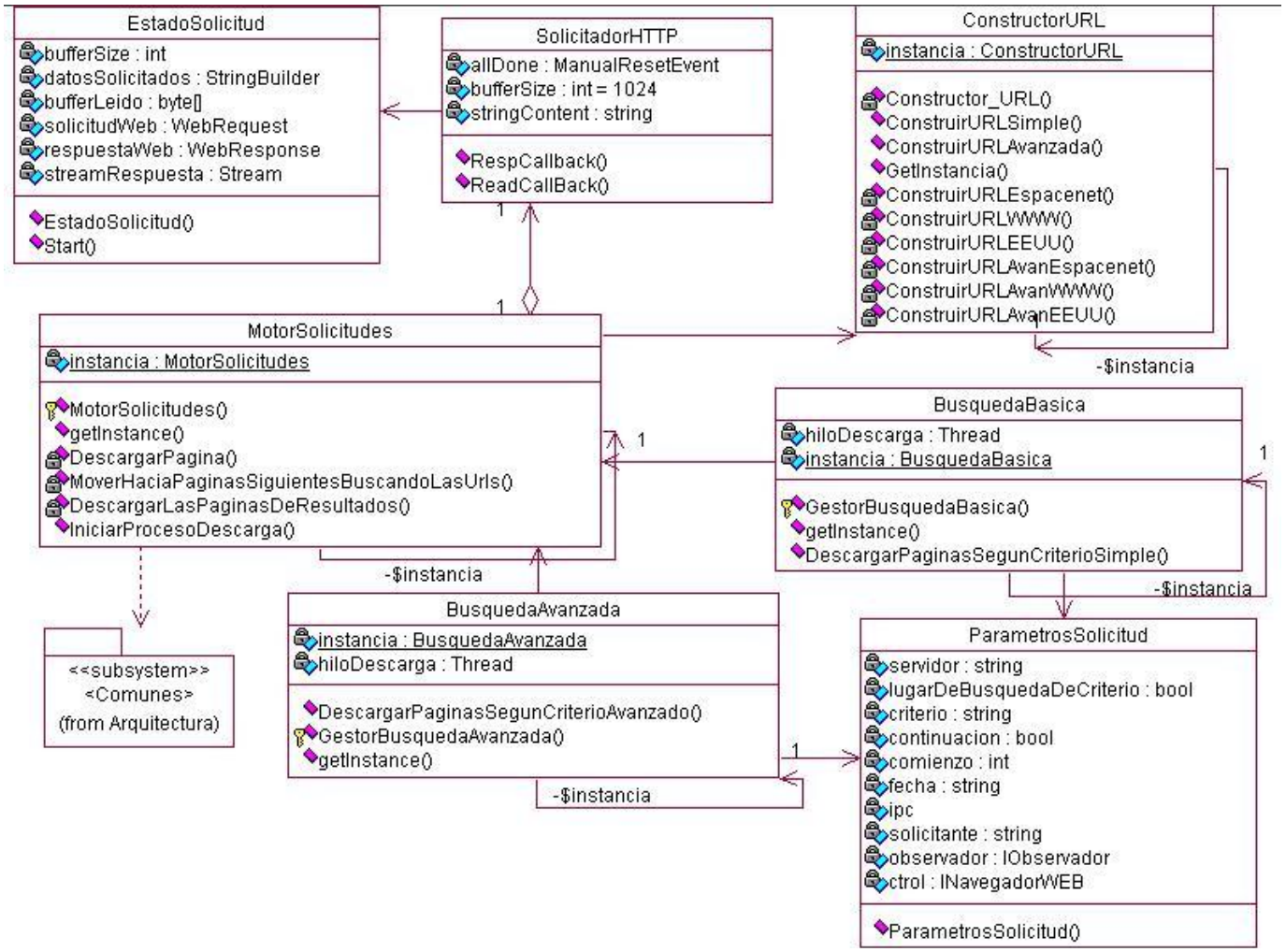


Figura 10 Diagrama de clases del diseño. Subsistema Descarga.

A continuación se describen las clases arquitectónicamente significativas para el Subsistema Descarga:

- ***ParametrosSolicitud****:* Es una clase entidad encargada de guardar los datos de los parámetros con que se realiza la solicitud, esos datos son los criterios de búsqueda y la base de datos seleccionada.

- ***BusquedaBasica****:* Es la encargada de gestionar todo el proceso de búsqueda básica. Con los parámetros de la búsqueda instancia al MotorSolicitudes para que comience con el proceso de descarga. Ya finalizada la descarga invoca a la clase gestora encargada del procesamiento de los ficheros para que se conforme el listado final de las patentes encontradas.

- ***BusquedaAvanzada****:* Tiene casi la misma función que la BusquedaBasica, pero con la única diferencia de que maneja más criterios de búsqueda. Para la avanzada se introduce la fecha

de publicación, el solicitante y otros, que en la básica no se tratan. Todo lo demás funciona de la misma forma que en la búsqueda básica.

- ***ConstructorURL****:* Se encarga de construir las *urls* para descargar las páginas. Para eso necesita de la base de datos seleccionada, el tipo de búsqueda y los criterios. Con esos datos entonces conforma las *urls* para que MotorSolicitudes realice las descargas.

- ***MotorSolicitudes****:* Es la clase que controla todo el proceso de descarga, con el objeto ParametrosSolicitud construye las *urls* y descarga las páginas encontradas, su relación con el subsistema Comunes se debe a que utiliza la clase Fichero.

2.4.3.5 Subsistema ProcesamientoHTML

Este subsistema se encuentra en la capa de negocio, su función es hacer el procesamiento de los ficheros descargados, que contienen el código *HTML*. Entre sus principales funciones se encuentran: seleccionar las *urls*, eliminar las que estén repetidas para de esa forma evitar que la información aparezca duplicada, construir el listado de patentes y crear las muestras finales, que son los dos ficheros que se obtienen como resultado de este subsistema, el Procite.txt y el Excel.txt, estos facilitan el posterior análisis de la información por los especialistas de Delfos.

El diagrama de clases de este subsistema está compuesto por las clases MotorProcesamiento y ResultadoProcesado, y por la clase interface IProcesable, además utiliza las clases definidas en el subsistema comunes, este subsistema es el que tiene la función de buscar y seleccionar los campos que se necesitan en los ficheros, de manera tal que sólo queden aquellos parámetros de las patentes que son importantes para conformar la información que va a ser guardada en los ficheros finales.

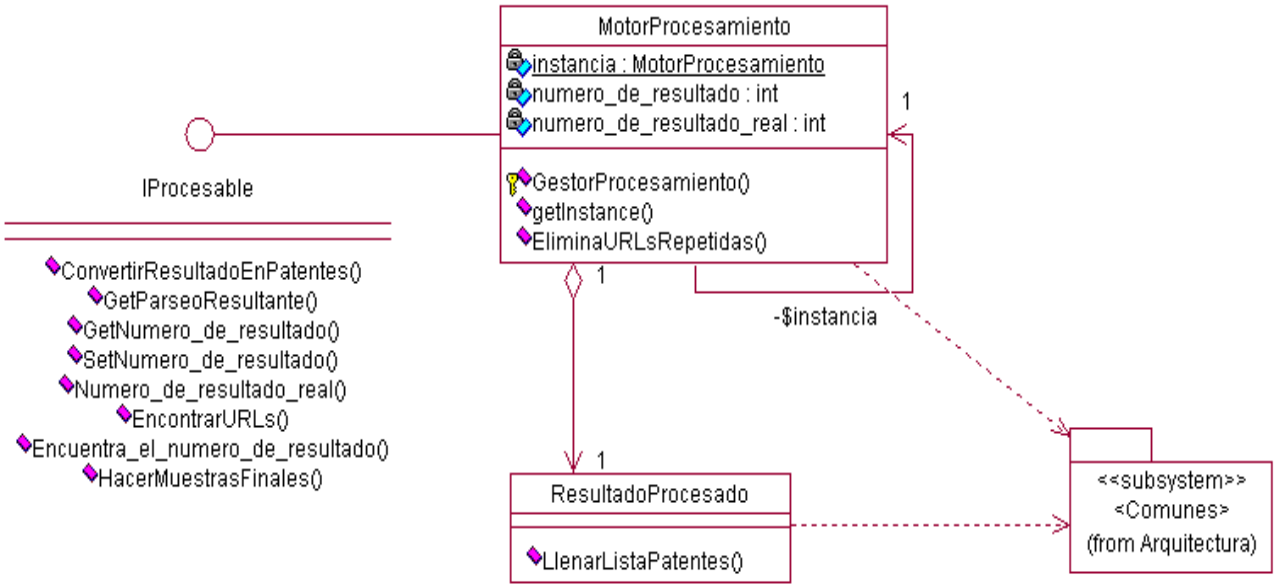


Figura 11 Diagrama de clases del diseño. Subsistema ProcesamientoHTML.

Las relaciones entre las clases MotorProcesamiento y ResultadoProcesado con el subsistema Comunes se deben a la necesidad de clases que existen en ellos para realizar sus funcionalidades. De Comunes usa la clase Patente para crear el listado de patentes; la clase BaseDatosDisponibles, para seleccionar la base de datos para realizar la descarga y la clase Fichero para tratar todas las operaciones con ficheros. La segunda también necesita de Fichero y Patente para realizar sus funcionalidades.

A continuación se explican que funciones cumplen cada una de las clases del subsistema:

- ***IProcesable***: Es una interface de la cual va a heredar su comportamiento la clase MotorProcesamiento. Tiene varias funciones que serían sobrecargadas en aquellas clases que la implementen.

- ***MotorProcesamiento****:* Es la clase gestora del subsistema, implementa la interface IProcesable. Su tarea es gestionar todas las funciones referentes al procesamiento. Tiene como atributos más importantes una instancia de ella misma y un objeto ResultadoProcesado. Además implementa el patrón Singleton.

- ***ResultadoProcesado****:* Es una clase que tiene como atributo un listado de patentes. Su funcionalidad principal es poder leer de un fichero y crear el listado de patentes que contenga el mismo. Tiene como función llenar una lista de patentes según un fichero dado, entre otras.

2.4.4 Vista arquitectónica del modelo de despliegue

El modelo de despliegue va a definir la estructuración física que tendrá el sistema, estas estructuras son representadas a través de nodos que responden a los elementos hardware que intervienen en el negocio y sobre los cuales se ejecutarán los elementos software. Los nodos son interconectados entre ellos, estableciendo la lógica de transportación de los datos y/o información.

La vista del Modelo de Despliegue muestra los artefactos relevantes para la arquitectura y de igual manera se visualiza como va a ser la comunicación entre nodos (características de conexiones), la capacidad de proceso, tamaño de memoria, etc.

A continuación se presenta la vista del Modelo de Despliegue, que está compuesta por 5 nodos físicos los cuales se describen posteriormente:

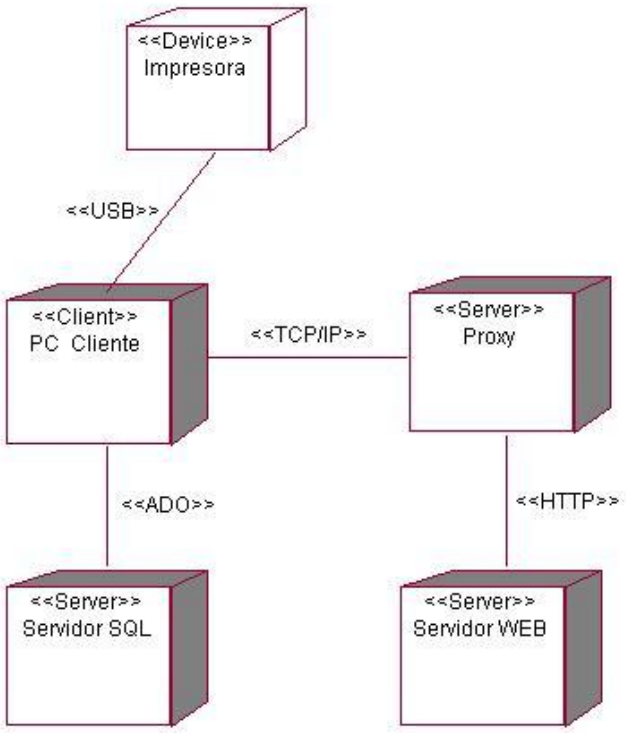


Figura 12 Vista Arquitectónica del Modelo de Despliegue.

2.4.4.1 Descripción de los nodos.

PC Cliente «ordenador»: Este nodo se asocia al computador donde va a estar *ejecutándose la aplicación Predictor*, es aquí donde el cliente va a obtener los resultados que generará el sistema. Este computador debe tener como características físicas ser Pentium IV, con un mínimo de 256 Mb de RAM para ejecutar la aplicación y generar los resultados.

Proxy «servidor»: Este nodo se asocia al servidor que brindará el *servicio de acceso a internet* a través de la intranet local de la consultoría Delfos. En el mismo será configurado el acceso a internet, así como compartir a los usuarios de la red la navegación a la misma. Se propone un ordenador Pentium IV o superior, con 512 Mb como mínimo de memoria RAM.

Servidor WEB «servidor»: Este nodo se corresponde al servidor que brinda el servicio de publicación de las patentes públicas en Internet (esp@cenet, world wide web y Estados Unidos). De este nodo sólo se *necesita su disponibilidad*.

SQL Server «servidor»: El nodo está referido para la persistencia de las descargas de patentes realizadas por los clientes Delfos dentro de la misma consultoría en aras de ganar en tiempo de descarga y procesamiento de la información. Por el gran volumen de información que va a almacenar y por las características de las consultas a efectuar a la base de datos que consumen demasiado recurso del sistema, se propone un servidor Pentium Dual Core, con 1 Gb de memoria RAM como mínimo.

Impresora «dispositivo»: Es definida para que el cliente si lo desease pueda imprimir los reportes generados después del procesamiento de la información de las patentes descargadas.

2.4.4.2 Protocolos de comunicación entre los nodos.

Los nodos son interconectados entre ellos, a través de estas conexiones navegan paquetes de información. Los protocolos de comunicación que se establecen son:

1. «ADO» Entre el nodo PC Cliente y el nodo SQL Server, se encarga de transportar los paquetes que gestionan los datos en el nodo de SQL Server.
2. «TCP/IP» Entre los nodos PC Cliente y el nodo Proxy, se encarga de crear el protocolo que brinda el acceso a internet.

3. «HTTP» Entre los nodos Proxy y el nodo Servidor WEB, este protocolo es el que empaqueta todo el flujo de interacción de información entre ambos servidores, solicitados por el nodo PC Cliente.
4. «USB» Entre el nodo PC Cliente y el nodo Impresora, por esta conexión se envía la solicitud empaquetada de información a imprimir a solicitud del cliente.

2.4.5 Vista arquitectónica del modelo de implementación

La vista arquitectónica del modelo de implementación incluye dentro de sus artefactos más significativos (6):

1. La descomposición del modelo en subsistemas, sus interfaces y las dependencias entre ellos.
2. Componentes clave, como aquellos que tienen trazas con las clases del diseño arquitectónicamente significativas, los componentes ejecutables y algunos otros que implementan mecanismos de diseño genéricos de los que dependen muchos otros componentes.

Los componentes se organizan de acuerdo a ciertos criterios, que representan decisiones de diseño. En este sentido, diferentes bibliografías apuntan que la arquitectura de software incluye justificaciones referentes a la organización y el tipo de componentes, garantizando que la configuración resultante satisface los requerimientos del sistema.

A continuación se presenta la vista general del modelo de implementación que incluye los diferentes subsistemas de implementación:

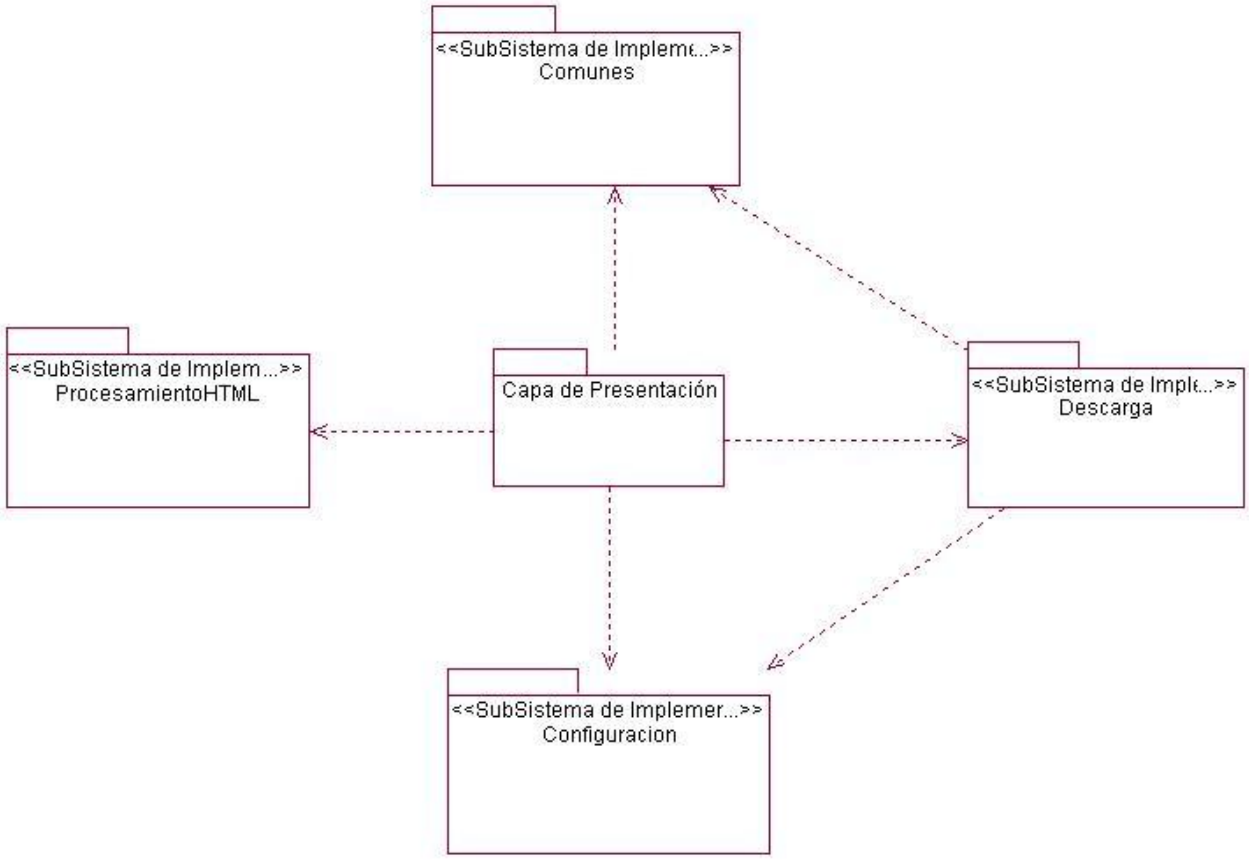


Figura 13 Diagrama de Implementación. Subsistemas de Implementación.

Los principales subsistemas que se definieron para la implementación fueron: Subsistema de Configuración, Subsistema Comunes, Subsistema de Descarga y Subsistema de Procesamiento, además se presenta la relación entre las interfaces de usuario que se encuentran en la Capa de Presentación, a continuación se presenta una vista arquitectónica de cada uno de estos subsistemas.

2.4.5.1 Diagrama de Implementación. Subsistema de Configuración.

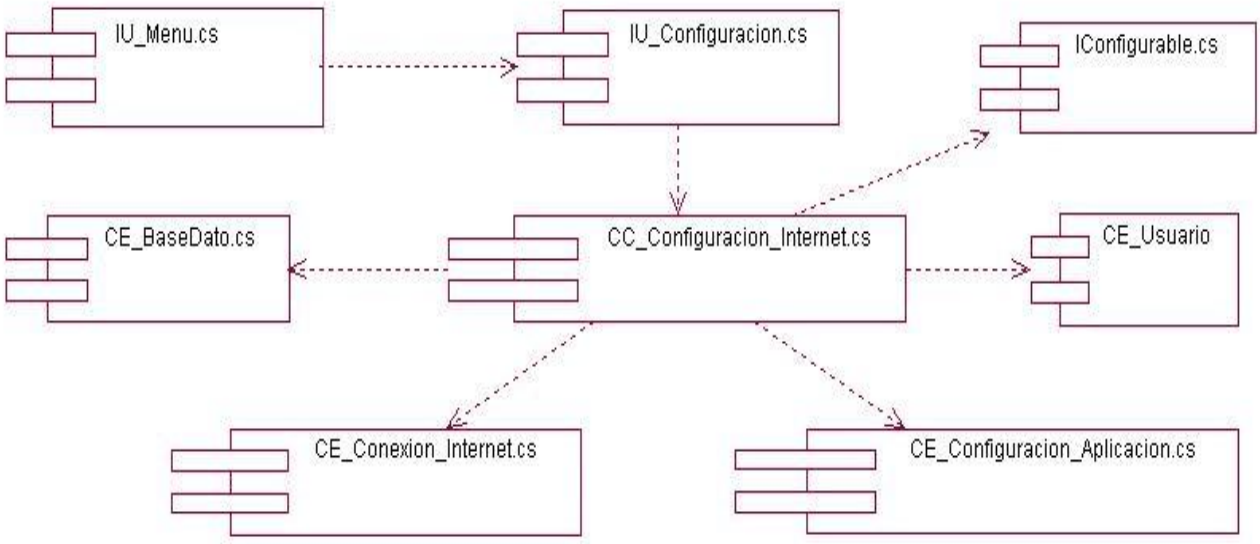


Figura 14 Vista arquitectónica del Modelo de Implementación. Subsistema de Configuración.

2.4.5.2 Diagrama de Implementación. Subsistema Comunes.

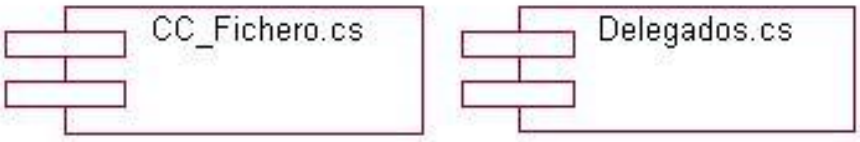


Figura 15 Vista arquitectónica del Modelo de Implementación. Subsistema Comunes.

2.4.5.3 Diagrama de Implementación. Subsistema de Descarga.

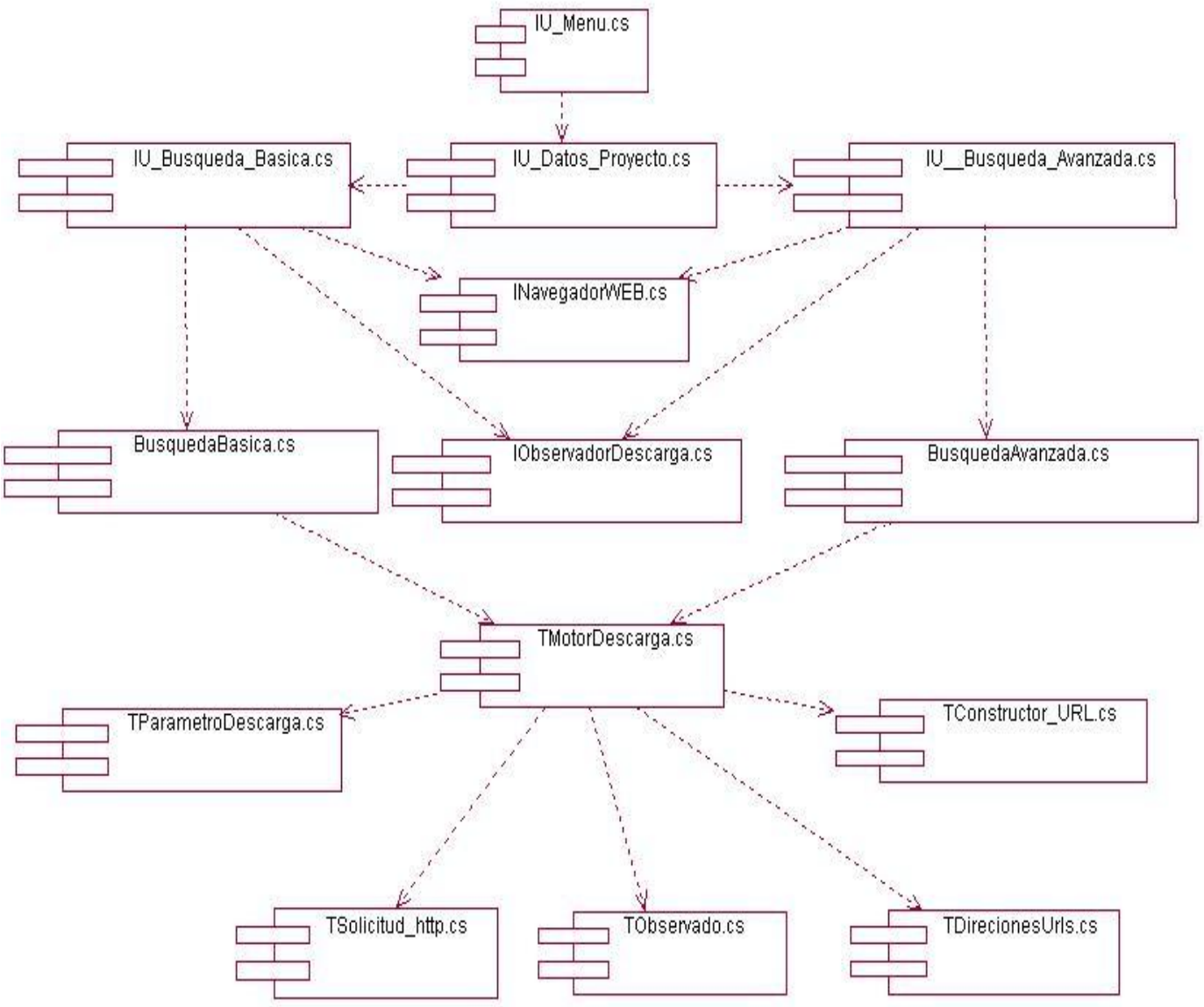


Figura 16 Vista arquitectónica del Modelo de Implementación. Subsistema de Descarga.

2.4.5.4 Diagrama de Implementación. Subsistema de ProcesamientoHTML.

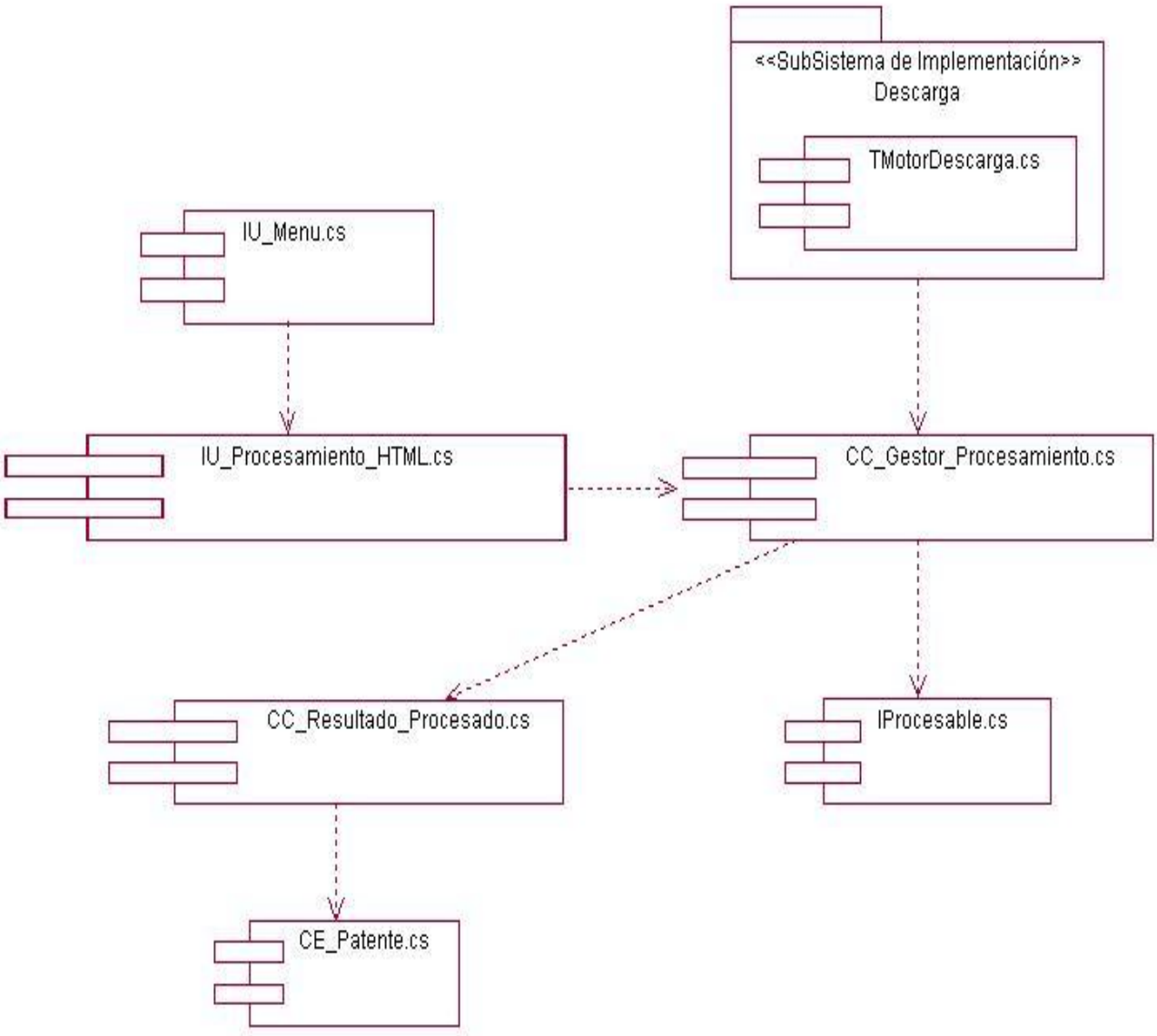


Figura 17 Vista arquitectónica del Modelo de Implementación. Subsistema de ProcesamientoHTML.

2.4.5.5 Relación entre las interfaces de usuario.

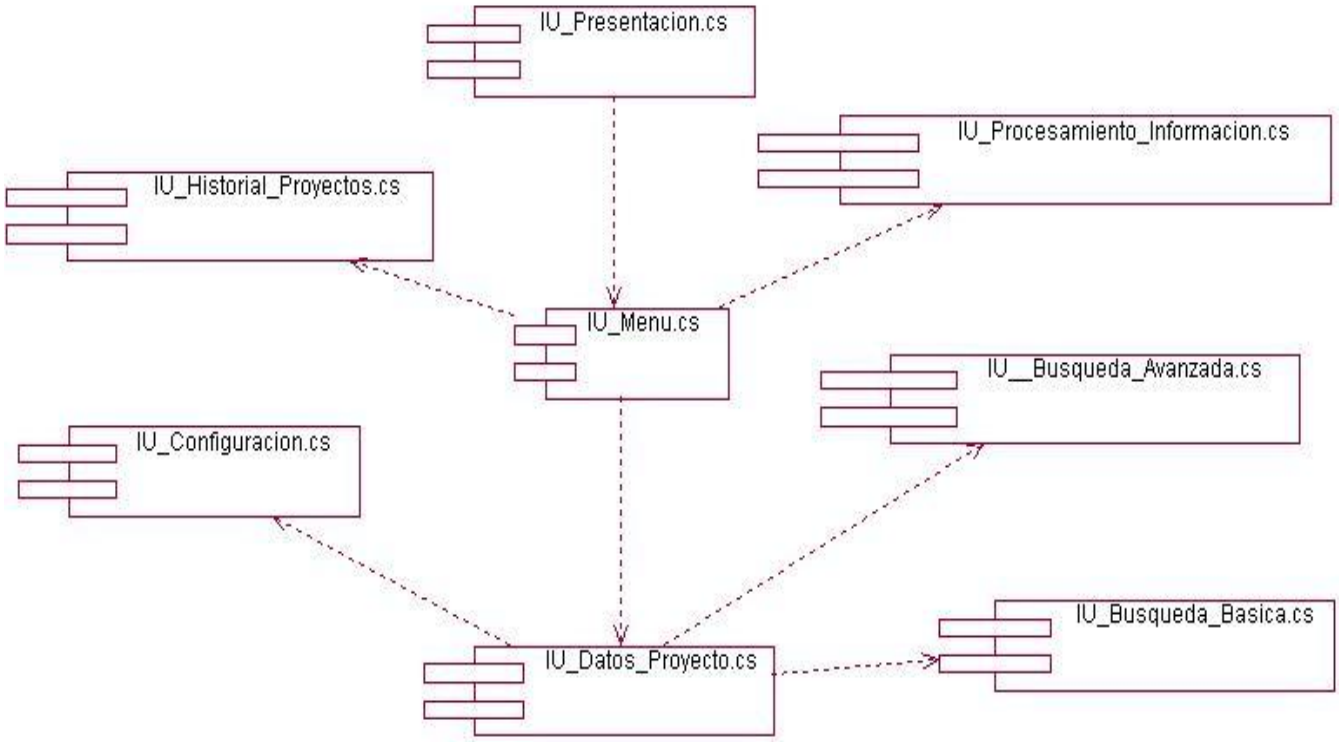


Figura 18 Vista arquitectónica del Modelo de Implementación. Relación entre las interfaces de usuario.

2.5 Estrategia de integración de los diferentes componentes.

Para lograr la integración de los diferentes componentes de un sistema se debe tener en cuenta los componentes creados por el equipo de desarrollo además de los ya existentes. La utilización de componentes ya existentes es de gran importancia ya que nos ahorraría tiempo de trabajo y tendríamos componentes independientes que podemos utilizar en el desarrollo de la aplicación sin tener que pasar por el proceso de su construcción, sólo integrarlos con los demás componentes. Dentro de los componentes existentes que se utilizan en este sistema se encuentran el: componente Outlook, y el TreeListView.

Para llevar a cabo la integración de los diferentes componentes ya existentes que se utilizan en el sistema se probaron primeramente de forma independiente, lo cual se conoce como pruebas unitarias o individuales, comprobando el comportamiento de los mismos de forma externa y analizando luego la implementación interna, pues se dispone del código fuente, los cuales son implementados por otras personas u organizaciones.

Para realizar la integración de las diferentes partes se realizó un análisis de los CU que aportan mayor valor a la arquitectura del sistema dejando definido el orden de integración de estas unidades de forma tal que se le dé cumplimiento a los requisitos funcionales correspondientes a cada uno de estos CU. Una vez creado el esqueleto de la aplicación (formulario principal), en el cual se cargarán todos los controles, componentes existentes y los creados por el equipo de desarrollo, se establecieron las relaciones e implementaciones necesarias para integrar:

- Componente Outlook: Ensamblado que se utiliza como menú izquierdo en la interfaz gráfica principal para mostrar las distintas opciones del sistema, el nombre viene dado por su parecido al original creado por Microsoft en su producto Outlook (cliente de correo incluido en su paquete office). Este componente proporciona al sistema una gran flexibilidad y provee una buena apariencia gráfica la cual se puede personalizar fácilmente.

- Componente TreeListView: Componente que se utiliza para mostrar reportes, el nombre viene dado por la mezcla de los controles básicos TreeView y ListView y con esta unión podemos lograr potencialidades graficas más personales y especializadas para llevar al usuario final la mayor satisfacción y comodidad posible, pues en cada item del Listview podemos encontrar un TreeView y en eso consiste la potencialidad de esta unidad compuesta.

Este sistema brinda soporte a la integración de nuevos o futuros componentes que se desarrollen después de haberse instalado el sistema en el entorno donde trabajará el cliente sin tener que reinstalar el mismo con otra versión del software. Esto proporciona una mayor extensibilidad al sistema, y facilita la tarea de actualización del mismo, este aspecto se logró siguiendo los siguientes pasos que mostraremos a continuación:

1. Primeramente se tiene en un fichero XML de configuración llamado config.XML, del cual se cargan en tiempo de ejecución, los controles de usuarios o plug-ins, que se deseen que el sistema pueda iniciar cuando el usuario lo solicite, ver código fuente en **¡Error! No se encuentra el origen de la referencia.**.

2. Luego en el evento Load del Formulario principal donde se desean mostrar los controles, se lee el fichero Config.XML usando un DataSet para ello, y el método ReadXml de dicha clase, luego iteramos cada uno de los DataRow del DataSet, llamando al método AddPlugin con el nombre y el camino de cada uno de los Plugin correspondiente a cada DataRow, ver código fuente en **¡Error! No se encuentra el origen de la referencia.**.

3. Implementación del método AddPlugin del paso anterior, ver **¡Error! No se encuentra el origen de la referencia.**

4. Implementación de la clase base Plugin, ver **¡Error! No se encuentra el origen de la referencia.**

5. ¿Cómo crear un nuevo Plugin para la interfaz de usuario?

    a. Usar Visual Estudio para crear un nuevo Windows Control Library.

    b. Adicionar una referencia a PlugInLib que contiene la clase base Plugin mencionada anteriormente.

    c. Ponerle un nombre descriptivo al control de usuario que se esté creando.

    d. Adicionar la directiva using para PlugInLib.

    e. Cambiar la clase base del control de usuario System.Windows.Forms.UserControl por PlugIn.

    f. Conectar cualquier evento que se quiera enviar a la aplicación principal.

    g. Adicionar las redefiniciones de métodos que sean necesarios para las llamadas desde la aplicación principal hacia el PlugIn.

    h. Construir la Interfaz de usuario en el control de usuario como cualquier otro control.

6. Implementación de la descripción anterior, ver **¡Error! No se encuentra el origen de la referencia.**

Con esta arquitectura se logra una interfaz de usuario más flexible, incorporando nuevos PlugIn con otras funcionalidades según soliciten los clientes, sin la necesidad de recompilar el código completo de la aplicación.

2.6 Definición de la configuración de los puestos de trabajo según los roles.

El equipo de desarrollo está compuesto por 9 integrantes, distribuidos en 4 puestos de trabajo, cuyas características mínimas relacionamos a continuación:

Tabla 1 Configuración de los puestos de trabajo según los roles.

Estación de Trabajo	Distribución de Trabajadores	Requisitos de ordenadores
1	1- Líder 1- Programador	Procesador 2.4 GHz. Memoria RAM de 512 Mb. Disco duro de 80 Gb. Tarjeta de red.
2	2- Arquitecto	Procesador 2.4 GHz. Memoria RAM de 512 Mb. Disco duro de 80 Gb. Tarjeta de red.
3	1- Analista 1- Programador 1- Planificador	Procesador 2.4 GHz. Memoria RAM de 512 Mb. Disco duro de 80 Gb. Tarjeta de red.
4	1- Analista 1- Diseñador de Sistema	Procesador 2.4 GHz. Memoria RAM de 512 Mb. Disco duro de 80 Gb. Tarjeta de red.

Las herramientas necesarias en cada puesto de trabajo se relacionan a continuación:

Tabla 2 Herramientas por estación de trabajo.

Herramientas	Estaciones de trabajo			
	1	2	3	4
Microsoft Project	X		X	
Subversion 1.3.0	X			
Tortoise	X	X	X	X
Rational Rose Enterprise Edition 2003	X	X	X	X
SQL Server 2000 (servidor)				X
SQL Server 2000 (cliente)	X	X	X	X

Microsoft .NET Framework 2.0	X	X	X	X
Microsoft Visual Studio 2005	X	X	X	X

2.7 Fundamentación de los patrones utilizados.

2.7.1 Patrones de Diseño

En el Modelo de Diseño de Predictor se proponen usar patrones pertenecientes a los GoF y GRASP, los que se pueden encontrar en cualquier bibliografía que trate ese tema. Es importante decir que se tuvieron en cuenta los patrones de asignación de responsabilidades (GRASP), para que el diseño tuviese un Bajo Acoplamiento, Alta Cohesión y que cada clase experta en algún tipo de información fuese la encargada de realizar las operaciones con la misma, logrando con eso un mejor funcionamiento del sistema.

Después de hacer un estudio de los patrones, analizando características y problemas que resuelven; así como las peculiaridades del sistema que se estaba diseñando, se seleccionaron patrones que ayudan en la solución a la problemática de diseñar el sistema haciendo uso de soluciones probadas satisfactoriamente en casos anteriores.

Patrones GRASP usados en el sistema:

2.7.1.1 Experto

Partiendo de que se conoce que una determinada clase posee una determinada información, se definió por este patrón donde colocar en cada clase las funcionalidades que necesitan de esa información, ya que esta clase seria nuestro experto en información (10).

Ejemplo: las clases que contienen una colección de elementos de otra, son las mejores candidatas a contener la definición e implementación de los métodos de manipulación de estos elementos como son las funcionalidades de insertar, buscar, eliminar estos de esta colección (10).

2.7.1.2 Creador

Se utilizó para detectar qué clase X debe crear elementos de otra clase Y basándose en que X debería:

a. Contener Y

b. Agregar Y

c. Tiene los datos de inicialización de Y

d. Registra a Y

e. Utiliza a Y

2.7.1.3 Alta Cohesión

Es necesario controlar la complejidad de cada clase utilizada en el sistema para mantener un buen comportamiento de las mismas, es por esto que las clases que fueron detectadas con muchas funcionalidades, se analizó la posibilidad de dividir en otras clases estas responsabilidades de manera que se repartiera equitativamente el peso de la complejidad, manteniendo además la coherencia del modelo de clases de diseño (10).

2.7.1.4 Bajo Acoplamiento

Si se desea que nuestro sistema genere componentes reusables, fácil de mantener y entender este es el patrón a tener presente en todo momento, para esto se trabajó cada subsistema de manera independiente (componentes para la descarga de información, procesamiento y configuración) y se definió una interfaz de comunicación con el exterior en cada uno de ellos (10).

2.7.1.5 Controlador

Se utilizó para la comunicación entre cada una de las capas del sistema, ejemplo de esto fue la creación de una clase Fachada que se encargara de la comunicación entre los eventos externos del sistema en la capa de presentación y los componentes de la capa de negocio (10).

Los patrones del GoF[6] usados en el sistema son:

2.7.1.6 Patrón Solitario (Singleton)

Con este patrón se garantiza que exista una única instancia de aquellas clases que se desee tener una sola en toda la aplicación. Reduce el espacio de nombres, es una mejora sobre las variables

[6] Design Patterns: Elements of reusable object oriented software. Conocido comúnmente como el libro GoF (Gang of four «en español – Banda de los cuatros»), cuyos autores son: Erich Gamma, Richard Helm, Ralph Johnson y John Vlissides.

globales. Es usado debido a la necesidad de trabajar con el mismo objeto en distintos momentos y distintos módulos. Específicamente, para realizar descargas es obligatorio configurar la conexión a Internet, este proceso lo realiza el Módulo Configuración, creando un objeto conexión, y esa misma instancia es llamada desde el Módulo Descarga para acceder a los datos del objeto y realizar la conexión para luego llevar a cabo la descarga de la información solicitada. También se usa en las clases fachadas de cada subsistema, garantizando una única instancia de ellas, esto suele hacerse cuando se usa el Patrón Fachada, su uso se explica a continuación. Las clases que implementan este patrón son: *MotorSolicitudes*, *BusquedaBasica*, *BusquedaAvanzada*, *GestorConfiguracion*, *ConfiguracionInternet y MotorProcesamiento*.

2.7.1.7 Patrón Fachada (Facade)

Proporciona una interfaz sencilla unificada para un conjunto de clases o subsistemas, siendo más fácil de usar. Permite reducir la complejidad y minimizar las dependencias, el acceso de los clientes a los subsistemas es por medio de la clase fachada, ella es la encargada de reenviar las peticiones a los objetos de los subsistemas, por lo que no se accede directamente a los mismos, ocultando la complejidad de ellos. Algo muy importante es que responde a unos de los Patrones GRASP, favorece a un Bajo Acoplamiento entre los clientes y los subsistemas, permitiendo variar las clases internas, de manera transparente a los clientes que las usan; favorece la división en capas de la aplicación. En este caso se crearon clases en los subsistemas que cumplen con esa función, logrando un Bajo Acoplamiento entre ellos, y toda la comunicación necesaria se realiza mediante ellas, las solicitudes, transferencia de objetos, envío de mensajes; sin saber de que manera se están procesando las peticiones. Como se dijo antes, estas clases fachadas también usan el Patrón Singleton, lo que permite el acceso global a ellas y la existencia de un único objeto de este tipo.

2.7.1.8 Patrón Observador (Observer)

Su propósito es definir una dependencia uno a muchos entre objetos, de modo que cuando un objeto cambia de estado, todos los que de él dependen son notificados y actualizados automáticamente. Se mantiene la dependencia entre los objetos sin necesidad de conocer al otro. Se usó por la existencia de una abstracción con dos aspectos, uno dependiente del otro, y se necesitaba encapsularlos en objetos separados para reutilizarlos de forma independiente. En Predictor la ventana que contiene los componentes que se configuran para la descarga, es un Observador de la clase encargada de realizar la descarga, que sería el Observado. La clase que

realiza la descarga (MotorSolicitudes) a medida que va realizando las descargas le notifica a los controles de usuarios: IUBusquedaBasica e IUBusquedaAvanzada, para que actualicen el ProgressBar que representa el porciento de descarga realizada hasta el momento, dichas clases implementan y heredan respectivamente la Interface IObservador y la clase Observado, quienes son los objetos abstractos y las primeras son los objetos concretos. El Patrón Observador también proporciona Bajo Acoplamiento al igual que el Fachada.

2.7.2 Patrón Arquitectónico.

2.7.2.1 Arquitectura en Capas

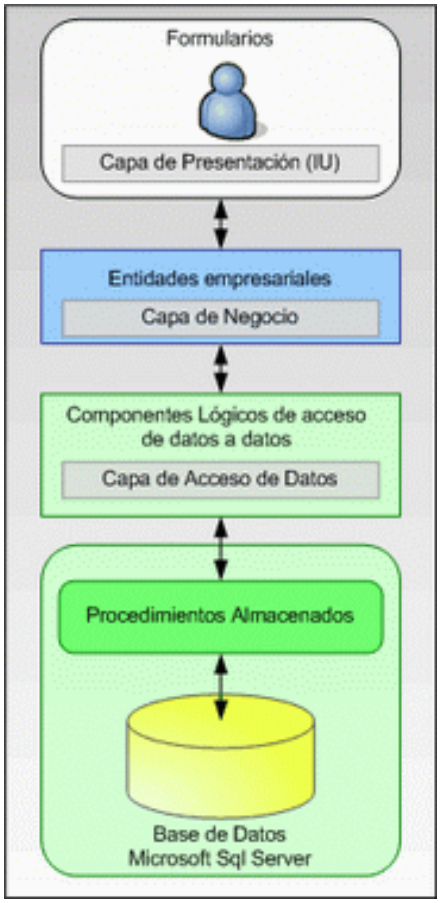
La arquitectura es fundamental en un proceso de desarrollo, pero realmente intervienen muchos factores que son determinantes en el correcto enfoque de la misma en un proyecto. Para garantizar que la arquitectura sea coherente con el sistema que se necesita construir hay que enfocarla en los requisitos que establece el cliente, garantizando esto, se logra un gran porciento de satisfacción del mismo, que es el principal objetivo en el desarrollo de un sistema. Un elemento importante dentro de la arquitectura es la forma en que se va a integrar todo el negocio del sistema, para lo cual se debe definir la estructura del software y establecer una estrategia de integración de las diferentes partes.

La propuesta para el sistema Predictor es la utilización de una arquitectura en capas, una de las más utilizadas internacionalmente por su fortaleza y coherencia con la abstracción, principio que introdujo Edsger Dijkstra. Mary Shaw y David Garlan definieron la arquitectura en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En la práctica, estas capas suelen ser entidades complejas, las cuales están compuestas por varios subsistemas o paquetes.

El objetivo de esta variante arquitectónica es aislar el negocio de la interfaz del usuario y el manejo de la persistencia. Con esta distribución modular estamos garantizando que el mantenimiento del sistema no sea complejo y por consiguiente no sean tantas las partes involucradas en un proceso de rectificación, incrementando la flexibilidad del conjunto.

Para el caso específico del sistema Predictor las tres capas que se definieron son: (1) Presentación, (2) Negocio y (3) Acceso a datos. La capa de *Presentación* no posee una complejidad considerable. Para la capa del *Negocio* se asocian tres subsistemas arquitectónicamente significativos, (1) Configuración, (2) Descarga y (3) ProcesamientoHTML; para

obtener más información acerca de la funcionalidad de los subsistemas, ver epígrafe 2.4.3. En el caso de la capa de *Acceso a datos*, ésta está definida para la gestión de la persistencia de la información ya procesada después que el usuario haga la descarga, pero será implementado para el próximo ciclo del proceso de desarrollo del software.



Capa de presentación: Es la que interactúa directamente con el usuario, captura la información entrada por éste (realiza un filtrado previo para comprobar que no hay errores de formato) y hace las peticiones a la capa inferior mostrando al usuario la respuesta proveniente de ésta. Únicamente se comunica con la capa de negocio.

Capa de 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.

Capa de Acceso a Datos: Contiene clases que interactúan con la base de datos y permiten, utilizando los procedimientos almacenados generados, realizar todas las operaciones con la base de datos de forma transparente para la capa de negocio.

2.8 Conclusiones

En este capítulo se trataron temas referentes al diseño arquitectónico del sistema Predictor, se realizó la descripción de la arquitectura mediante las diferentes vistas arquitectónicas de los modelos del sistema, se definió la estrategia de integración de los componentes definidos y de los ya existentes, se definieron los puestos de trabajo según los roles de cada integrante del equipo de desarrollo, y se justificaron los diferentes patrones utilizados tanto de diseño como el patrón arquitectónico en capas.

3 VALIDACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1 Introducción

En el presente capítulo se realiza un análisis de los resultados, tomando como principal basamento las métricas para determinar la calidad de un diseño de sistema. Varios actores proponen métricas para determinar la calidad de un diseño, en este capítulo se toman las más recomendadas a nivel mundial.

3.2 Resultado de las métricas de diseño arquitectónico.

En el presente epígrafe se dan solución a los cálculos planteados en el Capitulo 1, sobre las métricas propuestas para la evaluación de la arquitectura de software.

3.2.1 Métricas de Card y Glass

Esta métrica muestra la complejidad estructural ($S(i)$), la de datos ($D(i)$) y la del sistema ($C(i)$). Es esencial conocer de cada módulo la expansión y las variables de entrada y salida; para efectuar los cálculos con las fórmulas reflejadas en el Capitulo 1 (1.7.1). Para obtener la complejidad del sistema (entiéndase que i es un módulo, por lo tanto se calcula la complejidad del módulo como un sistema) se suma la complejidad estructural con la de los datos y si el valor es menor o igual que 32, hay carencia de complejidad; se manifiesta complejidad en el rango de mayor que 32 y menor o igual que 50 y muy complejo para aquellos valores por encima de 50.

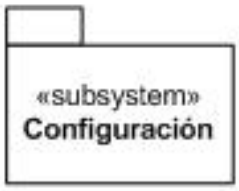
Nota: Para el cálculo de $S(i)$ y $D(i)$ es preciso conocer la expansión que estará reflejada por un número entero, puesto que la expansión de un módulo no puede ser expresada por un número real; lo mismo sucede con la cantidad de variables de entrada y salida para el cálculo de $D(i)$. Nótese que para la expresión $f_{out}^2(i)$ su resolución estaría dada de la siguiente manera:(1) $f_{out} = 3$; (2) $f_{out}^2(i) = (3)^2 = 9$ y es evaluado como no complejo para el caso de la expresión $S(i)$.

Tabla 3 Umbrales de complejidad.

Complejidad del sistema			
Complejidad	**S(i)**	**D(i)**	**C(i)**
No complejo	1,4,9,16,25	≤7	≤32
Complejo	36,49,64	>7 y ≤12	>32 y ≤50
Muy complejo	81…	>12	>50

A continuación se definen los módulos que presentan relaciones y se calculan para cada uno de estos, las complejidades anteriormente definidas:

3.2.1.1 Módulo Configuración

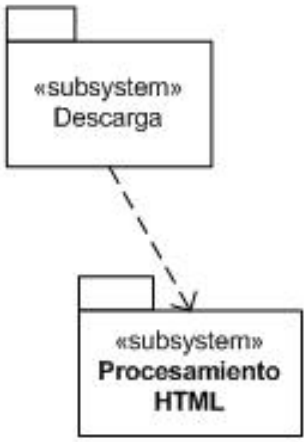


$$S(i) = f_{out}^2(i) = 0^2 = 0$$

$$D(i) = \frac{v(i)}{[f_{out}(i) + 1]} = \frac{4}{0+1} = 4$$

$$C(i) = S(i) + D(i) = 0 + 4 = 4$$

3.2.1.2 Módulo Descarga



$$S(i) = f_{out}^2(i) = 1^2 = 1$$

$$D(i) = \frac{v(i)}{[f_{out}(i) + 1]} = \frac{10}{1+1} = 5$$

$$C(i) = S(i) + D(i) = 1 + 5 = 6$$

3.2.1.3 Módulo ProcesamientoHTML

$$S(i) = f_{out}^2(i) = 0^2 = 0$$

$$D(i) = \frac{v(i)}{[f_{out}(i) + 1]} = \frac{4}{0+1} = 4$$

$$C(i) = S(i) + D(i) = 0 + 4 = 4$$

Resultado: A continuación se muestra la gráfica con un resumen de la sustitución de los valores correspondientes a los módulos previamente vistos, por tanto, se ha podido constatar, que para ninguno de los módulos existe una complejidad estructural considerable.

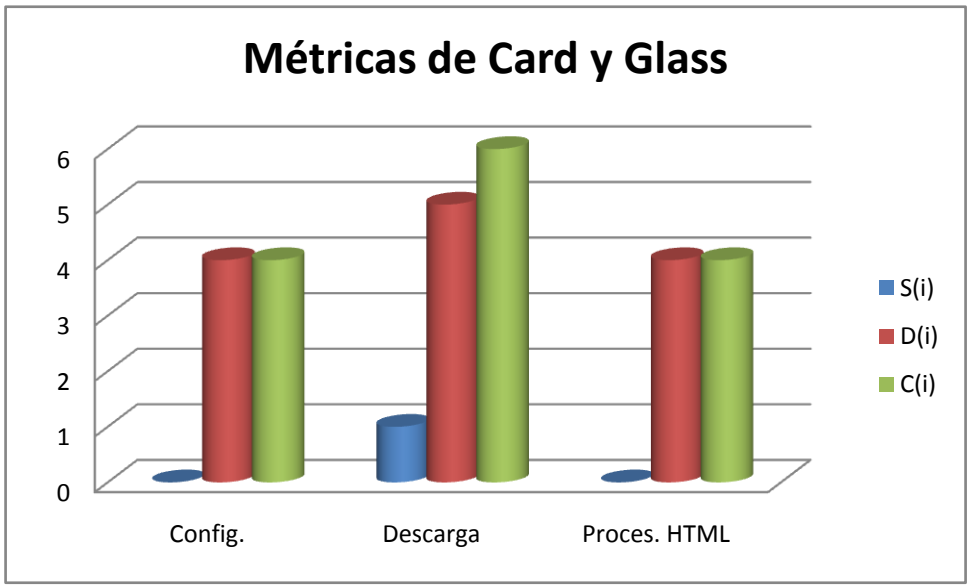


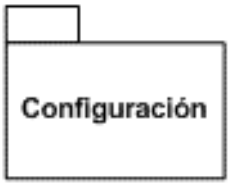
Figura 19 Valores de complejidad para la métrica de Card y Glass.

Es bien constatable, que el módulo más complejo del sistema es el de Descarga.

3.2.2 Resultados de la métrica de Henry y Kafura

Para desarrollar esta métrica, es necesario conocer una serie de valores esenciales, que al sustituirlos en la fórmula (ver epígrafe 1.7.2) aporta el conocimiento del esfuerzo de integración y pruebas que requiere el módulo. Para calcular el MHK durante el diseño puede emplearse el diseño procedimental para estimar el número de sentencias del lenguaje de programación del módulo i. El valor referente a $f_{in}(i)$ responde a la concentración del módulo i, es decir, es la cantidad de módulos de los que i recoge datos y para el $f_{out}(i)$ se cumple el mismo principio del epígrafe anterior «expansión» (epígrafe 3.2.1).

3.2.2.1 Módulo de Configuración

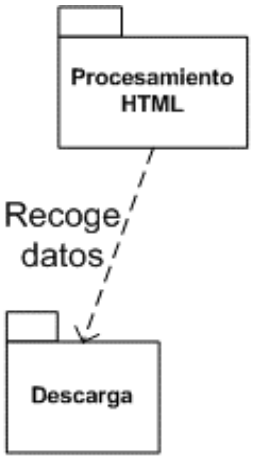


$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2 = 280 \times [0 + 0]^2 = 0$$

3.2.2.2 Módulo de Descarga

$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2 = 982 \times [1 + 1]^2$$
$$= 982 \times 4 = 3928$$

3.2.2.3 Módulo de ProcesamientoHTML



$$MHK = longitud(i) \times [f_{in}(i) + f_{out}(i)]^2 = 523 \times [1 + 0]^2 = 523 \times 1 = 523$$

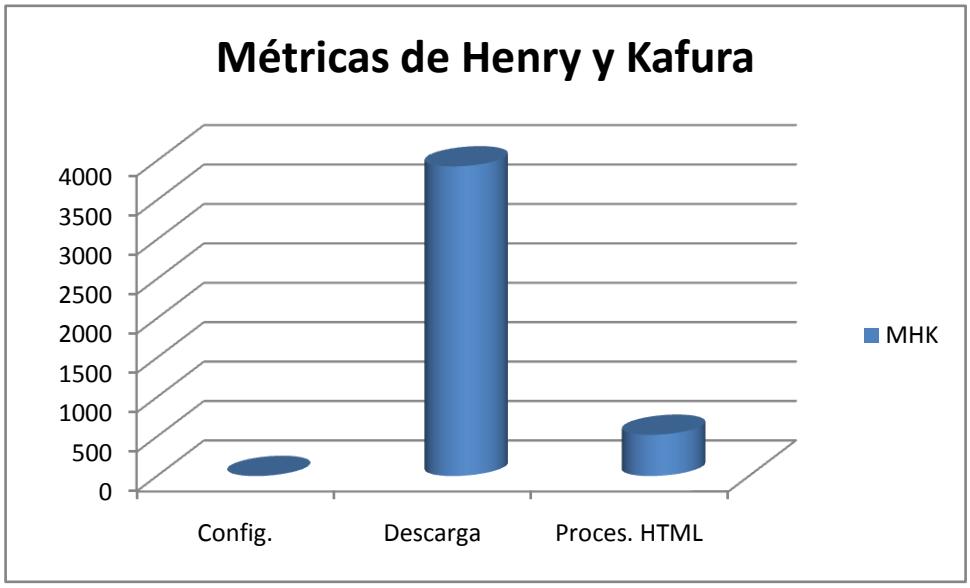


Figura 20 Valores para la métrica de Henry y Kafura.

Esta métrica refleja que el módulo Descarga es el más complejo y por tanto el que más esfuerzo de integración y pruebas va a requerir dentro del sistema.

3.3 Impacto de la arquitectura en el sistema Predictor.

En el desarrollo de un proyecto de software intervienen muchos factores y si bien la arquitectura tiene un papel protagónico no es la única que decide el éxito del proyecto, pero por esto no deja de ser uno de los pilares que sustenta y lleva al traste un buen producto final. Hace 15 años la arquitectura de software era una ciencia que estaba naciendo, sin embargo existían buenos

productos que tuvieron impacto en su época. En un entorno como el de la Consultoría del Ministerio de la Informática y las Comunicaciones, donde se realizan análisis de grandes volúmenes de información para hacer ejercicios de inteligencia de mercado, perfiles estratégicos y análisis de tendencias; solo por mencionar algunos de los más relevantes es necesario un sistema que automatice todo el proceso de descarga y procesamiento de la información, para realizar los trabajos antes mencionados se requiere del análisis de grandes volúmenes de información descargada desde Internet, entiéndase páginas HTML con información valiosa acerca de los datos bibliográficos de las patentes; información que debe ser procesada antes de comenzar el estudio.

El proceso de descarga de patentes en esta consultoría no está automatizado, y el proceso de filtrado para homogenizar la información y extraer datos relevantes es a través de macros que se le aplican a cada una de las páginas Web descargadas. Esto trae consigo demoras y en ocasiones, se producen incumplimientos en la realización de dichas tareas. Por este motivo, DELFOS propuso a la Universidad de las Ciencias Informáticas, específicamente a la Facultad 3, el desarrollo de un sistema informático de descarga y procesamiento de grandes volúmenes de información considerando las patentes como fuente fundamental. Una vez concluida la arquitectura propuesta para el sistema Predictor se cuenta con una plataforma que realiza búsquedas de patentes en Bases de Datos Públicas de Internet, procesa el contenido de dichas patentes y permite cargar esa información en una hoja de cálculo de Excel, reformatearla y hacerle análisis estadísticos, permite descargar y procesar 500 patentes en una hora, mientras que de forma manual sólo se podían procesar 50 en ese mismo intervalo de tiempo, además Predictor genera un fichero Procite.txt que facilita los análisis estadísticos en el sistema de igual nombre utilizado en la consultoría Delfos para el trabajo con las patentes. La arquitectura propuesta brinda soporte a la integración de nuevos o futuros componentes y facilita el mantenimiento del sistema.

3.4 Conclusiones

En el transcurso del capítulo se evaluaron las métricas del diseño arquitectónico, ilustrando cuales serían los módulos más complejos o de mayor envergadura tanto en complejidad del sistema como en esfuerzos de integración y pruebas. De igual manera se hizo una valoración del impacto que tuvo la arquitectura propuesta en el sistema, mostrando datos que evidencian los beneficios del mismo para el cliente, ganando en tiempo de procesamiento y calidad de la información procesada.

CONCLUSIONES GENERALES

Con la realización de este trabajo se arribaron a las siguientes conclusiones:

- Se obtuvo una arquitectura robusta, flexible y reusable que cumple con los requisitos funcionales y no funcionales del sistema.
- La metodología y las herramientas definidas permitieron que el proceso de desarrollo se realizara con una mayor organización y calidad.
- La descripción de la arquitectura mediante las 4 + 1 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.
- El patrón arquitectónico en Capas permitió que el sistema se estructurara de forma tal que el negocio estuviera aislado de la interfaz del usuario y el manejo de la persistencia, garantizando con esto la descomposición de la complejidad estructural y que el mantenimiento del sistema no sea complejo.
- Las métricas aplicadas con el objetivo de evaluar el diseño arquitectónico permitieron definir cuales módulos necesitan mayor esfuerzo de integración y pruebas.

RECOMENDACIONES

- Concluir el módulo de persistencia para optimizar las búsquedas y exportar hacia un fichero XML los datos de las patentes encontrados en la búsqueda.
- Realizar una aplicación Web que el sistema se conecte a ella y descargue las actualizaciones que pongan los desarrolladores periódicamente.

BIBLIOGRAFÍA

1. **González Hernández, Rolando.** *La información de marcas como indicador de innovación tecnológica.* Habana : s.n., 2006.

2. **Real Academia Española.** DICCIONARIO DE LA LENGUA ESPAÑOLA - Vigésima segunda edición . [En línea] Real Academia Española, 24 de Junio de 2004. [Citado el: 17 de 02 de 2007.] http://buscon.rae.es/draeI/.

3. **Billy Reynoso, Carlos.** Arquitectura de Software: Introducción a la Arquitectura de Software. [En línea] 11 de Junio de 2004. http://www.microsoft.com/spanish/msdn/arquitectura/roadmap_arq/intro.asp.

4. **Pressman, Roger S.** *Ingeniería del Software: Un enfoque práctico, Parte I.* La Habana : Félix Varela, 2005.

5. **Bass, Len, Clements, Paul y Kazman, Rick.** *Software Architecture in Practice.* Segunda edición. s.l. : Addison Wesley, 2003. 0-321-15495-9.

6. **Jacobson, Ivar, Booch, Grady y Rumbaugh, James.** *El Proceso Unificado de Desarrollo de Software, Volumen I.* Holguín : ARGRAF "José Miró Argenter", 2004.

7. *El prefabricado y la ampliacion de la UCLV: una arquitectura para la masivizacion de la educacion superior.* **ISLAS.** 133, 2002, Vol. 44.

8. **Billy Reynoso, Carlos y Kiccillof, Nicolás.** *Estilos y Patrones en la Estrategia de Arquitectura de Microsoft.* [Documento PDF] Buenos Aires : s.n., 2004.

9. **Hernández Suárez, José de Jesús.** *Arquitectura de software: Importancia de su ciclo de vida.* [Página Web] 2006.

10. **Larman, Craig.** *UML y patrones.* La Habana : Felix Varela, 2004.

11. **Camacho, Erika, Cardeso, Fabio y Nuñez, Gabriel.** *Arquitectura de Software. Guia de estudio.* 2004.

12. **Beznosov, Konstantin.** *Architecture of Information Enterprises: Problems and Perspectives.* [PDF] 1998.

13. **Fowler, Martin.** ¿Ha muerto el diseño? [En línea] Septiembre de 2004. [Citado el: 15 de Febrero de 2007.] http://www.programacionextrema.org/articulos/designdead.es.html .

14. **Naveda, Dr. J. Fernando.** *Architectural Design: Designing before Design.* [PDF] New York : s.n., 2006.

15. **Bastarrica, María Cecilia.** *Atributos de Calidad y Arquitectura del Software.* [PDF] Santiago de Chile : s.n., 2006.

16. **Shaw, Mary y Garlan, David.** *An Introduction to Software Architecture.* [PDF] Pittsburgh : s.n., 1994.

17. **Mahoney, Michael S.** *FINDING A HISTORY FOR SOFTWARE ENGINEERING.* [PDF] 2004.

18. **Kruchten, Philippe.** *Architectural Blueprints—The "4+1" View Model of Software Architecture.* [PDF] s.l. : IEEE Software, 1995.

19. **Shaw, Mary.** *The Coming-of-Age of Software Architecture Research.* [PDF] Pittsburgh : Institute for Software Research, International - Carnegie Mellon University, 2001. 1-412-268-2589.

20. **Montaldo, Diego Fernando.** *Patrones de Diseño de Arquitecturas Software Enterprise.* [PDF] Buenos Aires : Universidad de Buenos Aires (FIUBA), 2005.

21. **Losavio, Francisca, y otros.** *Quality Characteristics for Software Architecture.* [PDF] s.l. : ETH Zurich - JOT, 2003.

GLOSARIO DE TERMINOS

Abstracción

Las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporcionan así fronteras conceptuales definidas con nitidez en relación con la perspectiva del observador, la abstracción es uno de los elementos fundamentales del modelo de objeto.

Arquitectura

Conjunto de decisiones significativas acerca de la organización de un sistema software, la selección de los elementos estructurales a partir de los cuales se compone el sistema, y las interfaces entre ellos junto con su comportamiento, tal y como se especifica en las colaboraciones entre esos elementos, la composición de estos elementos estructurales y de comportamiento en subsistemas progresivamente mayores, y el estilo arquitectónico que guía esta organización: estos elementos y sus interfaces, sus colaboraciones y su composición. La arquitectura del software se interesa no solo por la estructura y el comportamiento, sino también por las restricciones y compromisos de uso, funcionalidad, funcionamiento, flexibilidad al cambio, reutilización, comprensión, economía y tecnología, así como por aspectos estéticos.

Artefacto

Pieza de información tangible, que puede ser usado en un proceso. Un artefacto puede ser un modelo, un elemento de un modelo, o un documento

Capa

Colección de categoría de clases o subsistemas al mismo nivel de abstracción

Centrado en la Arquitectura

En el contexto del ciclo de vida del software, significa que la arquitectura de un sistema se usa como un artefacto primordial para la conceptualización, construcción, gestión y evolución del sistema en desarrollo.

Clase

Representación abstracta de uno o más objetos, los términos clase y tipo suelen ser (no siempre) equivalentes, son conceptos ligeramente diferentes, en el sentido que la clase describe la estructura y el comportamiento de uno o más objeto(s) y un tipo solo define su comportamiento.

Componente

Una parte física y reemplazable de un sistema que proporciona la realización de una o más interfaces.

Descripción Arquitectónica

Descripción de la arquitectura del sistema, incluyendo las vistas arquitectónicas de los modelos. *Véase vista arquitectónica, vista arquitectónica del modelo de casos de uso, vista arquitectónica del modelo de análisis, vista arquitectónica del modelo de diseño, vista arquitectónica del modelo de despliegue, vista arquitectónica del modelo de implementación.*

Diagrama

La representación gráfica de un conjunto de elementos, usualmente representado como un grafo conectado de vértices (elementos) y arcos (relaciones).

Estilo Arquitectónico

Los sistemas que comparten una estructura de alto nivel y mecanismos clave similares se dice que tienen un estilo arquitectónico similar.

Fachada

Una fachada es un paquete estereotipado que no contiene más que referencias a elementos de modelos que pertenecen a otros paquetes. Utilizado para proporcionar una vista “pública” de algunos de los contenidos del paquete.

Hilo

Un flujo de control ligero que puede ejecutarse concurrentemente con otros hilos en el mismo proceso.

Interfaz

Una colección de operaciones que son utilizadas para especificar un servicio de una clase u otro componente.

Instancia

Una manifestación concreta de una abstracción; una entidad sobre la que pueden aplicarse un conjunto de operaciones y que tiene un estado que almacena el efecto de las operaciones; sinónimo de objeto.

Línea Base de la arquitectura

Conjunto de artefactos revisados y aprobados que representa un punto de acuerdo para la posterior evolución y desarrollo, y solamente puede ser modificado a través de un procedimiento formal, como la gestión de cambios y configuraciones.

Patrón

Solución común a un problema de un determinado contexto.

Patrón Arquitectónico

Patrón que define una cierta estructura o comportamiento, por lo general para la vista arquitectónica de un determinado modelo. Ejemplos son los patrones Capa, Cliente-Servidor, 3 Niveles y EntrePares, Three-tier y Peer-to-peer, cada uno de los cuales define una cierta estructura para el modelo de despliegue y sugiere también como deben ser asignados a sus nodos los componentes (la funcionalidad).

Proceso

Un flujo de control pesado que puede ejecutarse concurrentemente con otros procesos.

Sistema

Una colección de subsistemas organizados para llevar a cabo un propósito específico y descritos por un conjunto de modelos, posiblemente desde distintos puntos de vistas.

Subsistema

Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos.

Vista arquitectónica del modelo de análisis

Vista de la arquitectura de un sistema, abarcando las clases, paquetes y realizaciones de casos de uso del análisis; vista que fundamentalmente aborda el refinamiento y estructuración de los requisitos del sistema. La estructura de esta vista se preserva en la medida de lo posible cuando se diseña e implementa la arquitectura del sistema.

Vista arquitectónica del modelo de casos de uso

Vista de la arquitectura de un sistema abarcando los casos de uso significativos desde un punto de vista arquitectónico.

Vista arquitectónica del modelo de despliegue

Vista de la arquitectura de un sistema abarcando los nodos que forman la topología hardware sobre la que se ejecuta el sistema; vista que aborda la distribución, entrega e instalación de las partes que constituyen el sistema físico.

Vista arquitectónica del modelo de diseño

Vista de la arquitectura de un sistema, abarcando las clases, subsistemas, interfaces y realizaciones de casos de uso del diseño que forman el vocabulario del dominio de la solución del sistema; vista que abarca también los hilos y procesos que establecen la concurrencia y mecanismos de sincronización del sistema; vista que aborda los requisitos no funcionales, incluyendo los requisitos de rendimiento y capacidad de crecimiento de un sistema.

Vista arquitectónica del modelo implementación

Vista de la arquitectura de un sistema, abarcando los componentes usados para el ensamblado y lanzamiento del sistema físico; vista que aborda la gestión de la configuración de las versiones del sistema, constituida por componentes.