Universidad de las Ciencias Informáticas

Facultad 3

# Predictor: Sistema de descarga y procesamiento automatizado de patentes. Diseño del Sistema.

Trabajo de Diploma para optar por el título de

Ingeniero en Ciencias Informáticas

Autor

Alberto Limia Navarro

Tutores

Lic. Pedro Limia David

Ing. Yalice Gámez Batista

Ciudad de la Habana, Junio de 2007

**DECLARACIÓN DE AUTORÍA**

Declaro que soy el único autor de este trabajo y autorizo a la Facultad 3 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

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

| _____ | _____ | _____ |
|---|---|---|
| Alberto Limia Navarro | Ing. Yalice Gámez Batista | Lic. Pedro Limia David |

**OPINIÓN DEL TUTOR**

Durante la investigación realizada por el diplomante demostró gran independencia y responsabilidad en el cumplimiento de las distintas etapas de desarrollo de la misma. Las acciones planificadas fueron cumplidas exitosamente. Se evidenció su originalidad y creatividad teniendo en cuenta que logró abarcar de forma eficiente en el diseño del software las funcionalidades que le exigía el cliente. Como resultado de su labor el diplomante posee una carta de aceptación del cliente, expresando la contribución práctica del producto que se puso a disposición de Delfos.

Es importante destacar que la labor investigativa desplegada forma parte de presentaciones en forma de ponencias en Jornadas Científicas Estudiantiles, Concurso Provincial de Computación y Brigadas Técnicas Juveniles, en los cuales ha obtenido varios premios y el reconocimiento de sus compañeros y profesores. Las recomendaciones hechas por los distintos jurados de los eventos donde se han presentado partes de la investigación han sido oportunamente tenidas en cuenta.

La redacción del trabajo presenta buena coherencia y ortografía, lo que propicia la lectura de forma clara. Los contenidos tratados cumplen con el rigor científico requerido y están bien estructurados de acuerdo a las orientaciones recibidas.

Por lo antes expuesto considero que está listo para ejercer como Ingeniero en Ciencias Informáticas y le propongo una calificación de 5 puntos.

**AGRADECIMIENTOS**

*Agradezco a todos los que han contribuido de una manera u otra en mi formación profesional:*

*Especialmente A:*

*Nuestro Comandante por crear esta maravillosa escuela, y darme la posibilidad de formarme como Ingeniero en Ciencias Informáticas.*

*Mi tío Pedro por su ayuda y tiempo dedicado a la revisión de la tesis.*

*Mi tutora Yalice por la ayuda brindada.*

*Mis compañeros de proyecto: Luis Ernesto, Vlamir, Yaniet, Yoandry, Sergio, Mairelys, Mileisys y Yailin por todo el trabajo que hemos hecho juntos.*

*Mis compañeros de cuarto, especialmente a Juan Carlos, por su amistad y la incondicional ayuda brindada durante toda la carrera.*

*Todos los profesores que me han impartido clases durante estos cinco años.*

*Todos los que formaron parte del Proyecto Energía, quienes me enseñaron como se trabaja en equipo.*

*Aquellos que han formado parte de los grupos donde recibí clases.*

*Todos los que se preocuparon por mí y me ayudaron en momentos de enfermedad, especialmente a Somalia, Marcial, Zoila, Ale y Tía Mima. Sin ellos no hubiese logrado salir adelante.*

**DEDICATORIA**

*A todas las personas que de una forma u otra se han sacrificado y me han ayudado durante el transcurso de mi carrera.*

*Especialmente A:*

*Mis padres. Por todo el amor, cariño y comprensión que me brindan. Sin su guía no hubiese llegado hasta aquí, ustedes se merecen esto y mucho más.*

*Mi hermanita querida Isis Esther. Que te sirva de ejemplo para esforzarte y llegar a ser una buena profesional. Confío en Ti.*

*Mi primo Ale, has sido más que un hermano durante estos años.*

*Toda mi familia, que tanto me ayuda y se preocupa por mí.*

*"La familia es una joya única e invaluable"*

**RESUMEN**

La Consultoría del Ministerio de la Informática y las Comunicaciones (Delfos) tiene entre sus principales objetivos de trabajo el análisis de grandes volúmenes de información para realizar estudios de mercados, análisis de tendencias, perfiles estratégicos, y otros. Estas tareas las desarrollan gracias a la descarga y procesamiento de información que poseen sitios especializados en el seguimiento y publicaciones de patentes. Existen distintos softwares en el mercado mundial que automatizan esos procesos pero, por diversas razones, Delfos no puede usarlos y se ve en la necesidad de crear uno. Con ese fin surge el Proyecto Delfos, encargado de poner en marcha la construcción de Predictor, y en el juega un papel importante el diseño del sistema.

En este trabajo se tratan aspectos y temas importantes sobre diseño, que tienen que formar parte del conocimiento de los diseñadores de sistemas. Se realiza el Modelo de Diseño para un sistema informático que automatiza el proceso de búsqueda, descarga y procesamiento de la información necesaria para los análisis realizados en Delfos. Se aplican y tienen en cuenta los conceptos, principios y patrones de diseño para garantizar una correcta solución. Se elaboran los artefactos correspondientes al diseño, quienes sirven de entrada a la implementación. También se hace uso de algunas métricas aplicadas al diseño de software para evaluar y validar la calidad. Todo esto encaminado a lograr el funcionamiento estable, fiable y eficiente del sistema.

**INDICE**

## ÍNDICE DE FIGURAS

## ÍNDICE DE TABLAS

## INTRODUCCIÓN

*Antecedentes*

Décadas atrás, cuando no existía el auge actual de las nuevas tecnologías, la información que se manipulaba en el mundo era relativamente poca, y de difícil acceso para la inmensa mayoría de la sociedad. Tardaba años en darse a conocer a todo el planeta lo último con respecto a cualquier tema. Haciéndose casi imposible el seguimiento por parte de especialistas sobre temas de su interés.

Con el surgimiento de Internet y las Tecnologías de la Informática y las Comunicaciones se ha entrado en una nueva era, donde la información generada y almacenada en todo el mundo crece rápidamente. Según un estudio realizado por la Universidad de Berkeley la información que se genera y se registra crece a un 30% anual.

En la actualidad, debido a la gran competencia que existe en el mundo empresarial, toda compañía u empresa tiene que estar constantemente observando como se desarrolla su alrededor. Para tomar cualquier decisión se hace necesaria la búsqueda, análisis y procesamiento de grandes volúmenes de información relacionada con el tema, para de esta forma ser más competitivo y estar a la altura del resto de las compañías o empresas similares.

Las patentes son usadas como fuente de información para la investigación y análisis de tendencias. Su uso como indicador de innovación ha sido estudiado y ha alcanzado un alto nivel de madurez. La información obtenida de los documentos de patentes se refiere a la tecnología tratada en ellas, el inventor, su nacionalidad, las relaciones con otras patentes, el sector económico donde se crea la innovación, así como los campos y mercados cubiertos por las patentes.

Son documentos que contienen información muy valiosa desde el punto de vista legal, técnico y comercial. A partir de la cual se pueden determinar diferentes indicadores de innovación: año y país de registro, país de la invención, año y país de prioridad, año y país de publicación, documentos citados, países designados, año y país de concesión de la patente, familia de patente y clasificación de patentes.

Son importantes, no sólo para determinar indicadores de innovación de un país determinado, sino también para establecer estrategias de negocios de las compañías en los países menos desarrollados. Constituyen una fuente principal para la vigilancia tecnológica y la inteligencia competitiva.

Para ello se han desarrollado desde metodologías hasta grandes sistemas informáticos que se encargan de seleccionar la información útil y necesaria para los usuarios, resumiendo en pocos parámetros las características de grandes volúmenes de información. La visualizan de manera rápida y ordenada, usando

listas, redes, mapas, gráficos; mostrando las relaciones de las que se pueden extraer tendencias y patrones, ayudando a sacar conclusiones y tomar decisiones acertadas.

Existen varios softwares que realizan este tipo de trabajo, entre los más destacados están: Matheo Patent, Patent Lab, Vantage Point, Aureka, PAT-LIST-WPI, Bibexcel, entre otros. Cada uno de ellos tiene características específicas a la hora de tratar y mostrar la información. La mayoría de ellos han sido elaborados por pequeñas empresas con el propósito de mejorar el análisis de la información y el observatorio tecnológico.

Estos softwares se conectan a distintas bases de datos especializadas existentes en Internet, las cuáles almacenan grandes volúmenes de información sobre las patentes. Entre ellas se pueden encontrar tanto de acceso libre como propietarias. Entre las libres están: Worldwide, esp@cenet y la de la Oficina de Patentes y Marcas de EEUU. Cada una contiene información sobre las patentes generadas en distintas esferas y partes del planeta, siendo una fuente importante de información para cualquier empresa u organismo que desee llevar un estudio de las tendencias de patentamiento.

En Cuba existen varias empresas que llevan a cabo la vigilancia tecnológica, entre ellas está La Consultoría del Ministerio de la Informática y las Comunicaciones (Delfos). Entre sus principales objetivos de trabajo está el estudio de las iniciativas de patentamiento, con el fin de prever hacia donde se orientan las diferentes tecnologías, y así ayudar al desarrollo tecnológico dentro del país.

Dichos estudios se realizan descargando información sobre patentes de Bases de Datos existentes en Internet de forma manual. El trabajo se torna lento y engorroso, lo que trae consigo incumplimientos y mala calidad en el cumplimiento de sus objetivos de trabajo.

Delfos, para mejorar su desempeño le 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 fuentes fundamentales del mismo.

Para dar respuesta a está problemática surge el proyecto Delfos, compuesto por nueve desarrolladores. Fue escogido RUP para guiar el proceso de desarrollo del software. Quien plantea la existencia de varios roles dentro del equipo de desarrollo, los cuáles son asignados según competencias establecidas por los encargados de conformar el proyecto o por la experiencia demostrada por el personal en anteriores trabajos. En este caso se tuvo en cuenta la experiencia de los integrantes en proyectos anteriores donde habían ejercido dichos roles.

Uno de los roles que plantea RUP es: Diseñador del Sistema, el cuál juega un papel fundamental en el flujo de trabajo Análisis y Diseño junto con el Arquitecto del Sistema.

La tarea principal del Diseñador del Sistema es realizar el Modelo de Diseño, él cuál sirve de entrada para la disciplina de RUP que le sigue al Análisis y Diseño, la Implementación. Este modelo influye directamente en la calidad del producto final. Es aquí donde se tienen en cuenta todos los requerimientos del sistema, tanto los funcionales como los no funcionales. Por lo que es necesario que se realice correctamente, siguiendo todos aquellos conceptos, principios y patrones de diseño que puedan ser útil para resolver el problema que se esté tratando. Evitando con esto gastos innecesarios y garantizando un diseño estable, flexible, fiable y entendible para los programadores y demás desarrolladores.

*Problema*

¿Cómo diseñar un sistema de descarga y procesamiento automatizado de patentes publicadas en bases de datos de Internet que garantice que su funcionamiento sea estable, fiable, eficiente; que el diseño sea flexible y entendible para los demás desarrolladores?

*Hipótesis:*

Si se realiza un correcto diseño del sistema, se garantiza un sistema estable, eficiente, flexible, fácil de reparar y permitiría a los especialistas de Delfos llevar a cabo la observación tecnológica.

*Objeto de Estudio:*

Desarrollo de un sistema para la descarga y procesamiento de patentes procedentes de bases de datos en Internet.

*Campo de Acción:*

Diseño del Sistema Predictor para la descarga y procesamiento de patentes procedentes de bases de datos en Internet.

*Objetivos:*

El objetivo principal de este trabajo es realizar el diseño de un sistema de descarga y procesamiento automatizado de patentes publicadas en bases de datos de Internet que le permita a la Consultoría del Ministerio de la Informática y las Comunicaciones llevar a cabo la vigilancia tecnológica.

De este objetivo se derivan los siguientes objetivos específicos:

- Realizar el Modelo de Diseño usando patrones y que garantice un sistema poco complejo, reutilizable y eficiente.
- Elaborar los artefactos correspondientes al diseño.

- Demostrar la calidad del diseño propuesto mediante el uso de métricas.

*Tareas de Investigación:*

Realizar:

Análisis a softwares de descarga y procesamiento de información existentes en el mercado mundial.

Analizar conceptos y temas importantes para diseñar con calidad.

Analizar los Patrones de Diseño que se puedan aplicar en la solución.

Analizar métricas que se puedan aplicar para evaluar el diseño.

El trabajo está formado por tres capítulos donde se abarcan los siguientes temas:

En el primer capítulo se tratan los temas que fue necesario estudiar e investigar para realizar la propuesta de solución que se tiene como resultado final del trabajo. Se hace un estudio de las principales herramientas de procesamiento y descarga de información existente en el mercado. Se abordan temas claves para el diseño, tales como: conceptos, criterios, principios y patrones de diseño, los que tienen que formar parte del conocimiento de cualquier diseñador del sistema si desea realizar su trabajo con calidad. También se analizan algunas de las métricas más usadas para la evaluación del diseño, las que permiten validar la calidad del mismo.

En el segundo capítulo ya se plantea directamente la solución propuesta para dar respuesta a parte de los objetivos que se persiguen con este trabajo. Se habla de la arquitectura que conforma el sistema, los principales módulos con que cuenta, los patrones de diseño que fueron aplicados para garantizar una mejor solución, los subsistemas que conforman el Modelo de Diseño, así como las clases que contienen y las funcionalidades de las mismas. Se realizan los casos de usos que fueron designados para desarrollar en esta iteración del producto; se explica su funcionamiento, se presentan los diagramas de clases y de secuencia para cada uno de ellos. Es aquí donde se obtienen los artefactos generados por el diseñador del sistema.

En el tercer y último capítulo se aplican algunas de las métricas estudiadas en el primero al Modelo de Diseño realizado, permitiendo validar la solución y evaluar la calidad de la misma. Se aplica una métrica a nivel de arquitectura para verificar la complejidad de los módulos que forman al sistema. A nivel de clases se aplica la métrica Tamaño de Clase (TC) para conocer su tamaño, teniendo en cuenta la cantidad de atributos y operaciones; la de Árbol de Profundidad de Herencia (APH) para la profundidad de la herencia

entre las clases, y la de Relación entre las Clases, que tiene en cuenta las relaciones de uso que existen entre las distintas clases que conforman el diseño. Todas estas métricas permiten evaluar como influye el diseño en algunos parámetros que definen la calidad. Entre estos parámetros están: reutilización, complejidad, acoplamiento.

# CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

## Introducción

En la actualidad el estudio de las patentes se ha tornado importante para aquellos países o empresas en desarrollo. Gracias a la información que brindan se pueden tomar decisiones que posibilitan establecer pautas, estrategias y políticas a seguir para insertarse en el mercado mundial y seguir las tendencias actuales en determinados campos. Para la realización de estos estudios se han creado distintas herramientas que realizan análisis y descargas de patentes procedentes de distintas bases de datos en Internet. Cada una de ellas tiene características específicas que las hacen más o menos utilizables por aquellas empresas u organizaciones encargadas de llevar a cabo el observatorio tecnológico mediante el estudio de las tendencias de patentamiento. Entre las características esenciales que tienen estos sistemas están: manera de mostrar la información, lo hacen mediante gráficos de distintos tipos: mapas, gráficas en 2D, etc.; bases de datos a las que se conectan; parámetros seleccionados para el análisis.

A continuación se tratan algunos softwares especializados en la descarga y procesamiento de patentes; así como temas de interés a la hora de llevar a cabo el diseño del sistema que se propone como solución para la problemática existente en Delfos. Se habla sobre aspectos importantes a tener en cuenta para la elaboración de un diseño con calidad, sobre algoritmos de búsquedas en vectores, así como métricas que permiten evaluarlo.

## 1.1 Herramientas de Análisis y Descargas de Patentes

Hoy en el mundo existen varios softwares que son usados para realizar el seguimiento de las patentes. Ellos tienen características diferentes que los hacen más o menos poderosos.

### 1.1.1 Matheo Patent

Desarrollado por la empresa francesa Matheo Software. Está diseñado para la búsqueda, recuperación y análisis de patentes. Permite conectarse a la Oficina de Patentes de EEUU (USPTO) y a la Oficina Europea de Patentes (EPO).

Las principales funciones que realiza son: recuperación automática de los resultados de una búsqueda sin límite en el número de patentes descargadas, descarga de todas las informaciones relativas a una patente (ficha, resumen, reivindicaciones, estado legal, gráficos, documento pdf) y ordenación mediante pestañas, construcción de una base de datos local con todos los resultados obtenidos en varias estrategias de

búsqueda, almacenamiento de las estrategias de búsqueda y actualización selectiva de cada una de ellas para detectar novedades, sistema para puntuar las patentes de acuerdo con las preferencias de la persona usuaria según cuatro ejes (inversiones, formación, riesgo de infracciones e impacto en el negocio), análisis estadísticos y generación de representaciones gráficas (histogramas, matrices y redes), agrupación de los miembros de la misma familia de patentes en un solo registro, generación de informes personalizados, exportación de resultados en formato texto y *XML*. [1]

Es uno de los más sencillos de utilizar, incorpora cuatro tipos de visualizaciones que se pueden realizar con muchas combinaciones de las variables que caracterizan una patente. Tabla véase Anexo 1, Gráfico de Barras, véase Anexo 2, Matriz, véase Anexo 3 y Redes, véase Anexo 4.

### 1.1.2 Aureka

Desarrollado por Micropatent. Es una aplicación que puede usarse tanto en una estación de trabajo como en línea para la búsqueda, recuperación o análisis de la información. Esta herramienta agrupa una serie de utilidades que pueden ser implantadas y diseñadas según las necesidades del usuario. Algunos de los módulos mas destacables son los de Búsquedas, Organización y Agrupamiento, integra poderosas herramientas de Graficación de Relación (Aureka ThemeScape) y de Graficación de Referencias (Aureka Citation Tree) [2]

*Aureka ThemeScape*

Analiza documentos estadísticamente en términos dominantes, para conocer que tienen en común. Los temas se representan visualmente en mapas con aspecto cartográfico (transforma complejos documentos en un paisaje mediante alturas, valles, montañas, desniveles, etc.), identificando los conceptos predominantes y sus relaciones. Con él se pueden comparar compañías, competidores o tecnologías. [3] véase Anexo 5.

*Aureka Citation Tree*

Construye un árbol de citas a partir de una patente seleccionada. [4] véase Anexo 6.

### 1.1.3 PatentLab

Desarrollado por la compañía Wisdomain Inc. Es gratuito y se utiliza únicamente para analizar datos descargados de la base de datos Thomson Delphion. Respecto al análisis de patentes este software posibilita: la agrupación por familias de patentes, para evitar las duplicidades en el análisis; los conteos de los diferentes campos (tales como año de publicación, entidad solicitante o titular, inventor, temáticas. [5]

Es una herramienta que ofrece Delphion como complemento a su servicio de búsqueda de patentes, se emplea para el análisis de una serie de patentes seleccionadas o del resultado de una búsqueda. Realiza informes y gráficas de los campos de información bibliográfica de la patente. La generación de los informes y gráficas se realiza de forma casi instantánea, una vez obtenida la base de información, es una herramienta sencilla pero de gran utilidad para la realización de estudios rápidos de patentes. [2]

Cuenta con una interfaz sencilla y amigable, que permite crear rápidamente representaciones visuales con información de las patentes analizadas, ofreciendo la salida de estos datos en dos formas diferentes: tablas y gráficos (2D y 3D) e informes. [3] véase Anexo 7

### 1.1.4 VantagePoint

Desarrollado por Search Technologies, permite analizar rápidamente la búsqueda de resultados de bases de datos bibliográficas y literatura I+D. A diferencia de otras herramientas de minería de texto, VantagePoint está específicamente diseñada para interpretar búsquedas de resultados de bases de datos de ciencia y tecnología.

Sus características más relevantes son: la navegación rápida en grandes colecciones abstractas, la exhibición visual de relaciones mediante matrices de concurrencia o de factores, mapas tecnológicos, véase Anexo 8 y el uso y creación de filtros para reducir datos.

Más allá del análisis unidimensional (listas) y bidimensional (concurrencia de matrices), Vantage Point realiza análisis estadísticos multidimensionales para identificar grupos y relaciones entre conceptos, autores, países.

Permite la agrupación de patentes por familias, los recuentos por frecuencia de número de patentes por año, por organización, por autor, hasta realizar sofisticados análisis estadísticos, y mostrar toda la información relacionada con cualquier término, organización, año, etc., que le interese al usuario de manera interactiva y visual. [3]

## 1.2 Necesidad del software para Delfos

Los sistemas antes expuestos brindan ventajas y facilidades para su uso en el análisis de información, pero Delfos no puede valerse de ellos para realizar sus investigaciones.

La mayoría de los sistemas de análisis de patentes son softwares propietarios, se necesita pagar la licencia para usarlos, y por las características especiales que tiene el país y las leyes económicas que existen contra Cuba no es posible adquirirlos.

Algunos pocos son libres, 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 caso de los softwares propietarios.

Existen otros aspectos que impiden su uso por Delfos para realizar el trabajo relacionado con el seguimiento de las patentes. 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 entonces no se ajustan a las necesidades que tiene Delfos.

En Cuba existen otras organizaciones que también realizan el estudio de patentes, pero ninguno cuenta con una herramienta que automatice completamente el proceso. Dependen de sistemas de terceros para la realización de los análisis. Esto demuestra la necesidad de construir un software que automatice el proceso de descarga y análisis de la información en la Consultoría del Ministerio de la Informática y las Comunicaciones.

Ellos tienen la necesidad de descargar información de distintas bases de datos existentes en prácticamente todo el mundo, seleccionando parámetros específicos de las patentes, los cuáles tienen en cuenta a la hora de realizar sus análisis y estudios, sin tener que ajustarse a análisis predeterminados por otros. También le hace falta una base de datos local que permita almacenar los resultados de las búsquedas realizadas para mejorar el proceso.

Para cumplir con las necesidades de Delfos se llevó a cabo el proyecto, seleccionando como la metodología de desarrollo a usar RUP, quien tiene entre sus principales flujos de trabajo el Análisis y Diseño, jugando un papel fundamental el Diseñador del Sistema, que es el encargado de realizar una serie de actividades que generan artefactos importantes usados más adelante por los demás integrantes del proyecto para la implementación del sistema.

## 1.3 El Diseño

### 1.3.1 ¿Qué es el diseño?

El diseño es una representación significativa de ingeniería de algo que se va a construir. Se puede hacer el seguimiento basándose en los requisitos del cliente, y al mismo tiempo la calidad se puede evaluar y cotejar con el conjunto de criterios predefinidos para obtener un diseño "bueno". En el contexto de la ingeniería del software, el diseño se centra en cuatro áreas importantes de interés: datos, arquitectura, interfaces y componentes… [6]

El diseño es uno de lo pilares fundamentales en la ingeniería del software. Es una de las actividades técnicas necesarias para la elaboración del software. Entre las tareas fundamentales del diseño están: producir el diseño de datos, diseño arquitectónico, diseño de interfaz y diseño de componentes.

Cuando se diseñan Sistemas Informáticos es indispensable hacerlo de forma correcta. El diseño propuesto tiene que cumplir a cabalidad con los requerimientos del sistema. Es la única forma de convertir exactamente los requisitos de un cliente en un producto o sistema de software finalizado…[6]; debe ser capaz de facilitar las mejoras del software, tiene que ser entendible por otros profesionales de la especialidad, servir como guía para los demás pasos de la ingeniería de software, permitir la comprobación del sistema fácilmente.

Sin un buen diseño es imposible garantizar la calidad del producto final, por lo que es necesario el mayor esfuerzo posible en esta etapa, se debe garantizar buenas técnicas de diseño, es recomendable hacer uso de aquellas soluciones que han sido probadas en casos anteriores con resultados satisfactorios y que se ajustan al sistema que se esté construyendo.

### 1.3.2 Calidad del Diseño

La importancia del diseño del software se puede describir con una sola palabra –calidad--…[6]

No existe una manera concreta de definir entre varios diseños diferentes cual está mejor que otro; pero si existen ciertas propiedades claves que permiten evaluar la calidad de los mismos.

Propiedades claves:

- *Extensibilidad:* El diseño debe ser capaz de soportar nuevas funciones. Aunque sea perfecto en todos los demás aspectos, un sistema que no muestre disposición a integrar el más ligero cambio o perfeccionamiento resulta inservible. Quizás no haya necesidad de añadir nuevas funciones, pero

siempre es posible que se produzcan alteraciones en el dominio del problema que exijan introducir cambios en el programa. Siempre se tiene que diseñar pensando en futuras mejoras y la agregación de nuevas clases, componentes, ficheros, funcionalidades, etc.

- *Fiabilidad:* El sistema debe tener un comportamiento fiable, debe realizar todas las funcionalidades correctamente y de la forma prevista por el usuario. Dándole la misma importancia a todas las funcionalidades, tanto las complejas como las sencillas, permitiendo que el usuario comprenda fácilmente su funcionamiento.
- *Eficiencia:* El consumo de recursos por parte del sistema debe ser racional. Los recursos más específicos son el tiempo de ejecución y la cantidad de memoria que consume el software.

Si en el momento de diseñar se tienen en cuenta estas propiedades, entonces quedaría garantizado un buen diseño. Pero también existen otros aspectos importantes que influyen grandemente en la calidad del producto.

Existen criterios técnicos que también definen un buen diseño:

- Un diseño debe presentar una organización jerárquica que haga un uso inteligente del control entre los componentes del software.
- El diseño debe ser modular, es decir, se debe hacer una partición lógica del software en elementos que realicen funciones y subfunciones especificas.
- Un diseño debe contener abstracciones de datos y procedimientos.
- Debe producir módulos que presenten características de funcionamiento independiente.
- Debe conducir a interfaces que reduzcan la complejidad de las conexiones entre los módulos y el entorno exterior.
- Debe producir un diseño usando un método que pudiera repetirse según la información obtenida durante el análisis de requisitos de software.

### 1.3.3 Principios del Diseño

Roger Pressman en su libro:”Ingeniería del Software. Un enfoque práctico” plantea una serie de principios básicos que se tienen que tener en cuenta en el momento de diseñar, ellos garantizan el correcto funcionamiento del sistema.

- En el proceso de diseño no deberá utilizarse “orejas”[1].
- El diseño deberá poderse rastrear hasta el modelo de análisis.
- El diseño no deberá inventar nada que ya esté inventado.
- El diseño deberá minimizar la distancia intelectual entre el software y el problema como si de la misma vida real se tratara.
- El diseño deberá presentar uniformidad e integración.
- El diseño deberá estructurarse para admitir cambios.
- El diseño deberá estructurarse para degradarse poco a poco.
- El diseño no es escribir código y escribir código no es diseñar.
- El diseño deberá evaluarse en función de la calidad mientras se va creando, no después de terminado.
- El diseño deberá revisarse para minimizar los errores conceptuales.

Cuando estos principios son aplicados correctamente, entonces el diseño creado muestra los factores de calidad internos y externos. Los factores externos son aquellas propiedades del software que se observan fácilmente: velocidad, fiabilidad, grado de corrección, usabilidad. Los internos son importantes para los ingenieros y personal encargo de realizar el proyecto, conducen a un diseño de alta calidad. Para lograrlos es necesario comprender correctamente los conceptos básicos del diseño.

### 1.3.4 Conceptos del Diseño

Los conceptos básicos del diseño son de mucha importancia para cualquier diseñador, debido a que proporcionan la base para aplicar los métodos de diseño. Al hacer un programa se puede lograr que funcione, pero es necesario lograr que lo haga correctamente. Eso lo permiten los conceptos de software. Pressman en una de sus publicaciones trata una serie de conceptos importantes.

*Abstracción*

Se pueden exponer distintos niveles de abstracción, en dependencia del problema que se esté resolviendo. El nivel más alto es empleando el lenguaje del entorno del problema, en los inferiores a este

[1] Un buen diseñador deberá tener en cuenta enfoques alternativos, juzgando todos los que se basan en los requisitos del problema, los recursos disponibles para realizar el trabajo y los conceptos de diseño. [6]

se toma una orientación procedimental, hasta llegar al nivel más bajo, donde se establece la solución para implementarse.

Cada paso en el proceso de desarrollo de software es un refinamiento en el nivel de abstracción. Así desde el Negocio, los Requisitos, el Análisis y Diseño y demás disciplinas de RUP se van reduciendo los niveles de abstracción hasta llegar al más bajo, cuando se genera el código fuente.

Existen tres tipos fundamentales de abstracciones que se crean a la hora de adentrarse en los distintos niveles de abstracción, ellos son: abstracción procedimental, abstracción de datos y abstracción de control.

*Refinamiento*

El desarrollo de un programa se realiza refinando sucesivamente los niveles de detalle procedimentales [6]. El refinamiento es un proceso de elaboración, se comienza a un alto nivel de abstracción y se va trabajando sobre las sentencias originales, llegando a un alto nivel de detalle a medida que se van realizando los distintos refinamientos. Todos los pasos del refinamiento implican decisiones en el diseño.

*Modularidad*

Cuando un equipo de desarrollo está encargado de construir un sistema grande, comúnmente se dividen en distintas partes, llamadas módulos, que al integrarlos cumplen con los requerimientos que plantea el cliente. La modularidad es la encargada de dividir el problema en pequeñas partes, pero hay que prestar atención al dividir en módulos, debido a que un número mayor de módulo implicaría un menor tamaño de los mismos, pero un mayor costo de desarrollo. Existen cinco criterios que permiten definir un sistema modular efectivo:

- Capacidad de descomposición modular.
- Capacidad de empleo de componentes modulares.
- Capacidad de compresión modular.
- Continuidad modular.
- Protección modular.

*Arquitectura de Software*

Es la estructura del software, la estructura jerárquica de los componentes, módulos, la manera en que ellos interactúan y las estructuras de datos que usan los distintos componentes. Entre los objetivos del diseño está generar distintas vistas arquitectónicas del sistema, que son importante para la comprensión del mismo por los demás integrantes del equipo de desarrollo.

*Jerarquía de Control*

O denominada también como estructura de programa, representa la organización de los componentes e implica una jerarquía de control. No se puede aplicar a todos los estilos arquitectónicos. Para arquitecturas de llamada y de retorno el diagrama más común es el de forma de árbol. Para la elaboración de un software orientado a objetos no se aplican esas medidas.

*División Estructural*

Cuando el estilo arquitectónico es jerárquico, la estructura se puede dividir tanto horizontal como vertical. La división horizontal proporciona diferentes ventajas: software más fácil de probar, conduce a un software más fácil de mantener, propaga menos efectos secundarios, software más fácil de ampliar. En la vertical o factorización, como también se le suele llamar, los módulos de nivel superior llevan a cabo las funciones de control, y los inferiores son los que realizan todas las tareas de entradas, proceso y salida.

*Ocultación de Información*

Plantea que los módulos se caracterizan por las decisiones de diseño, donde cada módulo oculta al otro, los módulos tienen que quedar diseñados de tal manera que la información que esté dentro de un módulo sea inaccesible para aquellos que no la usan. Significa que se puede conseguir una modularidad efectiva teniendo los módulos de manera independiente, pero comunicándose entre sí y solo intercambiando la información necesaria para el correcto funcionamiento del sistema.

## 1.4 Patrones de Diseño

Uno de los principios de diseño plantea que no se debe inventar nada que ya esté inventado…Los sistemas se construyen utilizando un conjunto de patrones de diseño, muchos de los cuales

probablemente ya se han encontrado antes. Estos patrones deberán elegirse siempre como una alternativa para reinventar…[6]

Una buena técnica de diseño consiste en usar patrones. Un patrón de diseño es una solución a un problema de diseño no trivial que es efectiva (ya se resolvió el problema satisfactoriamente en ocasiones anteriores) y reutilizable (se puede aplicar a diferentes problemas de diseño en distintas circunstancias).

El diseño con patrones tiene como principal objetivo agrupar una colección de soluciones de diseño que son válidas en distintos contextos y que han sido aplicadas con éxito en otras ocasiones. Son soluciones de sentido común que tiene que formar parte del conocimiento de cualquier diseñador. Facilitan la comunicación entre diseñadores. Facilita el aprendizaje al programador inexperto, estableciendo parejas de problema-solución.

Los patrones de diseño proponen una forma de reutilizar la experiencia de los desarrolladores, para ello clasifica y describe formas de solucionar problemas que ocurren de forma frecuente en el desarrollo. Por tanto está basado en la recopilación del conocimiento de los expertos en desarrollo de software.

### 1.4.1 Historia de los Patrones de Diseño

El primero en hablar de patrones fue Christopher Alexander, quien en los años 70 publicó varios libros aplicando la idea de patrones a la arquitectura. Refiriéndose a los patrones dijo:

“Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces sin hacerlo si quiera dos veces de la misma forma”.

Las ideas de Alexander desencadenaron un vertiginoso desarrollo de la aplicación de patrones a distintas ciencias y ramas. La informática no se quedó detrás y de esa manera comenzaron a surgir los primeros patrones que más tarde servirían de guía para los que hoy son conocidos y aplicados por la comunidad de desarrolladores.

Los primeros fueron Ward Cummingham y Kent Beck quienes adaptaron por primera vez las ideas de Alexander a la Ingeniería de Software. Crearon cinco patrones para el diseño de las interfaces: Window per Task, Few Panes, Standard Panes, Nouns and Verbs, Short Menu.

A partir de ahí se siguieron realizando una serie de trabajos e investigaciones relacionados con los patrones, destacándose en 1991 la Tesis de Doctorado de Erich Gamma, primer trabajo publicado sobre los patrones aplicados al desarrollo de software, originalmente en alemán.

Así comenzó la década del 90, y una serie de publicaciones siguieron a Gamma. James Coplien, publica Advanced C++ Programming Styles and Idioms. En 1995 publican el libro Design Patterns, escrito por: Gamma, Helm, Johnson, Vlissides en el cuál se plantean una serie de patrones conocidos como los patrones Gof. En ese mismo año Peter Coad, publica Object Models – Patterns, Strategies and Applications, también Douglas Schmidt, publica artículos con varios patrones para sistemas distribuidos: Reactor, Aceptor, Connector. En 1996 Buschmann, publica: System of Patterns – Pattern Oriented Software Architecture. En 1998 Brown, Malveau, McCormick III, Mowbray, publican AntiPatterns – Refactoring Software, Architectures and Project in Crisis.

Esto es una breve historia de algunas de las publicaciones y hechos más importantes hasta la fecha en lo que respecta a los patrones. Ahora se verán los Patrones GRASP (General Responsability Asignment Software Paterns) y los Gof, que son de los más conocidos y usados por la comunidad de desarrolladores.

### 1.4.2 Patrones GRASP

Los patrones GRASP describen los principios fundamentales de la asignación de responsabilidades a objetos, expresados en forma de patrones. [7] GRASP es un acrónimo que significa General Responsability Asignment Software Paterns (patrones generales de software para asignar responsabilidades)… [7]

En los patrones GRASP se encuentran algunos de los principios de diseño, estos son aplicados a la hora de elaborar los diagramas de interacción y de asignar las responsabilidades a las clases y objetos que conforman el sistema que se esté diseñando. Es importante recalcar que responsabilidad no equivale a método…los métodos se ponen en práctica para cumplir con las responsabilidades… [7]

Existen nueve patrones GRASP: Experto, Creador, Alta Cohesión, Bajo Acoplamiento, Controlador, Polimorfismo, Fabricación Pura, Indirección y No Hables con Extraños. Cada uno de ellos indica la solución que se le debe dar a una problemática determinada. En la siguiente tabla se muestran estos patrones con los problemas que resuelven y las soluciones que le dan.

Tabla 1: Patrones GRASP

| Nombre | Solución | Problema que resuelve |
|---|---|---|
| Experto | Asignar una responsabilidad al experto en información: la clase que | ¿Cuál es el principio fundamental en virtud del cual se asignan las |

| | | |
|---|---|---|
| | cuenta con la información necesaria para cumplir la responsabilidad. | responsabilidades en el diseño orientado a objetos? |
| Creador | Asignarle a una clase la responsabilidad de crear una instancia de otra clase en casos diferentes. | ¿Quién debería ser responsable de crear una nueva instancia de alguna clase? |
| Bajo acoplamiento | Asignar una responsabilidad para mantener bajo acoplamiento. | ¿Cómo dar soporte a una dependencia escasa y a un aumento de la reutilización? |
| Alta cohesión | Asignar una responsabilidad de modo que la cohesión siga siendo alta. | ¿Cómo mantener la complejidad dentro de límites manejables? |
| Controlador | Asignar la responsabilidad del manejo de un mensaje de los eventos de un sistema a una de las clases. | ¿Quién debería encargarse de atender un evento del sistema? |
| Polimorfismo | Cuando por el tipo varían las alternativas o comportamientos afines, las responsabilidades del comportamiento se asignarán mediante operaciones polimórficas a los tipos en que el comportamiento presenta variantes. | ¿Cómo manejar las alternativas basadas en el tipo? ¿De que manera crear componentes de software conectables? |
| Fabricación Pura | Asignar un conjunto altamente cohesivo de responsabilidades a una clase artificial que no representa nada en el dominio del problema. | ¿A quién asignar la responsabilidad cuando se está desesperado y no se quiere violar los patrones Alta Cohesión y Bajo Acoplamiento? |
| Indirección | Se asigna la responsabilidad a un objeto intermedio para que medie entre otros componentes o servicios, y estos no terminen directamente acoplados. | ¿A quién se asignarán las responsabilidades a fin de evitar el acoplamiento directo? ¿De qué manera se desacoplarán los objetos de modo que se obtengan un Bajo Acoplamiento |

| | | |
|---|---|---|
| | | y se observe un alto potencial de reutilización? |
| No Hables con Extraños | Se asigna la responsabilidad a un objeto directo del cliente para que colabore con un objeto indirecto, de modo que el cliente no necesite saber nada del objeto indirecto | ¿A quién asignar las responsabilidades para evitar conocer las estructuras de los objetos indirectos? |

Es importante dominar estos patrones,…constituyen el fundamento de cómo se diseña un sistema orientado a objetos…[7]…no introducen ideas nuevas, son una mera codificación de los principios básicos más usados…[7]. Cada uno de ellos soluciona problemas que contribuyen a que los sistemas sean más robustos, más flexibles, fáciles de mantener y extender. Los patrones GRASP no compiten con los patrones de diseño, sino que guían para ayudar a encontrar los patrones de diseños adecuados para aplicar al sistema que se esté elaborando.

### 1.4.3 Patrones Gof

Estos patrones fueron publicados por Gamma, Helm, Jonson y Vlissides en su libro Design Patterns: Elements of Reusable Object-Oriented Software" (Gang of Four).

Los patrones de diseño tienen como características:

- Son soluciones concretas. Proponen soluciones a problemas concretos, no son teorías genéricas.
- Son soluciones técnicas. Indican resoluciones técnicas basadas en Programación Orientada a Objetos (POO). En ocasiones tienen más utilidad con algunos lenguajes de programación y en otras son aplicables a cualquier lenguaje.
- Se utilizan en situaciones frecuentes. Debido a que se basan en la experiencia acumulada al resolver problemas reiterativos.
- Favorecen la reutilización de código. Ayudan a construir software basado en la reutilización, a construir clases reutilizables. Los propios patrones se reutilizan cada vez que se vuelven a aplicar.

- El uso de un patrón no se refleja en el código. Al aplicar un patrón, el código resultante no tiene por que delatar el patrón o patrones que lo inspiró. No obstante últimamente hay múltiples esfuerzos enfocados a la construcción de herramientas de desarrollo basados en los patrones y frecuentemente se incluye en los nombres de las clases el nombre del patrón en que se basan facilitando así la comunicación entre desarrolladores.
- Es difícil reutilizar la implementación de un patrón. Al aplicar un patrón aparecen clases concretas que solucionan un problema concreto y que no será aplicable a otros problemas que requieran el mismo patrón.

Se clasifican según su propósito en:

*Patrones de Creación*

Se encargan de la creación de instancias de los objetos. Abstraen la forma en que se crean los objetos, permitiendo tratar las clases a crear de forma genérica, dejando para después la decisión de que clase crear o cómo crearla.

Según donde se tome dicha decisión se pueden clasificar los patrones de creación en: patrones de creación de clases (la decisión se toma en los constructores de las clases y usan la herencia para determinar la creación de las instancias)

Los patrones de creación son:

- *Abstract Factory (Fábrica Abstracta):* Permite trabajar con objetos de distintas familias de manera que las familias no se mezclen entre sí y haciendo transparente el tipo de familia concreta que se esté usando.
- *Builder (Constructor virtual):* Abstrae el proceso de creación de un objeto complejo, centralizando dicho proceso en un único punto.
- *Factory Method (Método de fabricación):* Centraliza en una clase constructora la creación de objetos de un subtipo de un tipo determinado, ocultando al usuario la casuística para elegir el subtipo que se crea.
- *Prototype (Prototipo):* Crea nuevos objetos clonándolos de una instancia ya existente.
- *Singleton (Instancia única):* Garantiza la existencia de una única instancia para una clase y la creación de un mecanismo de acceso global a dicha instancia.

*Patrones Estructurales*

Son los que plantean las relaciones entre clases, las combinan y forman estructuras mayores. Tratan de conseguir que los cambios en los requisitos de la aplicación no ocasionen cambios en las relaciones entre los objetos. Lo fundamental son las relaciones de uso entre los objetos, y éstas están determinadas por las interfaces que soportan los objetos. Estudian cómo se relacionan los objetos en tiempo de ejecución. Sirven para diseñar las interconexiones entre los objetos.

Los patrones estructurales son:

- *Adapter (Adaptador):* Adapta una interfaz para que pueda ser utilizada por una clase que de otro modo no podría utilizarla.
- *Bridge (Puente):* Desacopla una abstracción de su implementación.
- *Composite (Objeto compuesto):* Permite tratar objetos compuestos como si de uno simple se tratase.
- *Decorator (Envoltorio):* Añade funcionalidad a una clase dinámicamente.
- *Facade (Fachada):* Provee de una interfaz unificada simple para acceder a una interfaz o grupo de interfaces de un subsistema.
- *Flyweight (Peso ligero):* Reduce la redundancia cuando gran cantidad de objetos poseen idéntica información.
- *Proxy:* Mantiene un representante de un objeto.

*Patrones de Comportamiento*

Plantea la interacción y cooperación entre las clases. Los patrones de comportamiento estudian las relaciones entre llamadas entre los diferentes objetos, normalmente ligados con la dimensión temporal.

Los patrones de comportamiento son:

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

- *Command (Orden):* Encapsula una operación en un objeto, permitiendo ejecutar dicha operación sin necesidad de conocer el contenido de la misma.
- *Interpreter (Intérprete):* Dado un lenguaje, define una gramática para dicho lenguaje, así como las herramientas necesarias para interpretarlo.
- *Iterator (Iterador):* Permite realizar recorridos sobre objetos compuestos independientemente de la implementación de estos.
- *Mediator (Mediador):* Define un objeto que coordine la comunicación entre objetos de distintas clases, pero que funcionan como un conjunto.
- *Memento (Recuerdo):* Permite volver a estados anteriores del sistema.
- *Observer (Observador):* Define una dependencia de uno-a-muchos entre objetos, de forma que cuando un objeto cambie de estado se notifique y actualicen automáticamente todos los objetos que dependen de él.
- *State (Estado):* Permite que un objeto modifique su comportamiento cada vez que cambie su estado interno.
- *Template Method (Método plantilla):* Define en una operación el esqueleto de un algoritmo, delegando en las subclases algunos de sus pasos, esto permite que las subclases redefinan ciertos pasos de un algoritmo sin cambiar su estructura.
- *Visitor (Visitante):* Permite definir nuevas operaciones sobre una jerarquía de clases sin modificar las clases sobre las que opera.
- *Strategy (Estrategia):* Permite disponer de varios métodos para resolver un problema y elegir cuál utilizar en tiempo de ejecución.

## 1.5 Algoritmos de Búsquedas

Los algoritmos de búsqueda tienen como finalidad localizar un elemento dado dentro de una estructura de datos o determinar su inexistencia. Dependen de la estructura en la que se esté trabajando. Existen un gran número de ellos y cada uno tiene características y fines específicos. Los utilizados para buscar en grafos, árboles y listas, por solo mencionar algunas estructuras de datos, no son los mismos. En este epígrafe se tratan algunos de los que se pueden aplicar cuando se está buscando en una lista.

### 1.5.1 Búsqueda Binaria o Dicotómica

Tiene que cumplirse como precondición que la lista sobre la que se va a aplicar tiene que estar ordenada. Consiste en empezar buscando por la mitad del vector, se compara con el elemento que está en esa posición, si coincide se encontró. En caso de ser diferentes entonces se compara, si el elemento que se busca es menor que el que está en el centro, se procede a realizar la misma operación pero solamente con la sublista que se encuentra a la izquierda, y si es mayor se hace el mismo proceso pero hacia la derecha, y así respectivamente hasta encontrar el elemento. Esto se realiza con tres variables auxiliares que indican donde está el inicio, el medio y el final de la sublista en la que se está buscando. En comparación con otros algoritmos es rápido.

Existe una modificación de este algoritmo que es llamado Búsqueda por Interpolación, se diferencia en que no se comienza a buscar por el elemento del centro de la lista, pero como el anterior, los datos tienen que estar ordenados y la distribución ser uniforme.

### 1.5.2 Búsqueda Secuencial

Es un algoritmo de búsqueda sencillo recorre la lista desde el primer elemento hasta el último y si encuentra el valor buscado retorna su posición. No tiene en cuenta si los datos están ordenados, pero sin embargo, en el caso de estar ordenados resulta ineficiente. Realiza siempre el mismo número de iteraciones independientemente de la posición en que se encuentre el valor buscado.

Una mejora a este algoritmo es la Búsqueda Secuencial con Centinela, el cuál finaliza la búsqueda una vez encontrado el elemento, sin necesidad de recorrer completamente la estructura.

## 1.6 Métricas de Diseño de Software

El IEEE define como métrica: “una medida cuantitativa del grado en que un sistema, componente o proceso posee un atributo dado”. Muchos investigadores han intentado desarrollar una sola métrica que proporcione una medida completa de la complejidad del software. Aunque se han propuesto docenas de métricas o medidas, cada una de éstas tiene un punto de vista diferente; y por otro lado, aunque bien se sabe que existe la necesidad de medir y controlar la complejidad del software, es difícil de obtener un solo valor de estas métricas.

Es cada vez más frecuente el uso de ellas en los procesos de desarrollo de software, debido a que las mediciones son esenciales si se quiere obtener un producto de alta calidad. Existen diferentes tipos de

métricas aplicadas a los softwares, específicamente se trataran algunas de las aplicadas para evaluar el Modelo de Diseño.

Las métricas para softwares, como otras métricas, no son perfectas; muchos expertos argumentan que se necesita más experimentación hasta que se puedan emplear bien las métricas de diseño. Sin embargo el diseño sin medición es una alternativa inaceptable.

Junto con la aplicación de la informática a las diferentes ramas, ha ido aumentado el tamaño de los softwares y junto con ellos su complejidad. Con el objetivo de reducirla se usan técnicas de modularización y diseño estructurado, lo que brinda varias ventajas:

- *Comprensibilidad:* Programadores y usuarios pueden comprender fácilmente la lógica del programa.
- *Manejabilidad:* Los líderes pueden asignar fácilmente personal y recursos a los distintos módulos representados por tareas.
- *Eficiencia:* El esfuerzo de implementación puede reducirse.
- *Reducción de errores:* Los planes de prueba se simplifican notablemente.
- *Reducción del esfuerzo de mantenimiento:* La división en módulos favorece que las distintas funciones las lleven a cabo módulos diferenciados.

Estos beneficios aunque son discutidos y se le agrega toda clase de inconvenientes, en general se admite que el paso a la modularidad es un salto hacia adelante. Lo difícil está en como seleccionar los módulos que van a conformar un sistema. Diferentes autores plantean varias técnicas, pero se considera que la más idónea es que cada módulo responda a una funcionalidad por si solo.

Uno de los estudios más completos con respecto a la valoración de cómo conformar los módulos es el llevado por Troy y Zweben. Del cual obtuvieron una serie de principios que sirven para realizar una modularización adecuada. Ellos son:

- *Acoplamiento:* Se mide como el número de interconexiones entre módulos. El acoplamiento crece con el número de llamadas, o con la cantidad de datos compartidos. Se supone que un diseño con un acoplamiento alto puede contener más errores. Se cuantifica como el número de conexiones por nodo del gráfico de diseño.

- *Cohesión:* Valora las relaciones entre los elementos de un módulo. En un diseño cohesivo, las funciones están ubicadas en un solo módulo. Los diseños con una cohesión baja contendrán más errores. Las medidas que valoren la información compartida entre módulos cuantificarán la cohesión.
- *Complejidad:* Un diseño debe ser lo más simple posible. La complejidad crece con el número de construcciones de control, y el número de módulos de un programa. Un diseño complejo contendrá más errores. La complejidad se evidencia en el número de elementos del gráfico de diseño.
- *Modularidad:* El grado de modularidad afecta a la calidad del diseño. Es preferible un exceso a un defecto de modularidad, pues en este último caso contendrá más errores. La cuantificación de la modularidad se obtiene midiendo la cantidad de elementos del gráfico.
- *Tamaño:* Un diseño con grandes módulos, o gran profundidad en su gráfico contendrá más errores. De hecho, complejidad y tamaño están muy relacionados y las consecuencias de un exceso de cualquiera de los dos principios tienen los mismos resultados.

Los objetivos que persiguen las métricas que se muestran a continuación son: Comprender la calidad del producto, estimar la efectividad del proceso y mejorar la calidad del trabajo realizado en el nivel del proyecto. Aunque ninguna es perfecta, pueden proporcionarle al diseñador una mejor visión interna y así el diseño evolucionará a un mejor nivel de calidad.

### 1.6.1 Métricas de diseño arquitectónico

Son consideradas métricas de alto nivel. Miden la eficiencia de la arquitectura y los módulos que conforman el sistema. No se necesita conocer el funcionamiento interno de los módulos para evaluarlos, por ello se consideran como métricas de caja negra.
Card y Glass definen tres medidas de la complejidad del diseño del software: complejidad estructural, complejidad de datos y complejidad del sistema.[6]

- La complejidad estructural, definida como: $S(i) = f^2_{out}(i)$, donde i es el módulo y $f^2_{out}(i)$ es la expansión de i, lo que quiere decir que $f_{out}$ indica el número de módulos que son invocados directamente por el módulo i.

- La complejidad de datos proporciona una indicación de la complejidad en la interfaz interna de un módulo, se define como: $D(i) = v(i)/\left[f_{out}(i)+1\right]$, donde $v(i)$ es el número de variables de entrada del módulo i.
- La complejidad del sistema se define como la suma de las complejidades estructural y de datos, se define como: $C(i) = S(i) + D(i)$.

Los valores de complejidad que se obtienen al aplicar esta métrica tienen gran significado para la arquitectura. Si aumentan los valores, entonces la complejidad arquitectónica del sistema también aumenta, provocando que aumente el esfuerzo para la integración y las pruebas.

### 1.6.2 Métricas de diseño a nivel de componentes

Se concentran en las características internas de los componentes que forman el software, incluyen medidas de la cohesión, el acoplamiento y complejidad de los módulos; las que sirven para que los desarrolladores juzguen la calidad del diseño a nivel de componentes.

Requieren del conocimiento interno de los módulos para llevarse a cabo, por lo que se consideran de caja blanca. Se pueden retrasar hasta que el sistema esté implementado. Evalúan el Bajo Acoplamiento y la Alta Cohesión entre los módulos, permitiendo medir si los Patrones de Asignación de Responsabilidades (GRASP) fueron aplicados correctamente.

#### 1.6.2.1 Métricas de Cohesión

Dos especialistas en el tema, Biemen y Ott han desarrollado métricas para cohesiones funcionales fuertes, cohesiones funcionales débiles y pegajosidad, las definen con cinco conceptos fundamentales:

- Porción de datos: Es una marcha atrás a través de un módulo que busca valores de datos que afectan a la localización del módulo en el que empezó la marcha atrás. Se pueden definir tanto porciones de programas como porciones de datos.
- *Muestras de datos:* Las variables definidas para un módulo pueden definirse como muestras de datos para el módulo.
- *Señales de unión:* El conjunto de muestras de datos que se encuentran en una o más porciones de datos.

- *Señales de superunión:* Las muestras de datos comunes en todas las porciones de datos de un módulo.
- *Pegajosidad:* La pegajosidad relativa de una muestra de unión es directamente proporcional al número de porciones de datos que liga.

La cohesión funcional fuerte y la pegajosidad se obtienen cuando las métricas toman valor 1, y cuando toman valor 0, la cohesión funcional es débil. La cohesión funcional fuerte se define como: $CFF(i) = SU(SA(i))/muestra(i)$, donde $SU(SA(i))$ es la muestra de superunión, quienes son las señales de datos que se encuentran en todas las porciones de datos de un módulo i.

1.6.2.2 Métricas de Acoplamiento

El acoplamiento de un módulo proporciona una indicación de la conectividad del mismo con otros módulos, datos globales y el entorno exterior. Una métrica propuesta por Dhama para el acoplamiento de los módulos combina el acoplamiento de flujo de datos y de control, acoplamiento global y acoplamiento de entorno.

A continuación se muestran las medidas que se necesitan para aplicar esta métrica:

- *Acoplamiento de flujo de datos y de control:* $d_i$ (número de parámetros de datos de entrada), $c_i$ (número de parámetros de control de entrada), $d_o$ (número de parámetros de datos de salida) y $c_o$ (número de parámetros de control de salida).
- *Acoplamiento global:* $g_d$ (número de variables globales usadas como datos) y $g_c$ (número de variables globales usadas como control).
- *Acoplamiento de entorno:* $w$ (número de módulos llamados (expansión)) y $r$ (número de módulos que llaman al módulo en cuestión (concentración)).

Para esta métrica se define como indicador de acoplamiento del módulo de la siguiente manera: $m_c = k/M$, donde $k$ es una constante que toma valor 1 y

$M = d_i + a * c_i + d_o + b * c_o + g_d + c * g_c + w + r$, donde $a = b = c = 2$. Cuanto mayor es el valor de $m_c$, menor es el acoplamiento del módulo.

### 1.6.3 Métricas para Sistemas Orientados a Objetos

La POO ha alcanzado gran auge en las últimas décadas, esta permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar. Junto con la explosión de la POO también surgieron varias métricas con el fin de solucionar el problema de las mediciones a los sistemas OO.

La clase es la unidad principal de todo sistema OO. Las medidas y métricas para una clase individual, la jerarquía de clases, y las colaboraciones entre ellas resultarán sumamente valiosas para un ingeniero de software que tenga que estimar la calidad de un diseño. La clase encapsula a las operaciones (procesamiento) y a los atributos (datos). Suele ser el predecesor de las subclases que heredan sus atributos y operaciones. Suelen colaborar con otras clases.

En muchos de los conjuntos de métricas OO publicados se utilizan propiedades como cohesión, acoplamiento, encapsulamiento, complejidad y herencia que, sin ser todas específicas del diseño OO, son lo suficientemente concretas y además se pueden relacionar matemáticamente con los atributos generales de diseño. Se han propuesto distintas medidas para dichas propiedades e incluso las relaciones con los atributos generales de calidad de diseño. En otra se utilizan 5 medidas aplicadas a clases para medir el grado de satisfacción de dichas propiedades, pero otros especialistas realizan las medidas ya sea sobre los atributos de las clases, los métodos y paquetes.

Las propiedades de diseño que se pueden medir claramente son:

- *Tamaño de diseño.*
- *Jerarquía de clases.*
- *Abstracción.*
- *Encapsulamiento.*
- *Acoplamiento.*
- *Cohesión.*
- *Composición.*
- *Herencia.*
- *Polimorfismo.*

- *Complejidad.*

Existen criterios diferentes de varios autores sobre estas métricas, a continuación se tratan algunas de ellas.

#### 1.6.3.1 Métricas propuestas por Lorenz y Kidd

Estos autores las proponen basadas en las clases. Las separan en cuatro categorías: tamaño, herencia, valores internos y valores externos. También proponen otras orientadas al tamaño y la complejidad de las clases.

Dentro de las métricas orientadas al tamaño se centran en contar los atributos y operaciones de cada clase y los valores para el sistema como un todo son:

- Número de Métodos de Instancia Públicos (PIM).
- Número de Métodos de Instancia (NIM).
- Número de Variables de Instancia (NIV).
- Tamaño de clase (TC).
- Tamaño medio de operación (TMO).
- Número de escenarios (NE).
- Número de clases claves (NCC).
- Número de subsistemas (NSUB).

Las que se basan en la herencia para ver en la forma que las operaciones se reutilizan en la jerarquía de clases son:

- Número de operaciones redefinidas para una subclase (NOR).
- Número de operaciones añadidas por una subclase (NOA).
- Índice de especialización (IES).
- Número de Métodos Heredados (NMI).

#### 1.6.3.2 Métricas propuestas por Chidamber y Kemerer

Es uno de los conjuntos de métricas más difundidos y conocidas como las CK, también se les llama MOOSE. Adoptan tres criterios a la hora de definirlas: capacidad de satisfacer propiedades analíticas,

aspecto intuitivo a los profesionales y facilidad para su recogida automática. Definen los siguientes conceptos:

- Métodos ponderados por clase (MPC).
- Profundidad del árbol de herencia (PAH).
- Número de hijos (NDH).
- Acoplamiento entre clases objeto (AEC).
- Respuesta para una clase (RPC).
- Carencia de cohesión en los métodos (CCM).

#### 1.6.3.3 Métricas propuestas por Li y Henry

Se consideran como una extensión del MOOSE con otras métricas de tamaño, acoplamiento y diseño, también proponen un sistema de predicción de reutilización y mantenibilidad. Ellos modificaron y ampliaron las CK, además de añadir algunas nuevas. Tienen un objetivo claro; la medición y mejora de la mantenibilidad del software OO.

Las métricas tomadas de CK son:

- Métodos ponderados por clase (MPC).
- Árbol de profundidad de herencia (PAH).
- Número de descendientes (NDH).
- Respuesta para una clase (RPC).
- Carencia de cohesión en los métodos (CCM).

Definen otras cinco métricas para el acoplamiento y tamaño:

- Acoplamiento por paso de mensaje (APM).
- Acoplamiento por abstracción de datos (AAD).
- Número de métodos locales (NML).
- Tamaño 1.
- Tamaño 2.

Estas métricas permiten evaluar el diseño, así como saber con anterioridad a cual clase, subsistema y módulo se le debe prestar más atención y designar más recursos para su implementación.

## Conclusiones

En este capítulo se hace un análisis de las características de algunos softwares existentes en el mercado a nivel mundial para el procesamiento de patentes. También se analizan temas y conceptos que todo diseñador del sistema debe dominar para realizar un buen diseño. Como resultado de este capítulo se llegó a las siguientes conclusiones:

- Los softwares existentes actualmente en el mercado mundial no resuelven el problema que se presenta en Delfos al realizar el estudio de las patentes.
- Es necesario la construcción de Predictor para mejorar la calidad en el trabajo y los resultados obtenidos por los especialistas de Delfos en los análisis de información procedente de las patentes.
- Todo diseñador de sistema debe dominar los conceptos claves y los principios que se tienen que seguir en el momento de realizar un diseño de software.
- Una buena práctica de diseño es reutilizar aquellas soluciones que ya han sido probadas satisfactoriamente en otras ocasiones y que se ajustan al problema que se esté resolviendo.
- La aplicación de las métricas para evaluar el diseño dan una medida de la calidad que tiene la solución que se realice.

## CAPÍTULO 2: DISEÑO DEL SISTEMA

### Introducción

En este capítulo se abordan las características técnicas de Predictor. Es importante decir que el diseño está hecho para ser implementado en la Plataforma .NET, específicamente el FrameWork 2.0 y como lenguaje de programación el C #. Para el modelado se usa la herramienta case Rational Rose 2003.

Se escogió la Arquitectura en Tres Capas. Los controles de usuarios y los formularios para la interacción con el usuario se sitúan en la Capa de Aplicación. En la Lógica o de Negocio aquellas clases encargadas de implementar las reglas del negocio, así como las restricciones que desea el cliente, estas son las clases controladoras y entidades que permiten la realización de los casos de uso del sistema. En la de Acceso a Datos las clases que permiten el trabajo directo con los datos almacenados en la Base de Datos local con la información descargada.

El sistema se dividió en cuatro partes fundamentales, Módulo de Descarga, Módulo de Procesamiento, Módulo de Configuración y Módulo de Reportes, cada uno con sus funcionalidades bien definidas.

El Módulo Procesamiento realiza la importante tarea de analizar el código *html* que se encuentra en cada uno de los ficheros descargados y seleccionar las patentes. Este proceso se realiza utilizando un algoritmo de búsqueda secuencial con centinela.

Se aplicaron algunos patrones de diseño que se ajustaron a las necesidades del sistema, estos patrones fueron: Solitario (Singleton), Observador (Observer), Fachada (Facade), cada uno resuelve una situación existente y posibilitan una mejor solución.

El modelo de diseño esta compuesto por distintos subsistemas de diseño, según las funcionalidades de las clases que los integran y en la capa que se encuentren. Estos subsistemas responden a los módulos que forman el sistema.

Se realizan los casos de uso que fueron designados para la primera iteración del proyecto. Su realización juega un papel fundamental para la comprensión del funcionamiento del sistema y como debe ser implementado. Se presentan las clases que interactúan en cada uno de los Casos de Uso (CU), así como los diagramas de secuencia en cada uno.

Es importante decir que los artefactos que anteceden al Modelo de Diseño que se plantea en este trabajo se pueden encontrar dos trabajos de diplomas. Los artefactos pertenecientes al Analista de Sistema en [9] y los del Arquitecto en [10].

## 2.1 Arquitectura definida para el sistema

La arquitectura de software establece los fundamentos para que analistas, diseñadores, programadores, etc. trabajen en una línea común que permita alcanzar los objetivos y necesidades del sistema.
En el Análisis y Diseño que plantea RUP, trabajan en conjunto para lograr un correcto modelo de diseño y demás artefactos generados en esa disciplina dos trabajadores: el diseñador del sistema y el arquitecto. Es en este momento que se generan tres de las cinco vistas arquitectónicas del sistema, por lo que no se pueden ver separados el diseño y la arquitectura de un software. Es ella quien le da forma al software para que soporte todos los requisitos.
Se necesita una arquitectura robusta, que guíe el proceso de desarrollo, y que defina de manera abstracta, los componentes que llevan a cabo alguna tarea, sus interfaces y la comunicación ente ellos.
La arquitectura seleccionada fue la de tres capas. Esta se ha vuelto muy común a la hora de construir software de gestión debido a las facilidades que brinda, permite la reutilización y la independencia entre las capas, se pueden realizar cambios en capas sin tener que modificar las otras, facilita la estandarización, la utilización de los recursos y la administración. En este caso se crean tres capas:

- *Aplicación o Interfaz de usuario:* Tiene todos los aspectos del software que tienen que ver con las interfaces y la interacción con los usuarios, es aquí donde están ubicadas todas las ventanas y menús que permiten el intercambio de información con la aplicación. Específicamente en esta capa se encuentran los controles de usuarios que son usados para la realización de los distintos casos de uso, como por ejemplo, IU_Busqueda_Avanzada, entre otras. También forman parte de esta capa dos componentes que no pertenecen al FrameWork que brinda la plataforma de desarrollo. El TreeListView que es utilizado en los controles de búsqueda y el de reportes. Y el menú estilo Outlook para mejorar la presencia gráfica y acceder a las distintas funcionalidades del sistema mediante él.
- *Lógica de Negocio:* Esta capa reúne todos los aspectos del software que automatizan o apoyan los procesos de negocio que llevan a cabo los usuarios. Estos aspectos típicamente incluyen las tareas que forman parte de los procesos, las reglas y restricciones que aplican. Es aquí donde están ubicados los subsistemas y clases encargadas de realizar la mayoría de las funcionalidades que pide el cliente, las clases controladoras y las entidades.
- *Acceso de Datos:* Contiene las clases que interactúan con la base de datos y que permiten realizar las distintas operaciones sobre ella, como son: procedimientos almacenados, consultas.

La comunicación entre las distintas capas en este tipo de arquitectura se puede realizar de distintas formas, la usada en Predictor fue la top-down. Plantea que los elementos de una capa i+1 pueden enviar solicitudes de servicio a elementos de la capa inferior i. Típicamente se produce una cascada de solicitudes, es decir para satisfacer una solicitud a una capa i+2, ésta requiere enviar varias solicitudes a la capa i+1; cada una de estas solicitudes a la capa i+1 genera a su vez un conjunto de solicitudes a la capa i y así sucesivamente. La relación entre las capas es unidireccional, solo las capas superiores tienen acceso a la inmediata inferior. Los elementos existentes en cada capa si tienen relación entre ellos.

Otro aspecto importante y poderoso que brinda la arquitectura e influye positivamente en la extensibilidad del sistema, es la posibilidad que brinda de integrar nuevos componentes sin la necesidad de volver a instalarse en las estaciones de trabajo donde se halla instalado anteriormente, facilitando así las actualizaciones. Lo antes explicado se logra mediante un fichero de configuración XML del cual se carga en tiempo de ejecución los controles de usuarios que el cliente necesite.

El proceso de actualización se realizaría de la siguiente manera: Se procede a construir el componente nuevo para la funcionalidad que se le tenga que agregar al sistema. Una vez terminado se le agrega una nueva opción al menú Outlook y se modifica el XML de configuración. En cualquier estación de trabajo que esté instalado el sistema y se le quiera adicionar la nueva funcionalidad, solo se tiene que agregar la *dll* con el nuevo componente, cambiar el XML de configuración y el menú Outlook, sin necesidad de volver a reinstalar la aplicación. Para mayor información se pueden consultar [10] y [11].

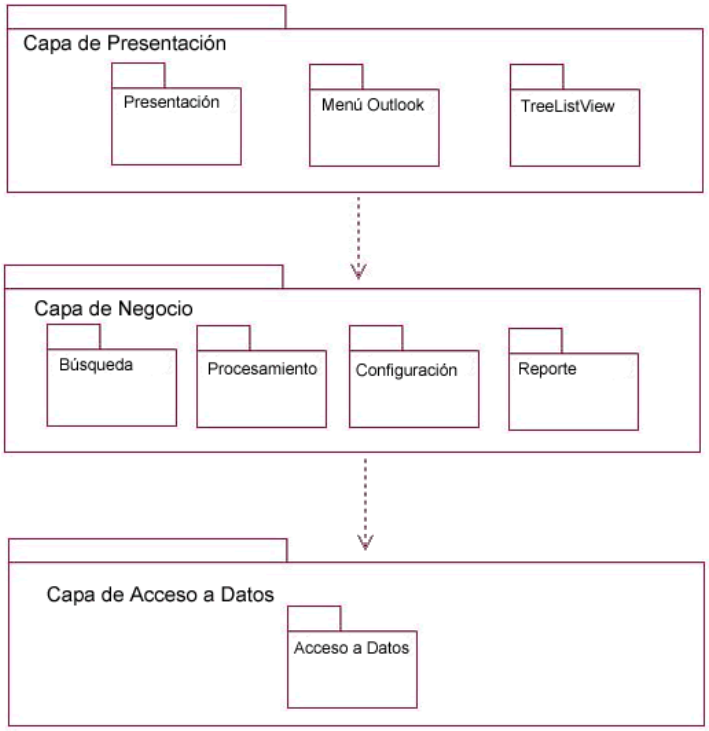


Figura 1: Arquitectura de Predictor

## 2.2 Descripción de los módulos principales

Una de las principales características que debe tener un buen diseño es la modularidad, al dividir un sistema en partes más pequeñas es necesario prestarle mucha atención para no incurrir en gastos innecesarios.

Predictor está compuesto por cuatro módulos principales, los cuáles interactúan entre ellos para llegar al propósito final, descargar la información necesaria y procesarla correctamente. Los módulos son: Configuración, Procesamiento, Descarga y Reportes.

### 2.2.1 Módulo de Configuración

Su función es configurar aspectos imprescindibles para realizar las solicitudes de búsqueda y descarga de información. Es el encargado de la captura de los datos necesarios para la conexión a Internet, como son: tipo de conexión (con o sin Proxy), datos del Proxy (número IP, puerto, usuario y contraseña). Las

interacción con los usuarios se realizaría mediante una ventana que le da las opciones de configurar la conexión; dicha configuración es usada posteriormente para la descarga. Véase Anexo 9

### 2.2.2 Módulo de Descarga

Es el módulo principal del sistema, su función es conectarse a las bases de datos seleccionadas, teniendo en cuenta los parámetros de búsquedas introducidos por el cliente y descargar la información existente. Para eso construye las *urls* y navega por las páginas automáticamente. Para realizar la descarga es necesaria la previa configuración de la conexión a Internet, así como donde se desea guardar la información, estas funcionalidades son responsabilidad del Módulo de Configuración y de él mismo respectivamente. Se usan también las funcionalidades del Módulo Procesamiento, una vez descargado el código *html* de las páginas, se les realiza el análisis, buscando los parámetros de cada patente para confeccionar los dos ficheros finales de la búsqueda, el Procite.txt y el Excel.txt.

### 2.2.3 Módulo de Procesamiento

Este se encarga de trabajar directamente con el código *html* de las páginas Web descargadas, buscando en ellas los parámetros de las patentes, de esta forma selecciona la información y confecciona dos archivos con los formatos exigidos por los clientes, el Procite.txt y el Excel.txt. Estos archivos son usados por el Módulo de Reporte y también para ser cargados con otras aplicaciones con que trabaja Delfos para realizar los análisis. Las funcionalidades que brinda este módulo se pueden ejecutar de dos formas: una, cuando se realizan las descargas el sistema procesa transparente al usuario, y lo que se tiene como resultado son los archivos finales generados con las patentes encontradas; pero también se puede procesar información almacenada de anteriores descargas, en este caso inicializada por el usuario.

### 2.2.4 Módulo de Reportes

Su función es mostrar los datos de las descargas realizadas con anterioridad por los usuarios, selecciona la descarga de la cuál desea ver los resultados y se muestran los datos de cada una de las patentes encontradas, una vez hecho el análisis de la información, trabaja directamente con los resultados provenientes del Módulo de Descarga. Véase Anexo 14

Como se puede observar, estos módulos están relacionados uno con otros, debido a que las funcionalidades de algunos están incluidas dentro de otro; así como la dependencia existente entre ellos, para que uno realice sus funciones, es necesaria la realización correcta de las tareas de otro módulo.
Fue dividido de esa manera producto a que son tareas específicas y concretas las que tiene cada módulo y son realizadas por conjuntos de componentes distintos. Para eso se crearon distintos subsistemas; aquellos componentes o clases usadas para la realización de distintas funcionalidades se implementaron en un subsistema aparte, al cuál acceden todos los demás. Esta división permite la implementación al mismo tiempo de cada uno de ellos después de haber sido implementada la arquitectura base y los componentes comunes para las distintas funcionalidades; para luego ser integrados, haciendo de esta manera más rápido el trabajo.

## 2.3 Procesos de Búsquedas y Procesamiento de ficheros

Los procesos más complejos que tiene el sistema son el de búsqueda y descarga de las páginas Web que contienen las patentes y el de procesamiento de los ficheros descargados con el código *html* de cada una de las páginas.
En este epígrafe se explica en que consiste cada uno de ellos y los aspectos que hay que tener en cuenta para la implementación.

### 2.3.1 Búsqueda y Descarga

Independientemente del tipo de búsqueda que se desee realizar (Básica o Avanzada), el procedimiento a seguir es el mismo. No se necesita implementar ningún algoritmo complejo para buscar la información pedida por el cliente. Consiste en interactuar con la página de la base de datos seleccionada por el usuario que trabaje con el sistema.
Las páginas de las oficinas de patentes funcionan como sistemas externos. El proceso se inicia visitando la página para el tipo de búsqueda que se escoja con los criterios introducidos, véase Anexo 17, se obtiene como resultado otra página con todos los vínculos, véase Anexo 18 . Se descarga esa página y se seleccionan cada uno de los vínculos para descargar las páginas a las cuáles hacen referencia, que serían las que tienen los datos de las patentes relacionadas con el tema a buscar. El código *html* de ellas es guardado en un directorio para posteriormente ser procesados y extraer de cada uno la patente que contiene.

Para solicitar y obtener las páginas Web es necesario hacerlo mediante el protocolo *http* y se hace uso de clases que brinda el framework de .NET. La clase WebRequest para la solicitud del recurso existente en Internet, y la WebResponse para obtener los datos devueltos y el WebProxy para los datos de la autenticación en el proxy. La clase WebResponse permite que se obtenga una secuencia de caracteres que representa la respuesta a la solicitud realizada, y es tratada de la misma manera que leer un archivo de texto local. Esta secuencia es la guardada en los ficheros que se almacenen en el directorio seleccionado.

### 2.3.2 Procesamiento de ficheros

Consiste en analizar cada uno de los ficheros descargados y extraer las patentes. Se analizan por separados cada uno, teniendo en cuenta que solamente poseen una patente.

Se inicia cargando en una lista cada una de las líneas que conforman el fichero, entiéndase que en estas líneas estaría el código *html* completo de la página. Con todas las líneas cargadas en memoria, se hace necesario aplicar un algoritmo de búsqueda para seleccionar los parámetros de interés dentro de la lista.

Existen básicamente dos algoritmos que se pueden usar, la Búsqueda Binaria o la Búsqueda Secuencial. En este caso se seleccionó una modificación de la secuencial, la Búsqueda Secuencial con Centinela, debido a que la lista no está ordenada y este algoritmo para automáticamente cuando se encuentre el objeto deseado.

Aplicando ese algoritmo se procede a extraer de la lista los datos de interés y almacenarlos en otra. Esta selección depende de la página que se descargó el archivo. Fue necesario estudiar minuciosamente como muestran la información sobre las patentes las distintas páginas con las que se trabaja, para así crear algoritmos específicos de selección de los parámetros.

Con la lista de los parámetros se procede a extraer el texto de cada una de las líneas, se eliminan los *tags* del *html*. Aquí se trabaja con clase Regex, perteneciente al framework de .NET. Para esto cada línea es tratada como una expresión regular y se le extraen todos los caracteres que no interesen, estos pueden ser los distintos *tags* del *html*. Luego se conforman los ficheros finales (Procite.txt y Excel.txt) con los formatos exigidos por los clientes.

## 2.4 Justificación de los Patrones de Diseño utilizados

En el Modelo de Diseño de Predictor se proponen usar patrones pertenecientes a los Gof, 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 tres patrones que encajan para darle solución a la problemática de diseñar el sistema haciendo uso de soluciones probadas satisfactoriamente en casos anteriores.

Los patrones del Gof usados en el diseño del sistema son:

### 2.4.1 Patrón Solitario (Singleton)

Con este patrón se garantiza una única instancia de aquellas clases que se desee tener una sola en toda la aplicación, proporciona un punto de acceso global a dichas clases. Reduce el espacio de nombres, es una mejora sobre las variables 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 aplica el Patrón Fachada, quién se explica a continuación. Las clases que implementan este patrón son: MotorSolicitudes, BusquedaBasica, BusquedaAvanzada, FachadaParser, GestorConfiguracion, ConfiguracionInternet, MotorProcesamiento, GestorDatosProyecto y GestorReporte.

### 2.4.2 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.

### 2.4.3 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 esta 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. Véase Figura 2.

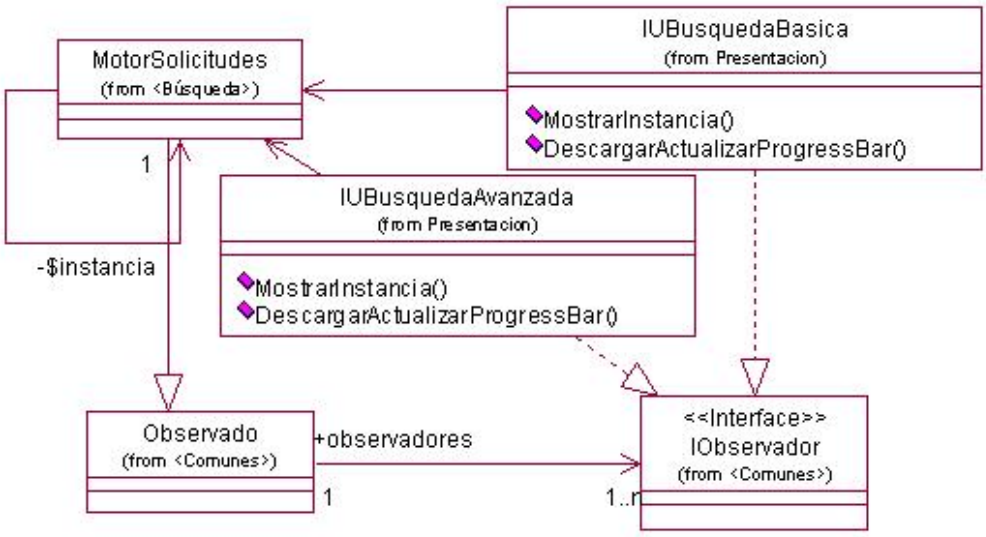


Figura 2: Diagrama de Clases Patrón Observador

## 2.5 Modelo de Diseño

En el diseño se modela el sistema y al mismo tiempo se le da forma a la arquitectura, soportando así todos los requisitos, restricciones y reglas. Un artefacto importante que se puede tener como punto de partida para hacer el modelo de diseño es el modelo de análisis, que proporciona detalladamente aspectos referentes a los requerimientos. Aunque RUP plantea que no es obligatoria la confección del modelo de análisis, este resulta de gran ayuda y sirve como base para crear el diseño orientado a diferentes plataformas y lenguajes de programación. [8]

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en cómo los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tiene impacto en el sistema a considerar. Además, el modelo de diseño sirve de abstracción a la implementación del sistema y es, de ese modo, utilizada como una entrada fundamental a las actividades de implementación. [8]

En este caso el modelo de diseño está compuesto por un sistema de diseño que a su vez lo conforman varios subsistemas.

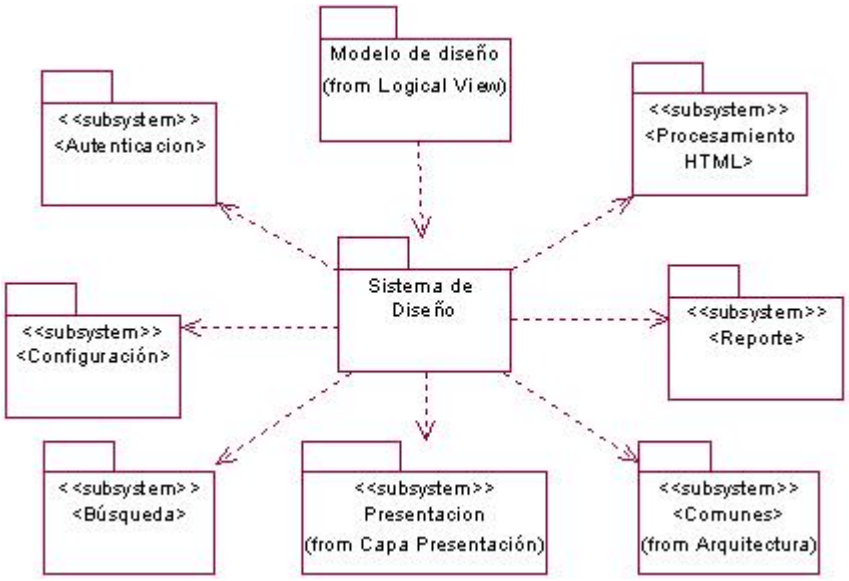


Figura 3: Modelo de Diseño del Sistema

### 2.5.1 Subsistemas de Diseño

El Modelo de Diseño está compuesto por subsistemas de diseño que interactúan entre si para realizar los casos de uso.

A continuación se explica la funcionalidad de los principales subsistemas, se muestra el diagrama de las clases que lo conforman y se describen las clases más importantes.

#### 2.5.1.1 Subsistema Comunes

Este es un subsistema de servicio, en él están todas las clases e interfaces que son usadas desde otros subsistemas para diferentes funcionalidades. Por la importancia para la comprensión de los demás subsistemas se decidió que fuese el primero en explicarse.

No todas las clases están relacionadas entre sí, la función de este subsistema es brindar un apoyo y facilitar el trabajo de los demás. Las clases que son usadas por más de un subsistema que tiene funcionalidades específicas, como el caso de Configuración, ProcesamientoHTML o Búsqueda están modeladas en Comunes, permitiendo así que los subsistemas antes mencionados estén menos acoplados, y dependan lo menos posible uno de los otros.

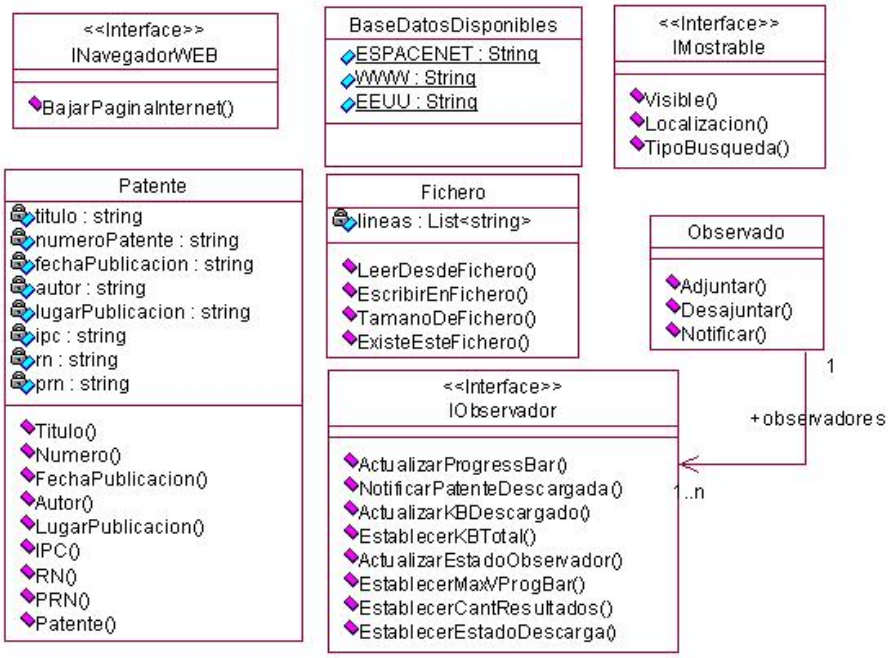


Figura 4: Diagrama de clases del Subsistema Comunes

A continuación se explican cada una de las interfaces y clases existentes.

- *Fichero:* Es la que permite hacer el trabajo con los ficheros. Es muy importante y se usa en la descarga, el procesamiento y los reportes. Tiene funciones que son las encargadas de leer, escribir, verificar si existe y devolver el tamaño del fichero seleccionado. Estas son llamadas desde los módulos de la aplicación en el momento de hacer sus funciones que necesitan procesar los ficheros descargados.
- *Patente:* Representa el motivo principal por el cuál se lleva a cabo el proyecto, tiene como atributos los parámetros de interés para hacer posteriormente 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. Es una clase crucial para todos los subsistemas.
- *Observado:* El propósito de esta clase es usar el patrón Observer. Hace función del observado abstracto, del cual hereda el observado concreto MotorSolicitudes, perteneciente al Subsistema Búsqueda. Tiene como atributos una lista de IObservador.

- *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 un 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. Se usa en el momento de seleccionar la fuente de búsqueda, para entonces conformar la dirección electrónica. Es usado en el momento de descargar las páginas. Es en esta clase donde se deben agregar 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. Estos métodos son sobre escritos de forma diferente según el tipo de búsqueda que se realice. Es implementada por todos los controles de usuarios que son cargados en algún momento en la aplicación.
- *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.

#### 2.5.1.2 Subsistema Presentación

Este subsistema se encuentra en la Capa de Presentación o Aplicación en la arquitectura del sistema, solamente están en él aquellos componentes que permiten interactuar con el cliente, es decir ventanas y formularios para capturar o mostrar datos.

El diseño propuesto para este subsistema, al igual que todos los demás, está estrechamente vinculado a la plataforma y lenguaje de programación en el cuál se va a implementar. Se usan los controles de usuarios que brinda el C # para crear las diferentes interfaces que se mostrarán. Esto permite que con una sola forma se puedan realizar todas las funcionalidades que el sistema tiene que cumplir, los controles son llamados con la ocurrencia de un evento determinado para mostrarse en una parte específica de la forma principal, evitando así la creación de varias formas.

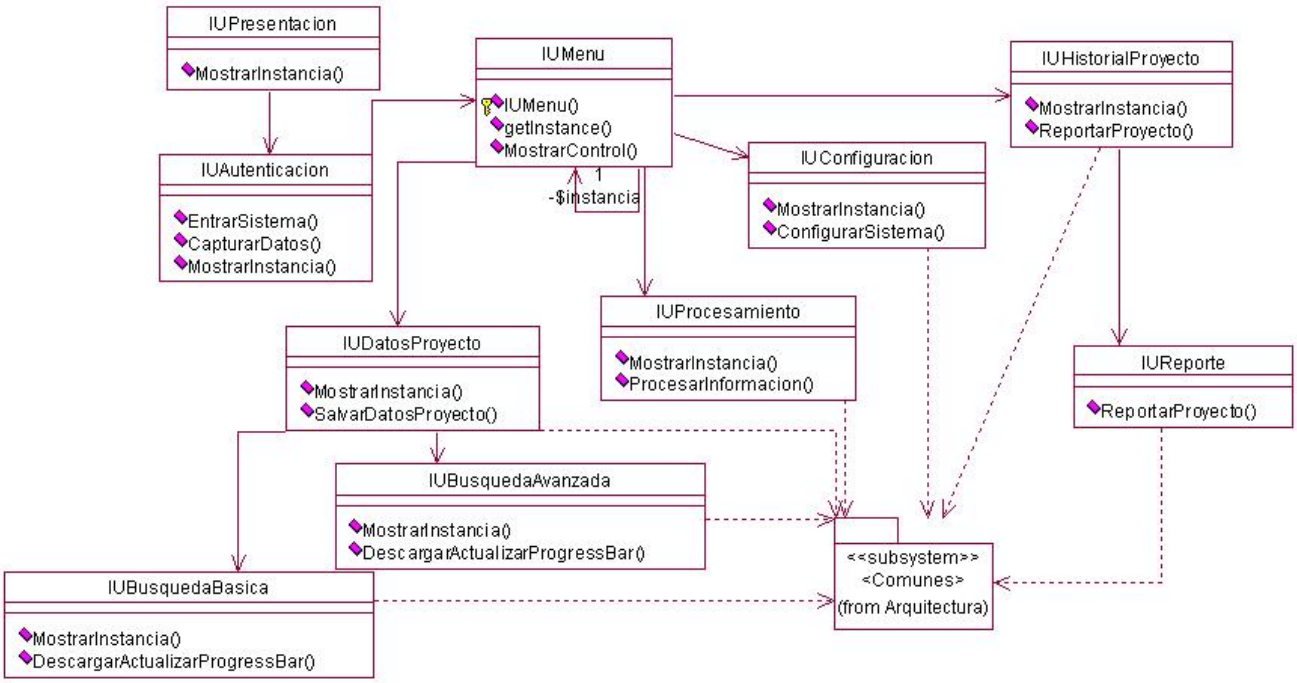


Figura 5: Diagrama de clases del Subsistema Presentación

En el diagrama se representa la relación entre los controles de usuarios, IUConfiguracion, IUBusquedaAvanzada, IUBusquedaBasica, IUDatosProyecto, IUProcesamiento, IUReporte y IUHistorialProyecto con el paquete Comunes porque todos implementan la interface IMostrable. Los controles de las búsquedas también implementan IObservador e INavegadorWEB.

Cada uno de los controles de usuarios tiene sus funciones, ahora se explicará en que consisten y que como están formados.

- *IUPresentación:* Esta es una forma que su función es mejorar la interfaz gráfica y hacer más amena la interacción con el usuario. Es un 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 encargada de controlar el acceso a la aplicación.
- *IUAutenticación:* Para trabajar con el sistema se necesita ser usuario del mismo. La función de esta forma es el control el acceso de los usuarios, tiene dos TextBox que permiten introducir nombre

y contraseña, así como un Botón que manda a comprobar la autenticidad de los datos introducidos, responsabilidad de otras clases que se encuentran en otras capas del sistema. Véase Anexo 10

- *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. Es mediante este que se pueden realizar todos los casos de uso del sistema.
- *IUConfiguración:* Este es un control de usuario que se invoca desde un evento de IUMenu, tiene como función brindar la posibilidad de configurar el tipo de conexión con 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. Véase Anexo 9
- *IUDatosProyecto:* Brinda la posibilidad de darle nombre a la descarga que se desea realizar, así como otras cosas de interés para el usuario, como puede ser, comentario y descripción. Algo muy importante es que aquí es donde se selecciona el directorio donde se va a guardar la descarga mediante un OpenDialog. Después de realizar lo antes mencionado se guardarían los cambios y automáticamente se pasaría a la interfaz de descarga, la cuál puede ser Básica o Avanzada, estas se explican a continuación. Véase Anexo 11
- *IUBusquedaBasica:* Su función es capturar los datos para realizar la búsqueda básica. Se seleccionan las Bases de Datos de las que se desea ver la información, dicha selección puede ser múltiple o sencilla, y se introduce el término a buscar, dando la opción de que esté en el título o en el resumen de la patente. Brinda la posibilidad al usuario de saber cuantas patentes encontró y el porciento descargado hasta el momento mediante un ProgressBar, informando finalmente cuando se termine de bajar todo la información para luego pasar a otras interfaces como la de Reporte. Véase Anexo 12
- *IUBusquedaAvanzada:* Al igual que la IUBusquedaBasica, su función es capturar los datos para la búsqueda, pero en este caso se diferencia en que se capturan más parámetros a buscar, como son: fecha de publicación, solicitante de la patente, clasificación internacional de la patente y el término de búsqueda, que puede ser en el título o resumen, como la búsqueda básica. Las bases de datos se configuran igual que en la básica, y brinda la misma posibilidad con un ProgressBar para mayor conocimiento por parte del usuario de la descarga que esté realizando. Véase Anexo 13

- *IUHistorialProyectos:* La función que tiene 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. Se realiza mediante un OpenDialog. En ambos casos después de estar seleccionada la descarga, se pasa a IUReportes.
- *IUReportes:* Es el que según la descarga seleccionada, muestra todas las patentes con sus parámetros. Lo que le posibilita al usuario una mayor visibilidad y conocimiento antes de pasar a realizar los análisis de la información con otros softwares. Véase Anexo 14
- *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.

#### 2.5.1.3 Subsistema ProcesamientoHTML

Se encuentra en la capa de negocio, su función es realizar el procesamiento de los ficheros descargados, quienes contienen código *html*. Las principales funciones que realiza son las de seleccionar las *urls*, eliminar las repetidas para evitar la duplicidad de la información, construir listado de patentes y crear las muestras finales, que serían los dos ficheros que se obtienen como resultado de este subsistema, el Procite.txt y el Excel.txt. Estos sirven para hacer los reportes y ser cargados con otras aplicaciones que facilitan los análisis de los especialistas de Delfos.

Está formado por dos clases, una interfaces y otro subsistema llamado Parser, quien tiene como función hacer la minería de texto a los ficheros, de manera tal que solo queden aquellos parámetros de las patentes que son importantes para conformar la información que va a ser guardada en los ficheros finales. Tiene una clase fachada (FachadaParser) que abstrae a las clases del subsistema ProcesamientoHTML de su estructura interna y su funcionamiento, mediante la cual se accede a las funciones que permiten llevar a cabo del análisis de la información descargada de distintas bases de datos.

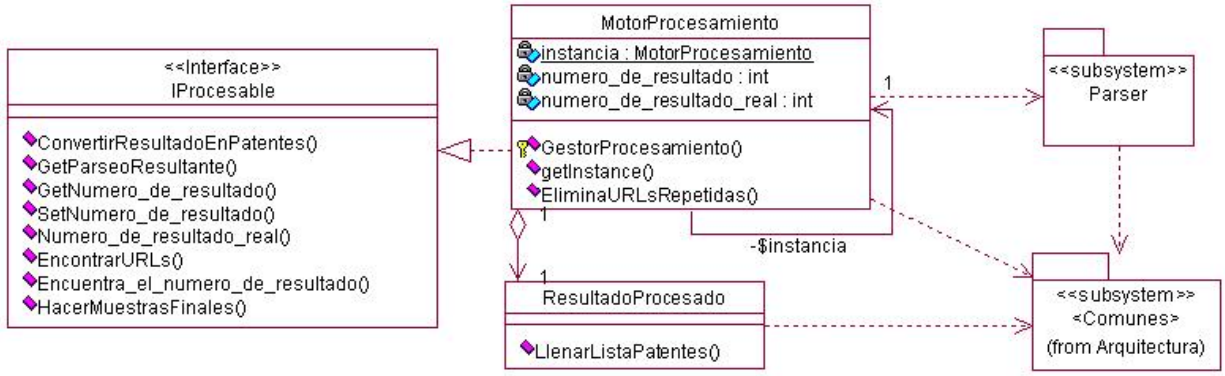


Figura 6: Diagrama de clases del Subsistema ProcesamientoHTML

Las relaciones entre las clases GestorProcesamiento y ResultadoProcesado con Parser y Comunes se deben a la necesidad de clases que existen en ellos para realizar sus funcionalidades. En el caso de la primera, llama a los métodos de la clase fachada de Parser para poder realizar el análisis de los ficheros. De Comunes usa la clase Patente para crear el listado de patentes; la BaseDatosDisponibles, para seleccionar la base de datos para realizar la descarga y la 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 las clases del subsistema.

- *IProcesable*: Es una interface de la cual va a heredar su comportamiento la clase GestorProcesamiento. Tiene varias funciones que serían sobrecargadas en aquellas clases que la implementen.
- *GestorProcesamiento:* 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. Es esta clase la que usa las funcionalidades del subsistema Parser para realizar el trabajo de procesamiento de los ficheros. 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 solamente dos funciones además del constructor de la clase; la función que permite retornar la lista de patentes y la que crea la lista de patentes según un fichero dado.

2.5.1.4 Subsistema Parser

Pertenece a la capa de negocio, se encuentra específicamente dentro del subsistema ProcesamientoHTML. Su función es trabajar directamente con los ficheros descargados, quienes contienen el código *html* de cada una de las páginas. Entre las principales tareas están: seleccionar las *urls* para realizar las descargas, eliminar los *tags* del código que no hagan falta, seleccionar los parámetros de las patentes en cada uno de los ficheros procesados, confeccionar listado de patentes, crear los ficheros finales de cada descarga. Da el punto final al proceso de descarga de información.
Está compuesto por varias clases, las que se muestran en el diagrama a continuación.

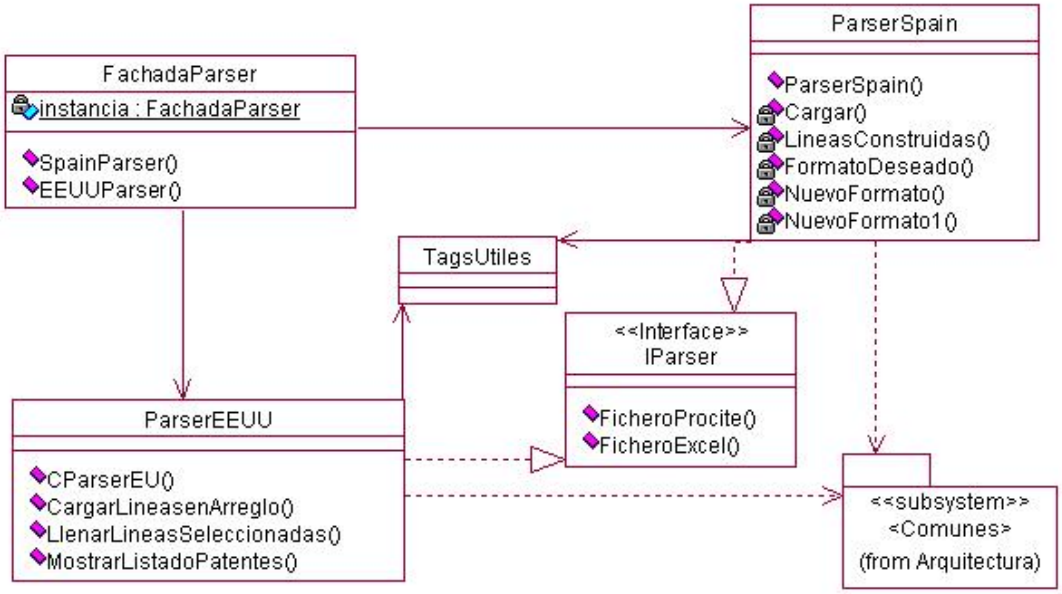


Figura 7: Diagrama de clases del Subsistema Parser

- *FachadaParser:* Es la clase que sirve como fachada del subsistema, mediante ella se brindan las funcionalidades de este a los demás. Tiene dos métodos que en dependencia de la base de datos que se halla descargado se llama a la clase parser correspondiente, esto se debe a que los ficheros descargados de distintas bases de datos no siempre tienen la misma estructura y se necesita procesarlo de manera diferente.

- *ParserEEUU:* Su función es realizar el análisis a los ficheros descargados de la base de datos de EEUU. Implementa la interface IParser, hace uso de las funciones que tiene la clase TagsUtiles para analizar las líneas de los ficheros y usa las clases Patente y Fichero del subsistema Comunes.
- *ParserSpain:* Al igual que ParserEEUU, realiza el procesamiento de los ficheros descargados, pero de las bases de datos World Wide Web y Espacenet. También usa TagsUtiles para limpiar las líneas, la Patente y Fichero, implementa lParser y crea los ficheros Procite.txt y Excel.txt.
- *IParser:* Es una interfaces que tiene los métodos que son comunes para las clases ParserSpain y ParserEEUU, es implementada por ambas, y son sobre escritos los métodos FicheroExcel y FicheroProcite en cada una de las clases concretas.
- *TagsUtiles:* Es muy importante, su trabajo es exclusivamente con las líneas de los ficheros, está clase es llamada desde las clases ParserSpain y ParserEEUU cuando se está realizando la limpieza del código *html*. Tiene métodos para eliminar *tags* y verificar la existencia de los mismos. La cantidad de métodos que contiene se debe a la variedad de *tags* diferentes que existen en *html.*

2.5.1.5 Subsistema Configuración

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

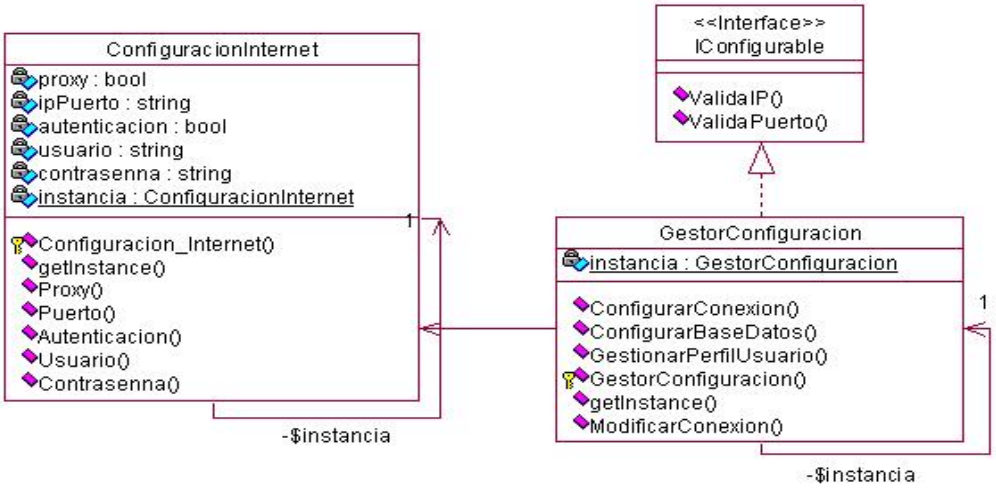


Figura 8: Diagrama de clases del Subsistema Configuración

- *IConfigurable*: Es una interface que tiene dos métodos que permiten validar el IP del Proxy y el Puerto para realizar la conexión. Son redefinidos en la clase GestorConfiguracion que implementa esta interface.
- *GestorConfiguracion:* Tiene una instancia de ella misma que permite que exista solo 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 solo exista una sola conexión a Internet.

#### 2.5.1.6 Subsistema Búsqueda

Es el principal, su función es buscar 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, quienes 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 limpiar 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 el gestor que se instancia, 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 descargada todas las páginas encontradas se pasa al procesamiento de los ficheros, tarea que realiza el Subsistema ProcesamientoHTML.

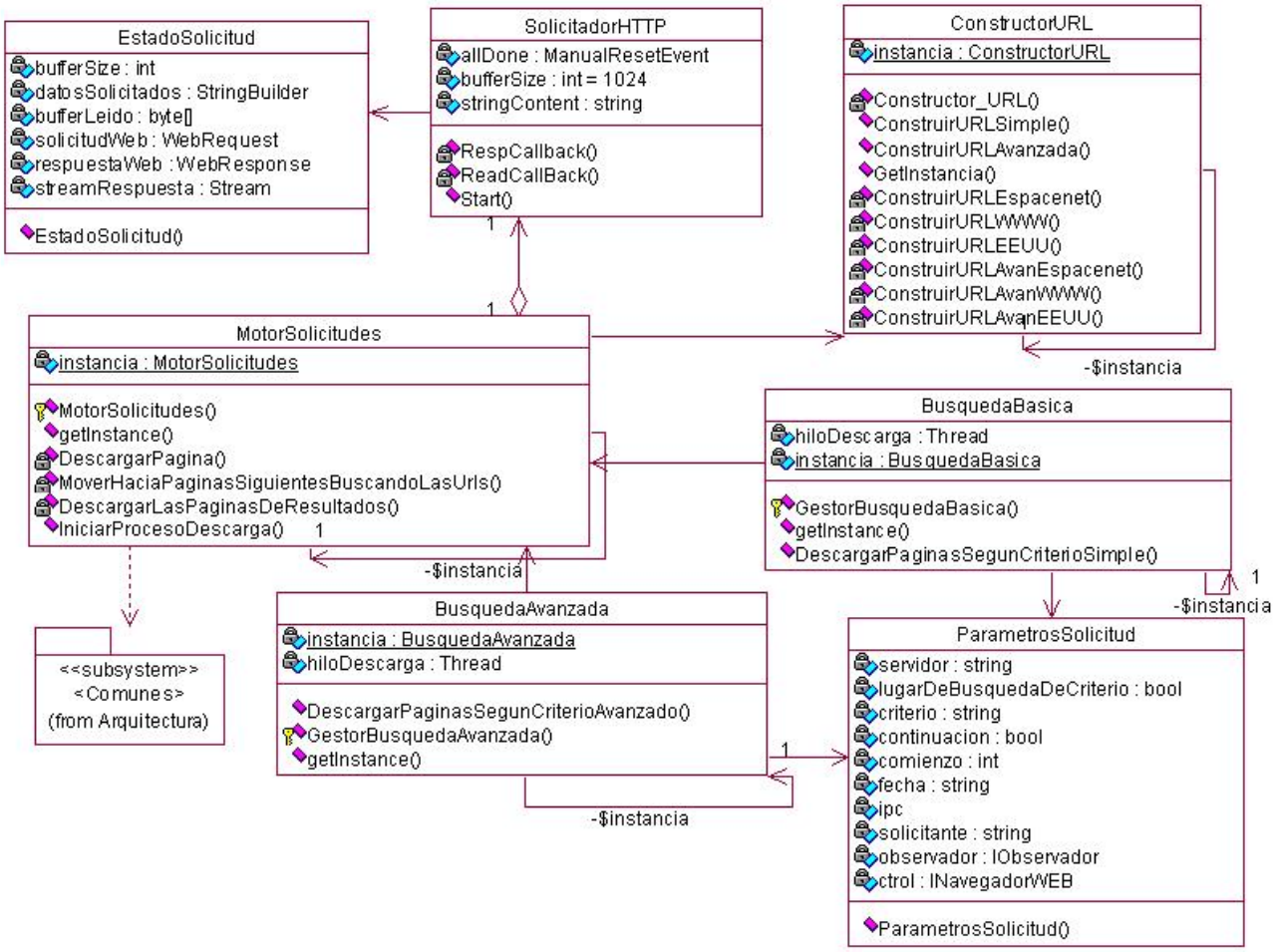


Figura 9: Diagrama de clases del Subsistema Búsqueda

- *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 manera.
- *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 descargas las páginas encontradas. Su relación con Comunes se debe a que utiliza la clase Fichero.
- *SolicitadorHTTP:* Esta clase tiene la función de descargar la página Web, teniendo como parámetros para realizarlo, la *url* y la configuración de Internet. La función encargada de eso es Start(), y se invoca a la clase EstadoSolicitud para hacer uso del WebResponse y el WebRequest.
- *EstadoSolicitud:* Su función es controlar los estados de las descargas.

### 2.5.1.7 Subsistema Reportes

Se localiza en la capa de negocio. Tiene las clases que permiten realizar los reportes y controlar los datos de los proyectos descargados. Es usado en dos funcionalidades principales; en la búsqueda y en los reportes.

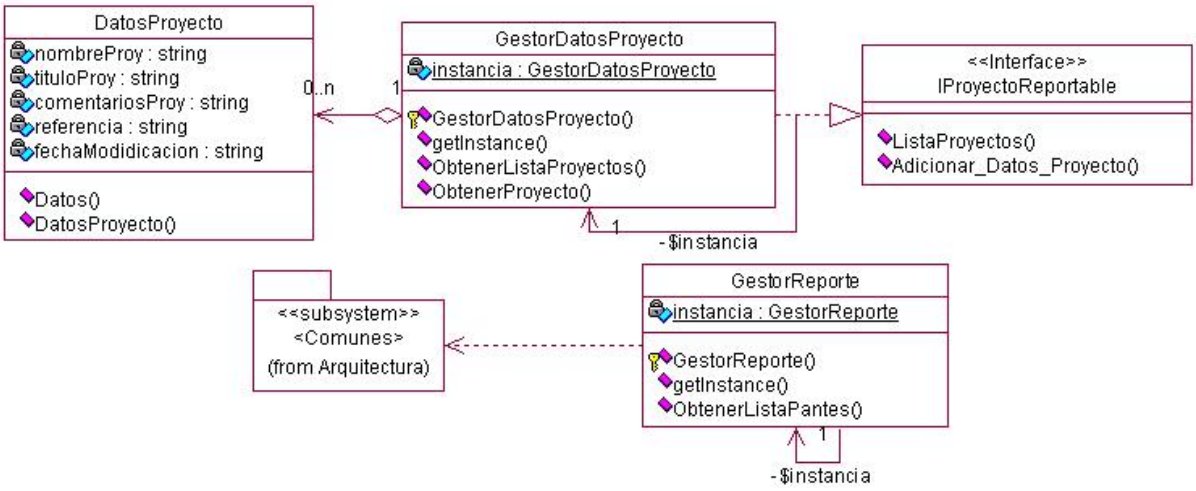


Figura 10: Diagrama de clases del Subsistema Reportes

- *IProyectoReportable:* Es una interface implementada por la clase GestorDatosProyecto que redefine sus funciones.
- *DatosProyecto:* Guarda los datos del proyecto que se va a descargar. Se usa en los casos de uso de búsqueda, para crear los proyectos y en el de reporte para consultarlo y mostrar las patentes pertenecientes al mismo. Tiene como atributos todo lo que se captura por la interfaz de usuario IUDatosProyecto.
- *GestorDatosProyecto:* Su función es gestionar los proyectos que se creen en la aplicación. Implementa el patrón Singleton. Tiene todas las funciones que se necesitan en las búsquedas para guardar las descargas y en los reportes para seleccionar un proyecto determinado.
- *GestorReporte:* Según un proyecto seleccionado busca el directorio donde está guardado y proporciona el listado de patentes. De ahí su relación con el Subsistema Comunes, necesita la clase Patente. Se usa exclusivamente en el caso de uso Reportes.

#### 2.5.1.8 Subsistema Autenticación

Este subsistema se encarga de controlar el acceso a la aplicación, según el usuario y contraseña introducidos por el especialista se comprueba sin son válidos y se le da acceso a trabajar con Predictor.

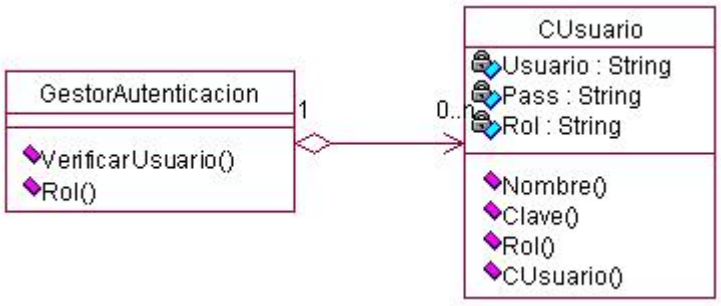


Figura 11: Diagrama de clases del Subsistema Autenticación

- *CUsuario:* Es la clase entidad que tiene los atributos de interés que se desean controlar de los usuarios que trabajan con la aplicación.
- *GestorAutenticacion:* Clase gestora que permite verificar la veracidad de los datos introducidos por los especialistas que trabajan con el sistema. Verifica la existencia según el usuario y la contraseña y devuelve el rol que tiene un determinado usuario.

Se representaron y explicaron cada una de las clases según a los subsistemas que pertenezcan, pero no se pone de manifiesto en su totalidad las relaciones que existen entre ellas para la realización de las funcionalidades que debe tener Predictor. En el epígrafe que viene a continuación, donde se realizan los casos de usos se puede observar mejor las relaciones entre las clases interfaces de usuarios, las controladoras y las entidades.

### 2.5.2 Realización de Casos de Uso

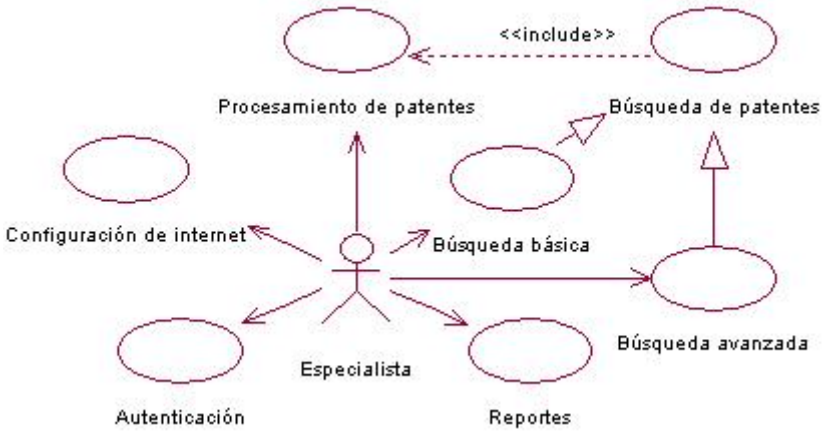


Figura 12: Diagrama de Casos de Uso del Sistema

Para describir las realizaciones de cada caso de uso se sigue el siguiente orden: se explica brevemente en que consiste el caso de uso, se presentan el diagrama de las clases que participan y los diagramas de interacción, específicamente diagramas de secuencia. Y por si queda alguna duda y no se comprende algún aspecto del diagrama de secuencia, también se explica una descripción del flujo de sucesos de diseño para cada caso de uso, así como una breve explicación de los mensajes entre objetos en los diagramas que no se hayan incluido los parámetros.

#### 2.5.2.1 Caso de Uso Autenticación

Este caso de uso es el encargado de controlar el acceso al sistema, para trabajar en él se necesita ser usuario. Si los datos introducidos para autenticarse son correctos, una vez que el usuario se conecta tiene acceso a las funcionalidades según los permisos correspondientes; pero en caso contrario, no se puede pasar de la forma de autenticación, y es imposible trabajar con el sistema. Esto permite el control de las

operaciones que realizan los distintos especialistas, ya sean descargas, procesamiento o simples consultas en la base de datos local.

Las clases que interactúan son las mostradas en la Figura 13, como se observa, existen dos interfaz de usuario, una controladora y una entidad. Las interfaces son formas, que una le da paso a la otra.

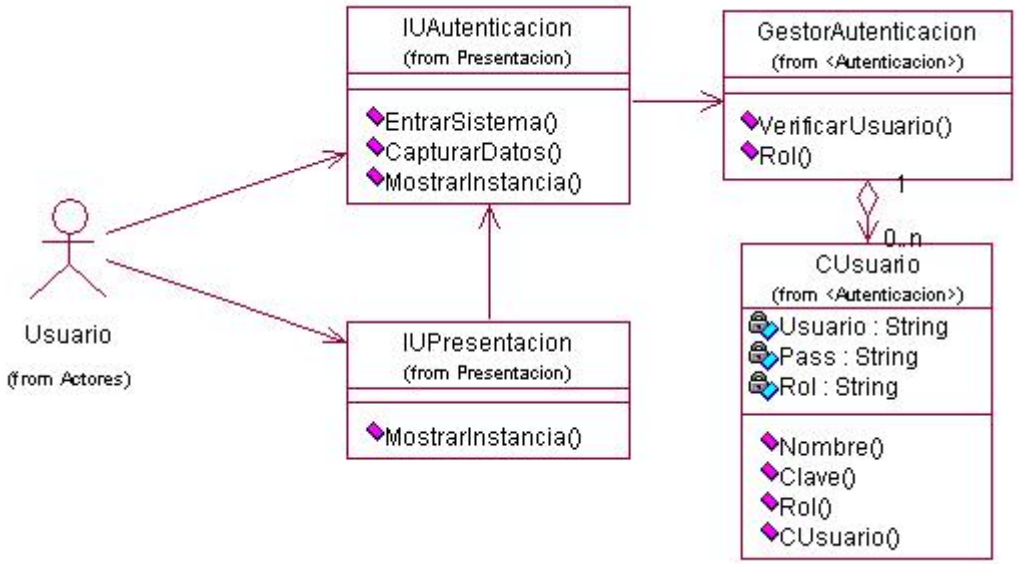


Figura 13: Diagrama de clases CU Autenticación

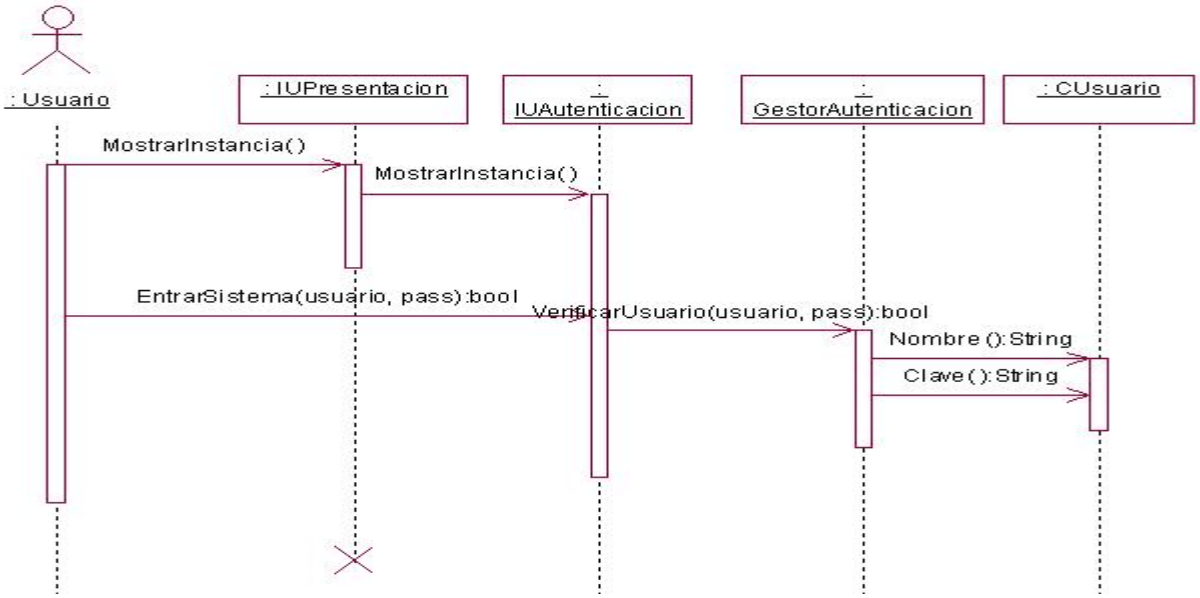


Figura 14: Diagrama de Secuencia del CU Autenticación

Este es el diagrama de secuencia para el flujo básico donde no ocurre ningún incidente que imposibilite la realización del CU. Dichos incidentes pueden ser: en el momento de autenticarse que se deje alguno de los dos campos a llenar vacíos o ambos, esto se controla y se lanza un mensaje al usuario solicitando llenar correctamente los campos. El otro y que si genera un flujo alterno de eventos es que el usuario o contraseña que se introduzca sea incorrecto.

Al verificar el usuario y contraseña se retornaría falso si no coincide con ninguno de los usuarios registrados en el sistema y se lanza un mensaje informando lo ocurrido y dando la posibilidad de volver a entrar los datos. El mensaje lo lanza IUAutenticacion.

*Flujo de Sucesos de Diseño*

El usuario levanta la aplicación, se muestra una ventana por un tiempo determinado dando la bienvenida a Predictor. Posteriormente viene la de autenticación, véase Anexo 10, quien da la posibilidad al usuario de introducir sus datos para trabajar con el software. Estos son verificados, en caso de que estén incorrectos, se lanza un mensaje de error y se posibilita introducirlos nuevamente. Si todo está normal, se capturan todos los datos del usuario que está trabajando y se muestra la forma principal de la aplicación, quién tiene todas las opciones para la realización de los demás casos de uso.

#### 2.5.2.2 Caso de Uso Configurar Conexión de 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 Proxy, número IP, el puerto, usuario y contraseña se guardan en un fichero que es cargado en el momento de la descarga para verificar si la conexión es correcta. Véase Anexo 9.

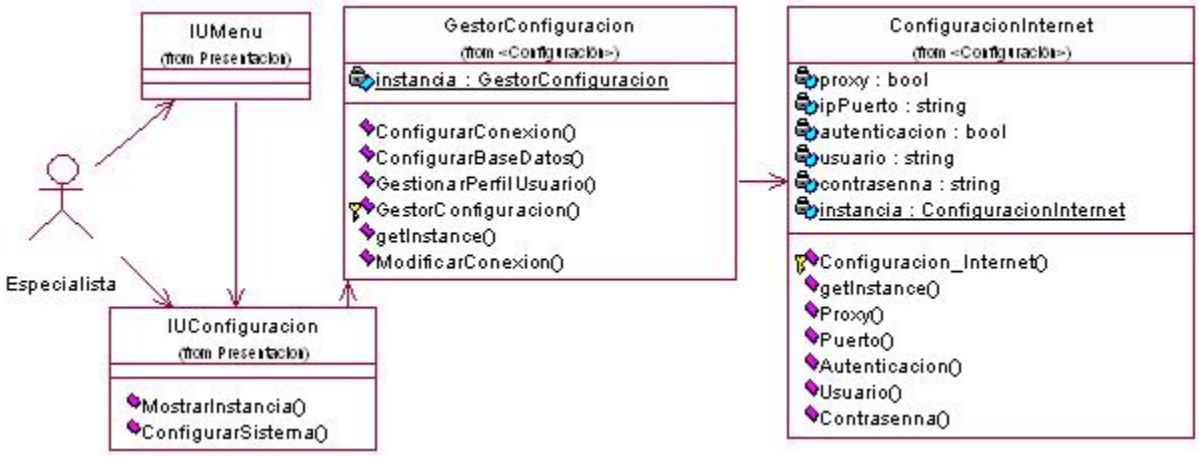


Figura 15: Diagrama de clases del CU Configuración de Internet

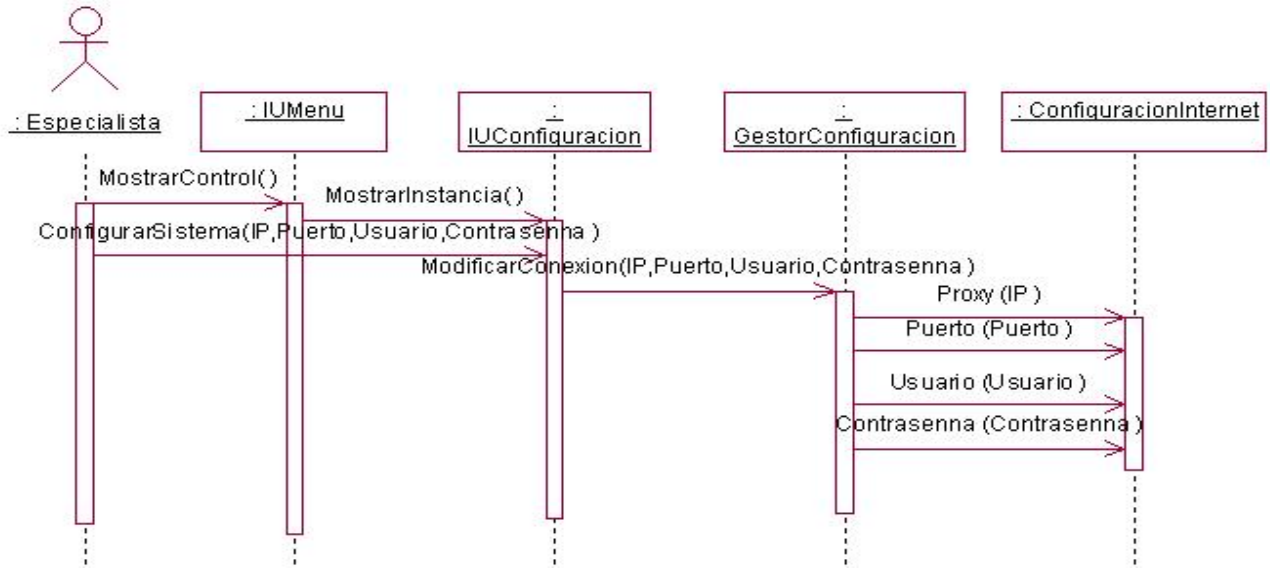


Figura 16: Diagrama de Secuencia CU Configuración Internet

*Flujo de Sucesos de Diseño*

El especialista selecciona en el menú la opción de configurar conexión de Internet, el sistema muestra el control que tiene los componentes que permite configurar la conexión. Introduce los datos, que pueden ser: tipo de conexión (con o sin Proxy), número IP del Proxy, puerto por el que se va a conectar, y si necesita conectarse como usuario. Con todos los datos introducidos, se modifica un fichero que se

encarga de guardar la configuración para ser usado en la descarga, y se lanza un mensaje informando que ha sido modificada la configuración.

#### 2.5.2.3 Caso de Uso Procesar Patentes

Este caso de uso 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.

Este caso de uso tiene que analizar cada fichero y seleccionar la patente que contenga. En los ficheros se encuentra el código *html* de la página. Este se carga en memoria y se seleccionan los parámetros de las patentes, hay que tener en cuenta que en todos los ficheros el algoritmo de selección no es el mismo, depende de las características de la página que fue descargado. Para buscar en la lista se implementa una búsqueda secuencial con centinela. Como resultado final se obtienen los ficheros finales con todas las patentes encontradas.

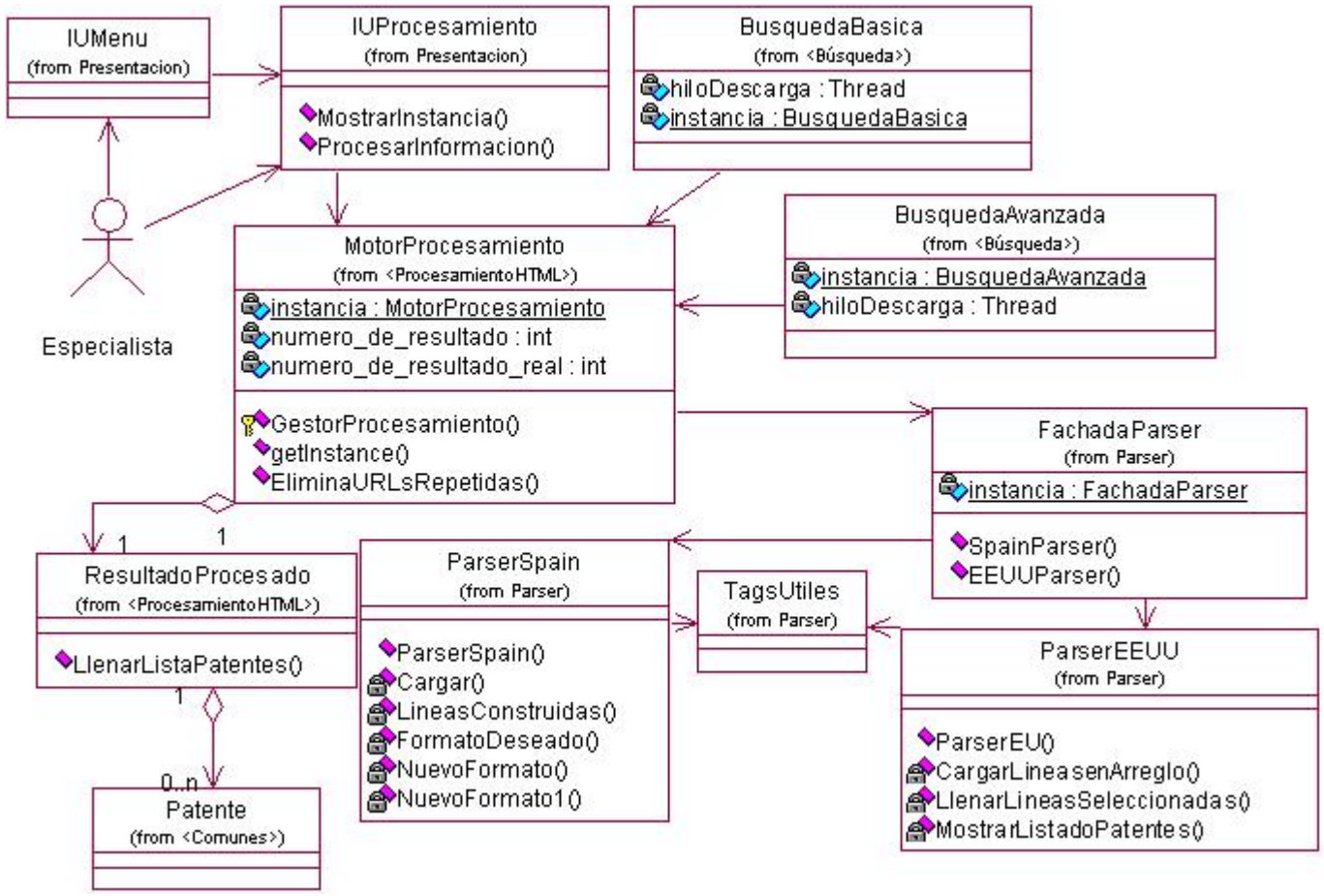


Figura 17: Diagrama de clases CU Procesar Patentes

Se incluyeron las clases del Subsistema Parser, aunque en el diagrama de secuencia solo se hace referencia a la FachadaParser, quien hace de fachada en el subsistema, permitiendo acceder a todas las funcionalidades que brinda mediante ella.

La clase GestorProcesamiento es instanciada por las clases gestoras de las búsquedas, en dependencia del tipo de búsqueda que se esté realizando en ese momento. Cuando se termina la descarga se comienza a procesar para obtener los ficheros finales.

La Figura 18 muestra una sección de este caso de uso, cuando es iniciado por el mismo sistema una vez que se está llevando a cabo la descarga. Y la Figura 19 cuando lo inicia el especialista para procesar una información que esté guardada en un directorio determinado.

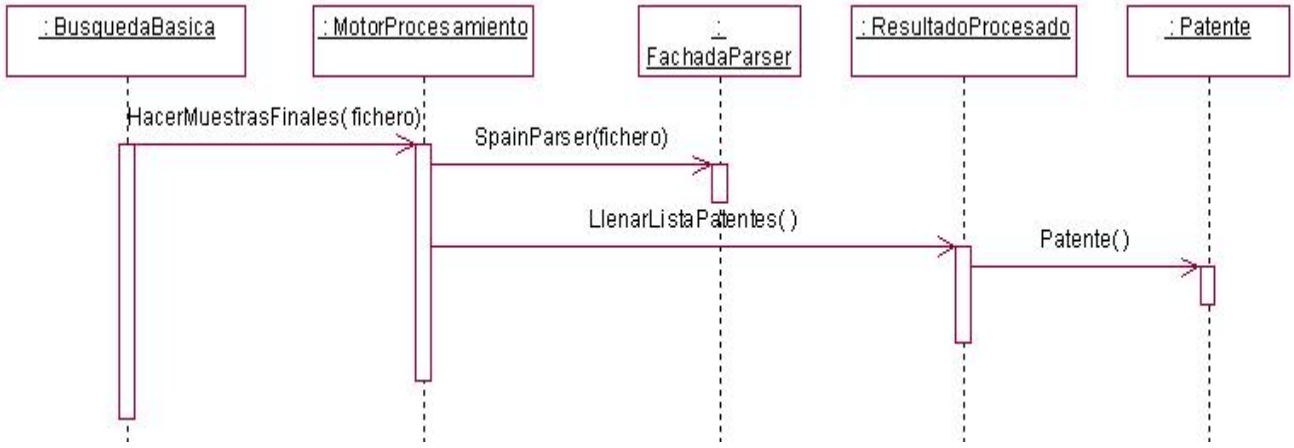


Figura 18: Diagrama de Secuencia CU Procesar Patentes (Sección 1)

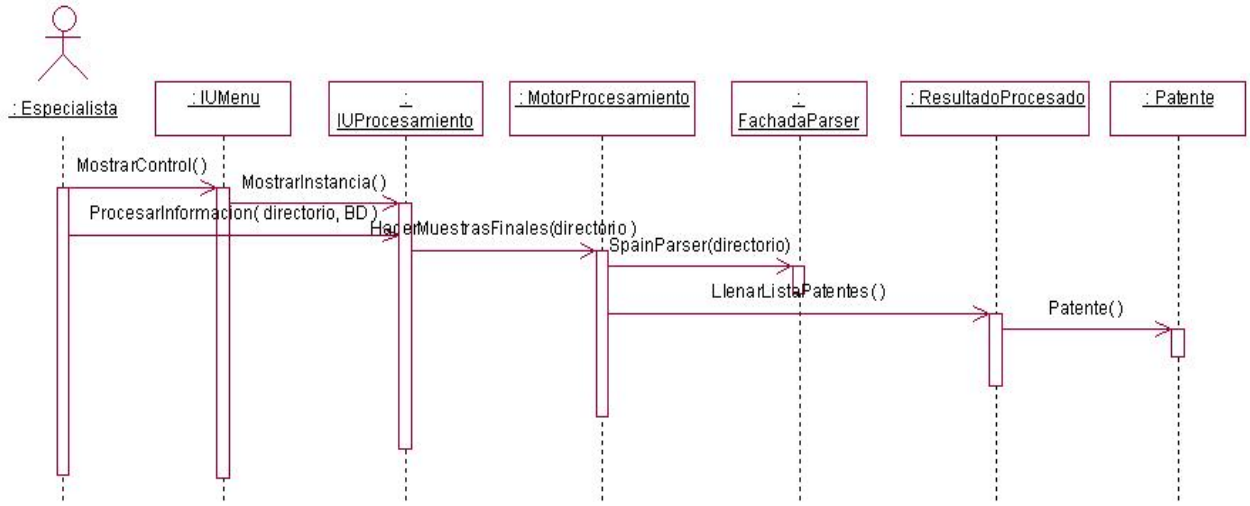


Figura 19: Diagrama de Secuencia CU Procesar Patentes (Sección 2)

Se escogió iniciar con GestorBusquedaBasica, pero puede ser el otro gestor también, eso depende del tipo de búsqueda hecha. Se obviaron las clases del Susbsistema Parser, debido a que se puede acceder a ellas desde FachadaParser. Se pueden realizar dos tipos de análisis, depende de la base de datos de la que se descargue la información. Para World Wide y Espacenet se invoca el método SpainParser y para la de EEUU el EEUUParser. Es así producto a que los ficheros tienen distintos formatos y se necesita recorrerlos y procesarlos de manera distinta.

*Flujo de Sucesos de Diseño*

Las clases gestoras de las búsquedas una vez terminada las descargas proceden a invocar a la clase GestorProcesamiento indicando el directorio donde están los ficheros que se van a procesar. Los mismos son analizados línea por línea buscando los parámetros de las patentes, hasta conformar los ficheros finales y crear los listados.

### 2.5.2.4 Caso de Uso Reportar Resultados

Permite mostrar los resultados de búsquedas realizadas anteriormente. Trabaja directamente con los ficheros generados por el procesamiento y muestra el listado de las patentes encontradas en la búsqueda seleccionada para reportar. Véase Anexo 14

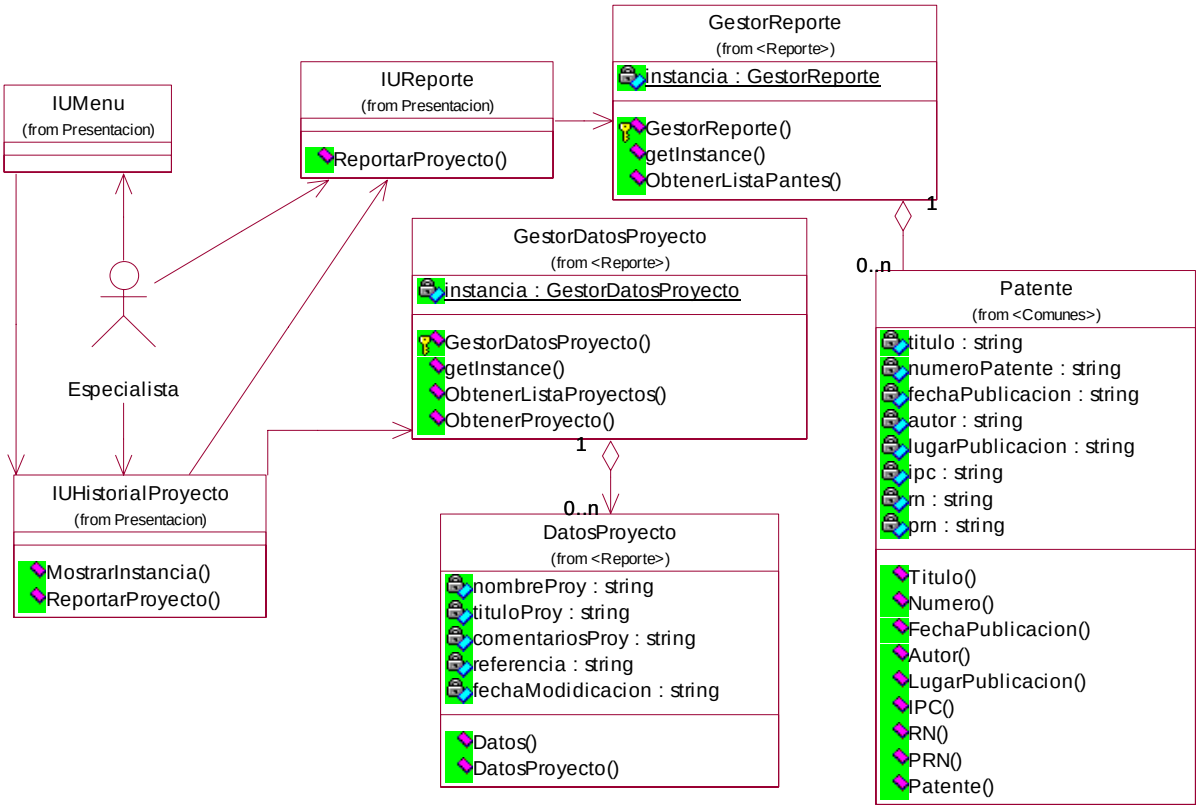


Figura 20: Diagrama de clases CU Reportar Resultados

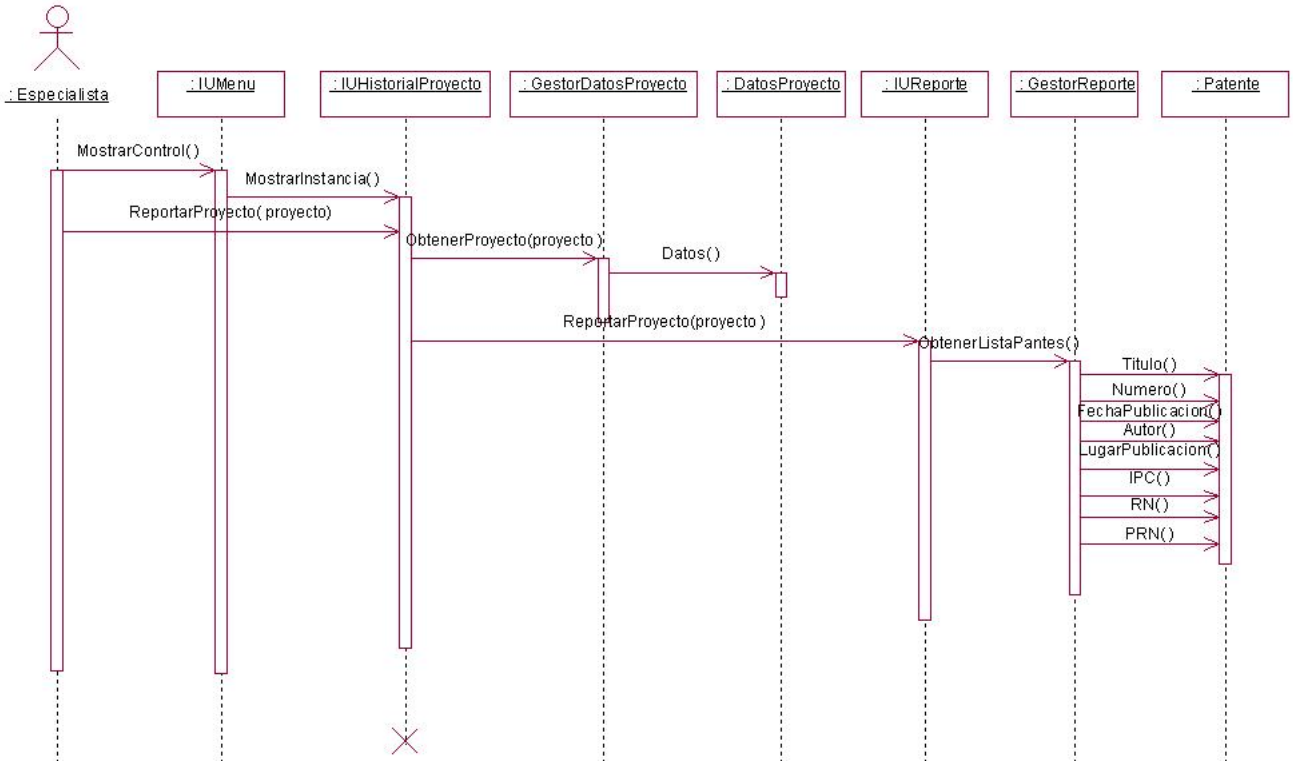


Figura 21: Diagrama de Secuencia CU Reportar Resultados

*Flujo de Sucesos de Diseño*

El usuario tiene la posibilidad de seleccionar cualquiera de las descargas que han sido realizadas por él para observar el reporte de las patentes encontradas. Se buscan los datos del proyecto seleccionado y se muestra otra ventana que tiene los componentes adecuados para mostrar el listado de las patentes con sus atributos.

### 2.5.2.5 Caso de Uso Búsqueda Básica

Su función 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. Véase Anexo 12.

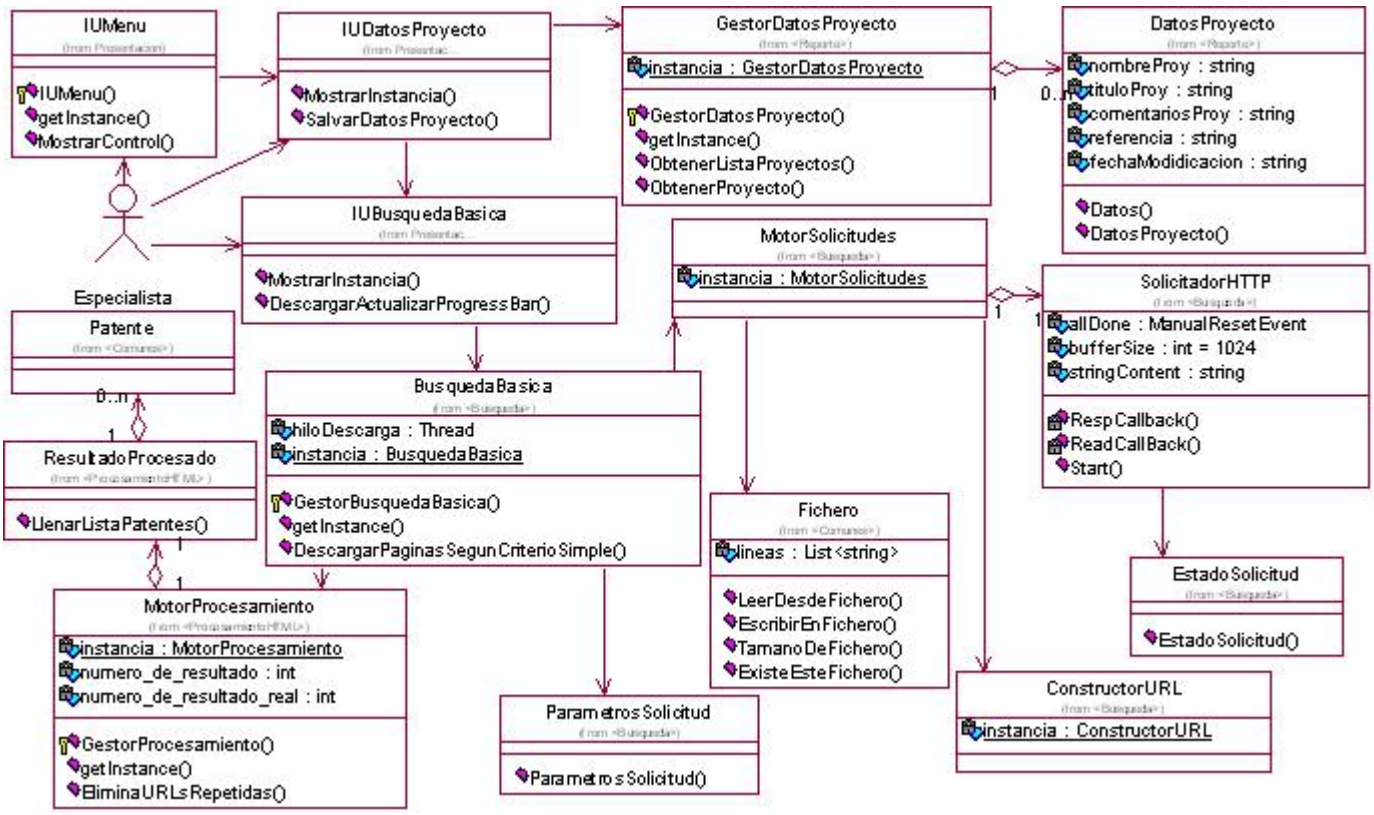


Figura 22: Diagrama de clases CU Búsqueda Básica

Por lo extenso del diagrama, se decidió dividirlo en dos partes, los parámetros de cada mensaje se explican más adelante.

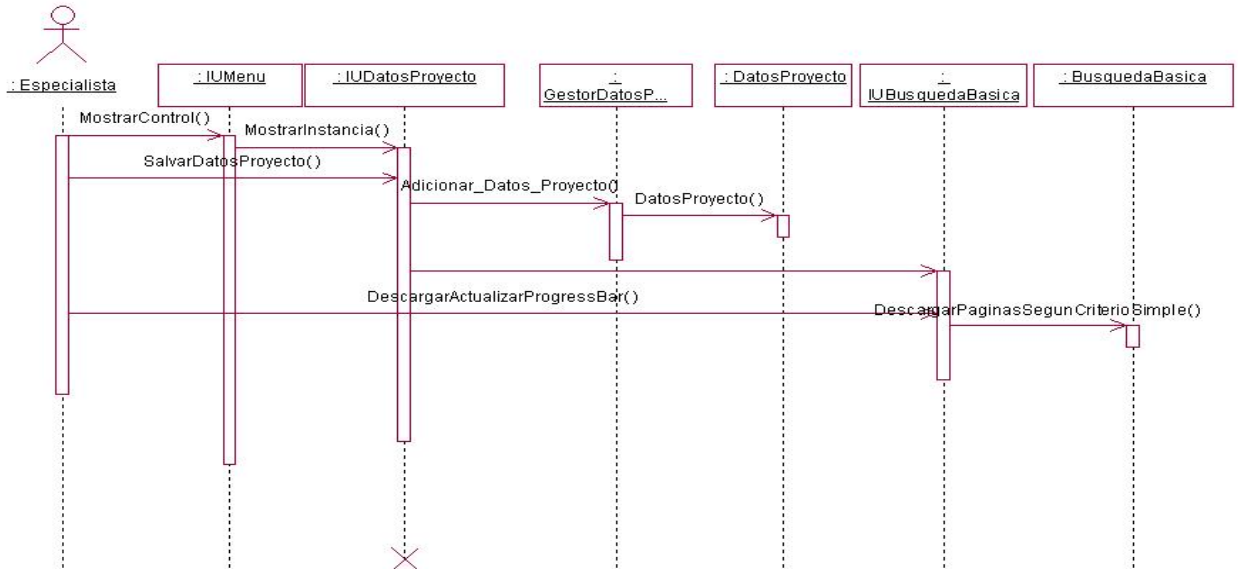


Figura 23: Diagrama de Secuencia CU Búsqueda Básica (Primera Parte)

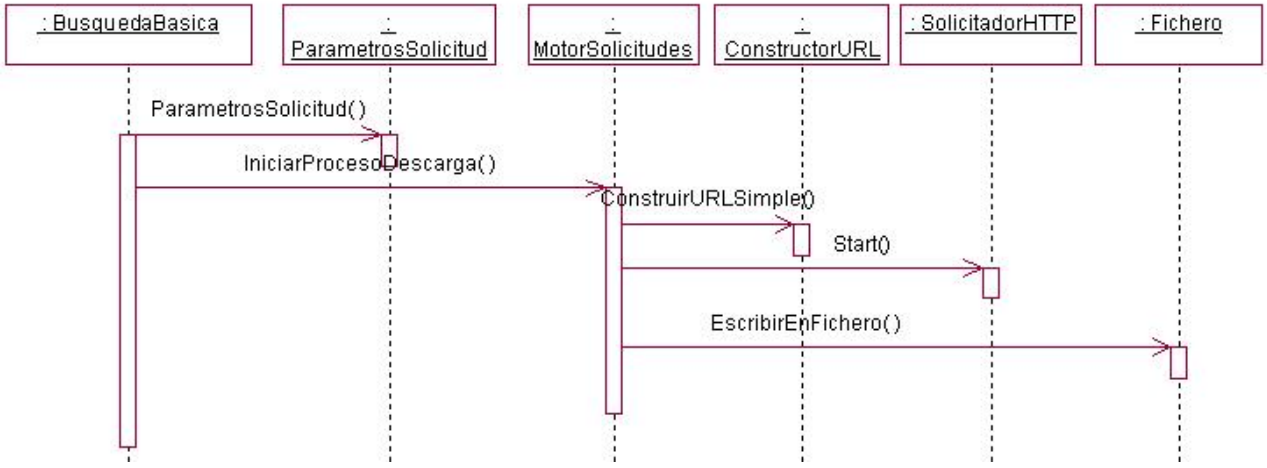


Figura 24: Diagrama de Secuencia CU Búsqueda Básica (Segunda Parte)

*Flujo de Sucesos de Diseño*

El usuario configura los datos para la descarga, nombre del proyecto, título, comentario y el directorio donde se guardará, véase Anexo 11. Estos datos son procesados por GestorDatosProcesados. Se levanta la ventana de búsqueda básica, donde se introduce el criterio y se selecciona la base de datos. La clase MotorSolicitudes inicia a descarga con los parámetros introducidos. Para eso construye la *url* de la página principal y la descarga. Ese fichero es procesado y se seleccionan las *urls* para seguir

descargando. Se procede a bajar cada una de las páginas encontradas y guardarlas en el directorio destinado por el usuario. Una vez terminada las descarga se instancia GestorProcesamiento para buscar todas las patentes dentro de los ficheros y confeccionar los ficheros finales con el listado de las patentes. Es importante decir que en el momento de iniciar la descarga se usan los datos de configuración de Internet.

A continuación se describen los mensajes del diagrama de secuencia, el diagrama completo se puede observar en el Anexo 15.

- *MostralControl:* El usuario selecciona en el menú la opción que desea ejecutar. El mensaje recibe como parámetro el control que va a ser cargado, en este caso es IUDatosProyecto.
- *MostrarIntancia:* Muestra una instancia de la interfaz de usuario correspondiente.
- *SalvarDatosProyecto:* El usuario entra los datos del nuevo proyecto. Recibe como parámetro el nombre, título, comentarios y carpeta donde se va a guardar el proyecto.
- *Adicionar_Datos_Proyecto:* Se crea un objeto DatosProyecto con los parámetros pasados en el mensaje anterior.
- *MostrarInstancia:* Carga el control de IUBusquedaBasica, no se le pasan parámetros.
- *DescargarActualizarProgressBar:* El usuario comienza el proceso de bajar información de Internet según los criterios deseados. Los parámetros son: criterio de búsqueda, donde se desea buscar, en el título o el resumen de la patente y la base de datos seleccionada.
- *DescargarPaginaSegunCriterioSimple:* Gestiona las descargas de las páginas de Internet según el criterio y la base de datos seleccionada.
- *ParametrosSolicitud:* Construye el objeto ParametrosSolicitud con los datos capturados en el mensaje anterior.
- *IniciarProcesoDescarga:* Con el ParametrosSolicitud creado anteriormente inicia la descarga.
- *ConstruirURLSimple:* Obtiene de ParametrosSolicitud el criterio de búsqueda y la base de datos, con lo que construye la *url* para la visitar y descargar la página.
- *Start*: Descarga la página que se haya visitado con la *url* construida.
- *EscribirEnFichero*: Escribe en un fichero ubicado en la carpeta donde se guarda el proyecto todo el código *html* de la página descargada.

### 2.5.2.6 Caso de Uso Búsqueda Avanzada

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. La búsqueda realizada es más específica que la simple, se tiene en cuenta la fecha de publicación, el solicitante, la clasificación internacional y otros aspectos, además del criterio de búsqueda.

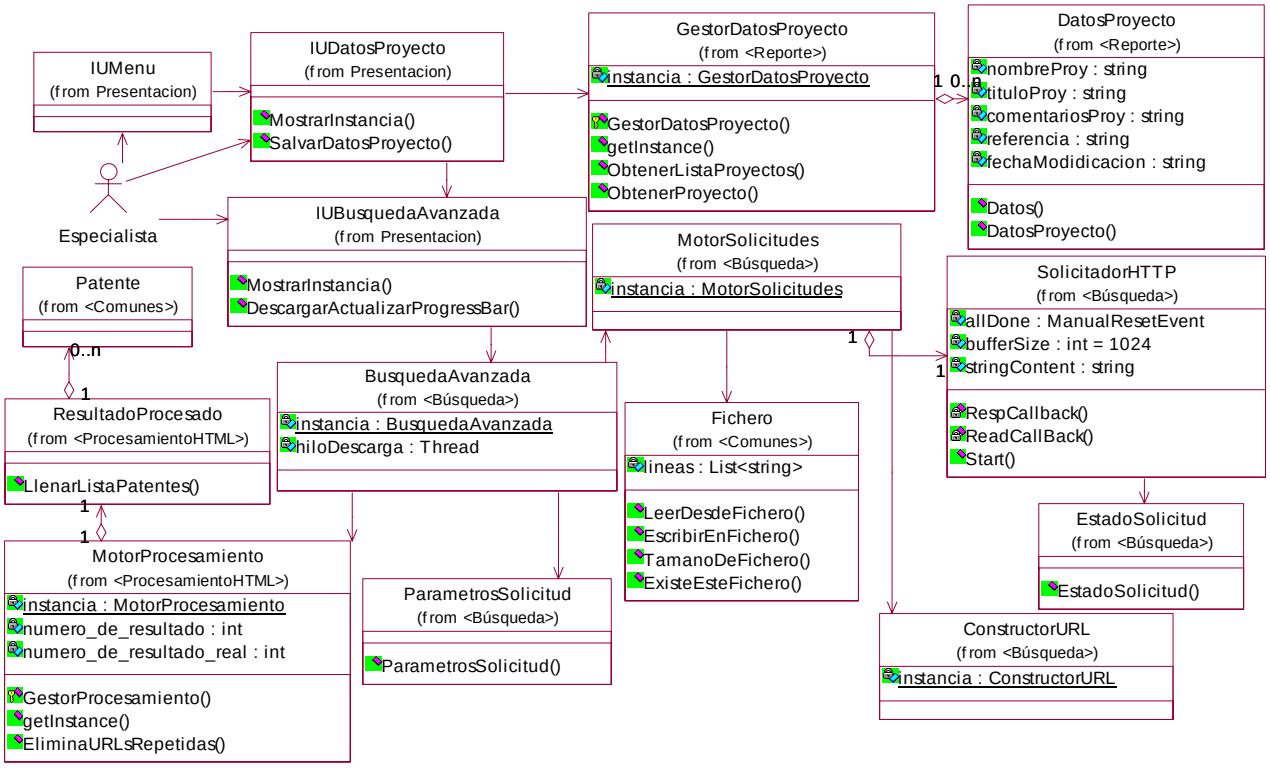


Figura 25: Diagrama de clases CU Búsqueda Avanzada

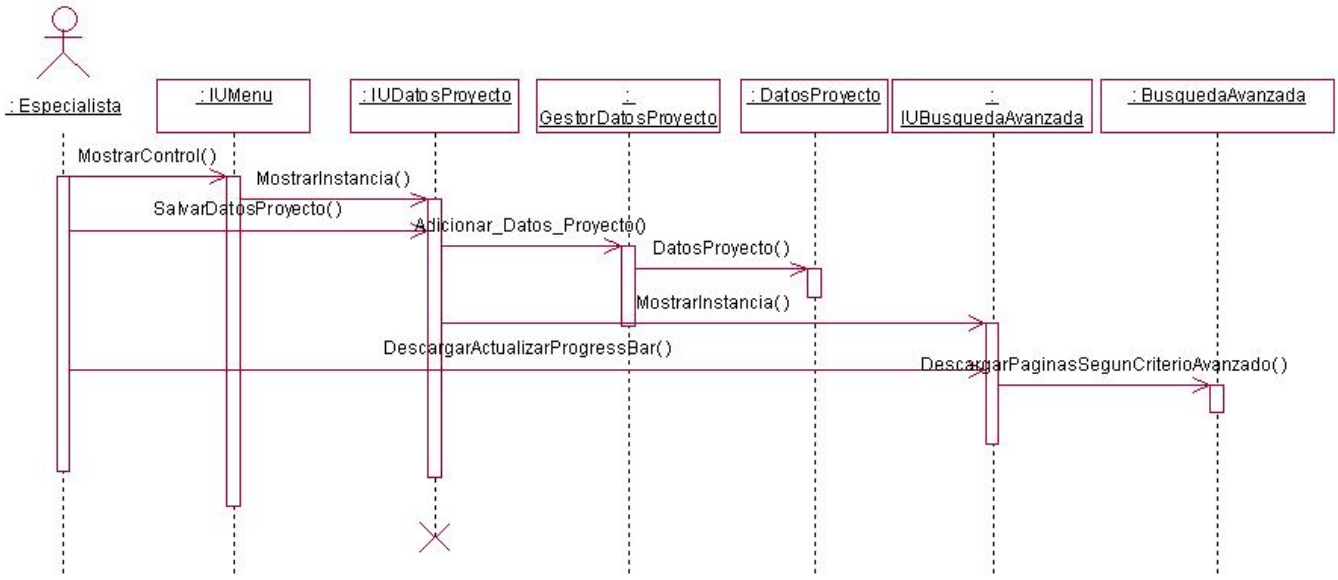


Figura 26: Diagrama de Secuencia CU Búsqueda Avanzada (Primera Parte)

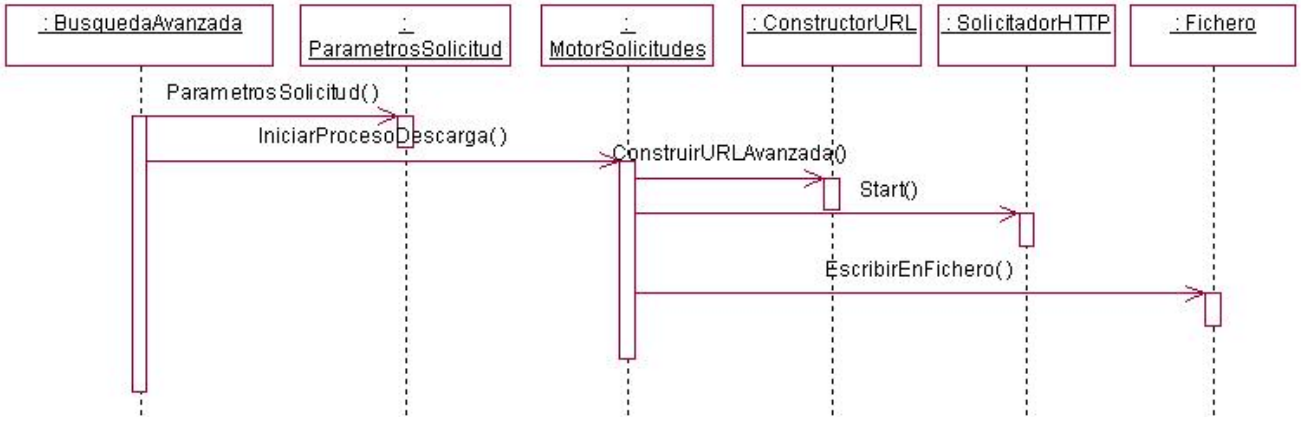


Figura 27: Diagrama de Secuencia CU Búsqueda Avanzada (Segunda Parte)

*Flujo de Sucesos de Diseño*

El usuario configura los datos para la descarga, nombre del proyecto, título, comentario y el directorio donde se guardará, véase Anexo 13. Estos datos son procesados por GestorDatosProcesados. Se levanta la ventana de búsqueda avanzada (IUBusquedaAvanzada), donde se introducen los parámetros para buscar, estos son: término de búsqueda, fecha de publicación, solicitante, clasificación internacional de la patente (IPC) y la base de datos. La clase MotorSolicitudes inicia la descarga con los parámetros introducidos. Para eso construye la *url* de la página principal y la descarga. Ese fichero es procesado y se

seleccionan las *urls* para seguir descargando. Se procede a bajar cada una de las páginas encontradas y guardarlas en el directorio destinado por el usuario. Una vez terminada las descarga se instancia GestorProcesamiento para buscar todas las patentes dentro de los ficheros y confeccionar los ficheros finales con el listado de las patentes. Es importante decir que en el momento de iniciar la descarga se usan los datos de configuración de Internet.

A continuación se describen los mensajes del diagrama de secuencia, el diagrama completo se puede observar en el Anexo 16.

- *MostralControl:* El usuario selecciona en el menú la opción que desea ejecutar. El mensaje recibe como parámetro el control que va a ser cargado, en este caso es IUDatosProyecto.
- *MostrarIntancia:* Muestra una instancia de la interfaz de usuario correspondiente.
- *SalvarDatosProyecto:* El usuario entra los datos del nuevo proyecto. Recibe como parámetro el nombre, título, comentarios y carpeta donde se va a guardar el proyecto.
- *Adicionar_Datos_Proyecto:* Se crea un objeto DatosProyecto con los parámetros pasados en el mensaje anterior.
- *MostrarInstancia:* Carga el control de IUBusquedaAvanzada, no se le pasan parámetros.
- *DescargarActualizarProgressBar:* El usuario comienza el proceso de bajar información de Internet según los criterios pasados. Los parámetros son: término de búsqueda, fecha de publicación, solicitante, clasificación internacional de la patente (IPC) y la base de datos
- *DescargarPaginaSegunCriterioSimple:* Gestiona las descargas de las páginas de Internet según los parámetros introducidos y la base de datos seleccionada.
- *ParametrosSolicitud:* Construye el objeto ParametrosSolicitud con los datos capturados en el mensaje anterior.
- *IniciarProcesoDescarga:* Con el ParametrosSolicitud creado anteriormente inicia la descarga.
- *ConstruirURLSimple:* Obtiene de ParametrosSolicitud el criterio de búsqueda y la base de datos, con lo que construye la *url* para la visitar y descargar la página.
- *Start*: Descarga la página que se haya visitado con la *url* construida.
- *EscribirEnFichero*: Escribe en un fichero ubicado en la carpeta donde se guarda el proyecto todo el código *html* de la página descargada.

Conclusiones

En este capítulo se trató todo lo referente al diseño del sistema. Se hace referencia al tipo de arquitectura, se justificaron los patrones usados, se explicaron los módulos que conforman a Predictor, los diferentes subsistemas que forman el Modelo de Diseño, las realizaciones de los casos de uso diseñados. Todo esto ayudo a llegar a las siguientes conclusiones:

- Es importante dividir correctamente en módulos, de forma tal que cada uno cumpla funcionalidades determinadas he independientes, contribuyendo así con el Bajo Acoplamiento.
- Los patrones de diseño aplicados hacen más eficiente, robusto y flexible el sistema.
- Los diferentes subsistemas diseñados posibilitan trabajar de manera independiente y desarrollar al mismo tiempo varios de ellos.
- Las realizaciones de los casos de uso brindan a los programadores suficiente información que permite programar distintas clases para que el sistema tenga las funcionalidades necesarias. Aunque es obligatorio trabajar con el archivo de generado por la herramienta case.
- Los diagramas de secuencia dan una visión adecuada de todos los mensajes que se intercambian entre las clases en el momento de realizar un caso de uso.
- Se desarrollaron correctamente los artefactos generados por el diseñador del sistema y que fueron planteados entre los objetivos.

## CAPÍTULO 3: ANÁLISIS DE RESULTADOS

Introducción

La medición es esencial para cualquier disciplina ingenieril, la ingeniería de software no es una excepción. Las métricas de software se refieren a un amplio rango de medidas para el software. Existen distintos tipos, entre ellas están: las de análisis, de diseño, de la codificación, las pruebas y otros aspectos dentro del proceso de desarrollo de un software.

En este capítulo se aplican algunas de las métricas tratadas durante el Capítulo 1 Véase 1.6 Métricas de Diseño de Software. Se usan con el fin de evaluar el diseño propuesto para la construcción de Predictor.

### 3.1 Acoplamiento

Este principio responde a uno de los Patrones de Asignación de Responsabilidades, (Bajo Acoplamiento). El diseño propuesto está formado por cuatro módulos principales, entre los cuáles se considera que existe un bajo acoplamiento. Eso lo demuestra la independencia que existe entre ellos, son muy pocas las llamadas entre uno y otro. El Módulo Descarga llama al Módulo Procesamiento cuando necesita realizar el trabajo con los ficheros que contienen el *html* de las páginas bajadas de las bases de datos de Internet para obtener los resultados finales de la descarga. En los demás módulos, todos trabajan por sí solos, cada uno cumpliendo con las funcionalidades para las cuales fueron creados. El Patrón Fachada se usó en todos los módulos, permitiendo crear una interfaz de acceso a ellos, abstrayendo al exterior de la estructura que tiene por dentro cada módulo, disminuyendo el acoplamiento entre ellos.

En las realizaciones de los casos de usos explicados en el capítulo anterior, se puede observar perfectamente la relación existente entre los módulos.

La mayoría de los componentes son reutilizables por si solos, son independientes uno de otro, se puede hacer uso de ellos en otras aplicaciones, por ejemplo, el módulo de descarga se puede usar para descargar información de cualquier buscador, solo se necesita incluir la *url* del buscador que se desee.

### 3.2 Complejidad de los Módulos

A continuación se calcula la complejidad de los cuatro módulos principales que conforman el sistema según las fórmulas planteadas en el capítulo 1. 1.6.1 Métricas de diseño arquitectónico.

- *Complejidad estructural:* $S(i) = f^2_{out}(i)$.

- *Complejidad de datos:* $D(i) = v(i) / \left[ f_{out}(i) + 1 \right]$.
- *Complejidad del sistema:* $C(i) = S(i) + D(i)$.

*Módulo Configuración*


Figura 28: Expansión del Módulo Configuración

Tabla 2: Complejidad Módulo Configuración

| **Complejidad estructural** $S(i)$ | **Complejidad de datos** $D(i)$ | **Complejidad del sistema** $C(i)$ |
|---|---|---|
| $S(i) = 0^2 = 0$ | $D(i) = 4 / [0+1] = 4$ | $C(i) = 0 + 4 = 4$ |

*Módulo Procesamiento*

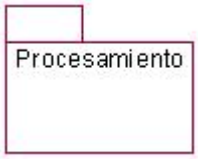


Figura 29: Expansión del Módulo Procesamiento

Tabla 3: Complejidad Módulo Procesamiento

| **Complejidad estructural** $S(i)$ | **Complejidad de datos** $D(i)$ | **Complejidad del sistema** $C(i)$ |
|---|---|---|
| $S(i) = 0^2 = 0$ | $D(i) = 4 / [0+1] = 4$ | $C(i) = 0 + 4 = 4$ |

*Módulo Descarga*


Figura 30: Expansión del Módulo Descarga

Tabla 4: Complejidad Módulo Descarga

| **Complejidad estructural** $S(i)$ | **Complejidad de datos** $D(i)$ | **Complejidad del sistema** $C(i)$ |
|---|---|---|
| $S(i) = 1^2 = 1$ | $D(i) = {}^{10}\!/_{[1+1]} = 5$ | $C(i) = 1 + 5 = 6$ |

*Módulo Reportes*


Figura 31: Expansión del Módulo Reportes

Tabla 5: Complejidad Módulo Reportes

| **Complejidad estructural** $S(i)$ | **Complejidad de datos** $D(i)$ | **Complejidad del sistema** $C(i)$ |
|---|---|---|
| $S(i) = 0^2 = 0$ | $D(i) = {}^{1}\!/_{[0+1]} = 1$ | $C(i) = 0 + 1 = 1$ |

*Resultados*

Tabla 6: Complejidad de los Módulos

| **Configuración** | **Procesamiento** | **Descarga** | **Reportes** |
|---|---|---|---|
| 4 | 4 | 6 | 1 |

Como se puede observar, ninguno de los módulos es considerado complejo. Los valores tomados por la complejidad del sistema en todos los casos son muy bajos. El de Reportes solo tiene como entrada la dirección donde está el proyecto que se desea reportar. En los demás casos los valores son muy parecidos, pero el de Descarga y Procesamiento son los más complejos. El primero debido a que invoca a Procesamiento para realizar sus funcionalidades y además requiere de varios parámetros de entrada. Y el segundo por los parámetros de entrada que necesita, a pesar de no tener relación con otro módulo.

Es necesario mencionar que la mayoría de los autores que tratan el tema de las métricas de diseño arquitectónico proponen tres tipos de sistemas según su complejidad, ellos son:

- No muy complejo.
- Complejo
- Muy complejo.

Para saber cuando un sistema está dentro de alguno de los tres grupos se proponen umbrales. Para los no muy complejos tiene que cumplirse: $D(i) \leq 7$ y $C(i) \leq 32$, lo que demuestra que Predictor es un sistema no muy complejo.

## 3.3 Métricas orientadas a clases

### 3.3.1 Tamaño de Clase (TC).

Para medir el tamaño de las clases (TC), se tienen en cuenta los aspectos siguientes:

- Total de operaciones, ya sean las de ella o las heredadas de las clases padres o interfaces que implementen.
- Cantidad de atributos, al igual que el anterior, tanto los de ella, como los de los padres.
- Promedio general de los dos anteriores para el sistema completo.

Para evaluar las métricas son necesarios los valores de los umbrales, aspecto que es muy discutido por los especialistas en el tema, y dependen mucho del sistema que se esté diseñando. Las clases se clasifican en tres grupos según su tamaño, los que se presentan en la siguiente tabla junto con los umbrales seleccionados para su clasificación.

Tabla 7: Valores de los umbrales para TC

| Clasificación | Valores de los umbrales |
|---|---|
| Pequeño | <=20 |
| Medio | >20 y <=30 |
| Grande | >30 |

**La**
**Tabla 8** ilustra las clases del sistema aplicándole la métrica seleccionada.

Tabla 8: Tamaño de las clases de negocio

| No | Nombre | Cantidad Atributos | Cantidad Operaciones | Tamaño |
|---|---|---|---|---|
| 1 | Fichero | 1 | 4 | Pequeño |
| 2 | BaseDatosDisponibles | 3 | 0 | Pequeño |
| 3 | Patente | 8 | 9 | Pequeño |
| 4 | Observado | 1 | 3 | Pequeño |

| | | | | |
|---|---|---|---|---|
| 5 | IObservador | 0 | 8 | Pequeño |
| 6 | IMostrable | 0 | 3 | Pequeño |
| 7 | INavegadorWEB | 0 | 1 | Pequeño |
| 8 | IProcesable | 0 | 8 | Pequeño |
| 9 | MotorProcesamiento | 3 | 3 | Pequeño |
| 10 | ResultadoProcesado | 1 | 1 | Pequeño |
| 11 | FachadaParser | 1 | 2 | Pequeño |
| 12 | ParserSpain | 6 | 6 | Pequeño |
| 13 | ParserEEUU | 3 | 4 | Pequeño |
| 14 | TagsUtiles | 0 | 46 | Grande |
| 15 | IParser | 0 | 2 | Pequeño |
| 16 | IConfigurable | 0 | 2 | Pequeño |
| 17 | GestorConfiguracion | 1 | 6 | Pequeño |
| 18 | ConfiguracionInternet | 6 | 7 | Pequeño |
| 19 | ParametrosSolicitud | 10 | 11 | Medio |
| 20 | BusquedaBasica | 2 | 3 | Pequeño |
| 21 | BusquedaAvanzada | 2 | 3 | Pequeño |
| 22 | ConstructorURL | 1 | 10 | Pequeño |
| 23 | MotorSolicitudes | 4 | 13 | Pequeño |
| 24 | SolicitadorHTTP | 3 | 3 | Pequeño |
| 25 | EstadoSolicitud | 6 | 1 | Pequeño |
| 26 | IProyectoReportable | 0 | 2 | Pequeño |
| 27 | DatosProyecto | 5 | 8 | Pequeño |
| 28 | GestorDatosProyecto | 1 | 6 | Pequeño |
| 29 | GestorReporte | 1 | 3 | Pequeño |
| 30 | CUsuario | 3 | 4 | Pequeño |
| 31 | GestorAutenticacion | 0 | 2 | Pequeño |

Cuando existe un TC grande afecta los aspectos de calidad definidos por esta métrica. Se reduce la reutilización de las clases, se hace más compleja su implementación, las pruebas son difíciles de realizar y aumenta la responsabilidad de las clases.

La mayoría de las clases que conforman el sistema entran dentro de la clasificación de pequeñas, lo que demuestra que el sistema no es complejo, no es difícil de implementar, las pruebas se les pueden realizar fácilmente y estas clases son reutilizables. Los resultados obtenidos son positivos según esta métrica, así se ilustra en las siguientes tablas.

Tabla 9: Cantidad de clases por clasificación

| **Clasificación** | **Cantidad Clases** |
|---|---|
| Pequeño | 29 |
| Medio | 1 |
| Grande | 1 |

Tabla 10: Resultados de la Métrica TC

| **Cantidad Clases** | **Promedio Atributos** | **Promedio Operaciones** |
|---|---|---|
| 31 | 2.32 | 5.93 |

### 3.3.2 Árbol de Profundidad de Herencia (APH)

En el capítulo 1 se tratan las métricas propuestas por estos especialistas. Entre ellas está el árbol de profundidad de herencia (APH)

*APH*

Está métrica está definida por la máxima longitud que exista entre el nodo y la raíz del árbol. Donde el nodo es una clase hija que hereda de una clase, y así respectivamente hasta llegar a la raíz. A medida que esa longitud va creciendo, entonces se van heredando más operaciones y atributos por las clases hijas. Se hace difícil predecir el comportamiento de las clases que se encuentran en los niveles más bajos del árbol. Esta tiene sus ventajas y desventajas. Si los valores de APH son grandes, entonces se garantiza que se reutilice gran cantidad de código; pero al mismo tiempo hace que el diseño sea más complejo. Esto provoca un mayor acoplamiento entre las clases.

En Predictor no hay necesidad de hacer demasiado uso de la herencia. En la mayoría de los casos es mediante las interfaces para que varias clases hereden el comportamiento de ellas, y además para que otras puedan heredar de una clase normal y de una interface. Quedando solucionado el problema de la herencia múltiple.

Aplicando esta métrica al diseño propuesto se obtienen resultados que demuestran su poca complejidad. El árbol de profundidad de herencia toma valor 2, por lo que existe bajo acoplamiento y es de fácil reparación.

### 3.3.3 Relaciones entre Clases (RC)

Esta métrica está dada por la cantidad de relaciones de uso que existe entre las distintas clases que forman el diseño propuesto. Se le aplica a las mismas clases que le fue aplicada la métrica TC. Los aspectos de calidad que se miden son: Acoplamiento, Complejidad de mantenimiento, Reutilización y Cantidad de pruebas.

Tabla 11: Cantidad de relaciones de uso entre las clases

| **No** | **Nombre** | **Relaciones de uso** |
|---|---|---|
| 1 | Fichero | 0 |
| 2 | BaseDatosDisponibles | 0 |
| 3 | Patente | 0 |
| 4 | Observado | 1 |
| 5 | IObservador | 0 |
| 6 | IMostrable | 0 |
| 7 | INavegadorWEB | 0 |
| 8 | IProcesable | 0 |
| 9 | MotorProcesamiento | 3 |
| 10 | ResultadoProcesado | 1 |
| 11 | FachadaParser | 2 |
| 12 | ParserSpain | 2 |
| 13 | ParserEEUU | 2 |
| 14 | TagsUtiles | 0 |

| | | |
|---|---|---|
| 15 | IParser | 0 |
| 16 | IConfigurable | 0 |
| 17 | GestorConfiguracion | 1 |
| 18 | ConfiguracionInternet | 0 |
| 19 | ParametrosSolicitud | 0 |
| 20 | BusquedaBasica | 2 |
| 21 | BusquedaAvanzada | 2 |
| 22 | ConstructorURL | 0 |
| 23 | MotorSolicitudes | 3 |
| 24 | SolicitadorHTTP | 1 |
| 25 | EstadoSolicitud | 0 |
| 26 | IProyectoReportable | 0 |
| 27 | DatosProyecto | 0 |
| 28 | GestorDatosProyecto | 1 |
| 29 | GestorReporte | 1 |
| 30 | CUsuario | 0 |
| 31 | GestorAutenticacion | 1 |

Tabla 12: Resumen de RC

| **Cantidad de Clases** | **Relaciones de uso** |
|---|---|
| 17 | 0 |
| 7 | 1 |
| 5 | 2 |
| 2 | >2 |

Para medir el acoplamiento según los resultados de esta métrica, algunos especialistas plantean los siguientes valores:

Tabla 13: Acoplamiento

| **Categoría** | **Relaciones de uso** | **Cantidad de Clases** |
|---|---|---|
| Ninguno | 0 | 17 |
| Bajo | 1 | 7 |
| Medio | 2 | 5 |
| Alto | >2 | 2 |

Los demás parámetros de calidad que mide esta métrica dependen del valor promedio de las dependencias de uso entre todas las clases, en este caso ese promedio es 0.74.

Tabla 14: Cantidad de Pruebas y Complejidad de Mantenimiento

| **Categoría** | **Criterio** | **Cantidad de Clases** |
|---|---|---|
| Baja | <= Prom. | 17 |
| Media | >Prom. Y <=2*Prom. | 7 |
| Alta | >2*Prom. | 7 |

Tabla 15: Reutilización

| **Categoría** | **Criterio** | **Cantidad de Clases** |
|---|---|---|
| Baja | >2*Prom. | 7 |
| Media | >Prom. Y <=2*Prom. | 7 |
| Alta | <= Prom. | 17 |

De manera general, los resultados de esta métrica son positivos. El acoplamiento existente entre las clases es bajo, el 55% no tiene ningún acoplamiento, el 23% es bajo, el 16% medio, y solo el 6% tiene alto acoplamiento. La nivel reutilización de las clases es bastante bueno, el 77% de las clases puede es fácil de reutilizar. Lo mismo pasa con la cantidad de pruebas y la complejidad de mantenimiento, el 77% de las clases son fáciles de reparar y la cantidad de pruebas a realizar el relativamente poca.

Conclusiones

Al aplicar estas métricas al diseño propuesto se arribaron a las siguientes conclusiones:

- Los módulos más complejos son los de Descarga y Procesamiento, lo que demuestra que se necesita realizar más esfuerzo para su construcción y se le deben asignar más recursos para ello.
- El tamaño de la mayoría de las clases que conforman el diseño es pequeño. Las clases consideradas pequeñas representan el 94 % del sistema, lo que demuestra que el diseño propuesto tiene calidad, las clases son de fáciles de probar, implementar y son reutilizables.
- La relación que existe entre las clases que heredan de otras demuestra que el diseño no es complejo y cumple con uno de los patrones GRASP, el de Bajo Acoplamiento.
- De forma general, las métricas aplicadas demuestran que la solución propuesta tiene calidad. Los valores que toman los indicadores en cada uno están dentro de los rangos aceptados.

## CONCLUSIONES

- El estudio realizado sobre los conceptos, principios y patrones de diseño, así como algunas de las métricas que permiten evaluar el diseño fue útil. Sirvieron como base para diseñar el sistema y luego comprobar su calidad.
- Se cumplió el objetivo por el cuál se llevó a cabo este trabajo. Realizar el diseño del sistema para la descarga y procesamiento de patentes procedentes de bases de datos de Internet.
- Se utilizaron patrones de diseño que posibilitan un sistema robusto, flexible y poco complejo.
- Se generaron los artefactos necesarios que sirven de entrada al flujo de trabajo de implementación. Modelo de Diseño, Subsistemas de Diseño, Clases de Diseño, Realización de los CU del Diseño.
- Los valores tomados por los indicadores en cada una de las métricas aplicadas demuestran que el diseño tiene calidad. La mayoría de las clases son reutilizables, fáciles de implementar, probar y el sistema es poco complejo.
- El diseño propuesto brinda la posibilidad de construir el primer software cubano de descarga y procesamiento de patentes. El cual resolverá el problema existente actualmente en Delfos con las demoras en las descargas y análisis de información procedente de las patentes.

**RECOMENDACIONES**

Para próximas versiones del producto:

- Migrar la solución hacia software libre, valorar la idea de implementarlo en Java o Mono.
- Profundizar en la forma que los softwares existentes en el mercado mundial crean los gráficos y mapas con la información de las patentes, para incluirlo en Predictor, siendo esto de gran ayuda para los análisis.
- Estudiar los distintos métodos de parseo de *html* que se usan para valorar y tratar de hacer más genérico el algoritmo de procesamiento de la información.
- Incluirle nuevas bases de datos para ampliar el campo de búsqueda de los especialistas.
- Usar el Patrón Command para proporcionar más flexibilidad en las interfaces gráficas.

**BIBLIOGRAFÍA**

[1]. CDE. Curso Online. Vigilancia Tecnológica e Inteligencia Competitiva. Las Patentes, 2006. [Disponible en: http://www.zaintek.net/ebizkaia/Cursos/Curso_1/menu/pdf/3.pdf ]

[2]. Rodríguez, H. Trayectoria Innovativa y Estrategias Tecnológicas en los Procesos FCC: un Análisis de Patentes Otorgadas en Estados Unidos 1976-2000. Universidad Autónoma Metropolitana-Xochimilco.

[3]. Larreina, Hernando, Grasaleña. La evolución de la inteligencia competitiva: Un estudio de las herramientas cienciométricas, 2006. [Disponible en: http://redalyc.uaemex.mx/redalyc/pdf/548/54852001.pdf ]

[4]. CDE. Curso Online. Vigilancia Tecnológica e Inteligencia Competitiva. Las Patentes, 2005. [Disponible en: http://www.zaintek.net/ebizkaia/Cursos/Curso_1/menu/pdf/3.pdf ]

[5]. Cárdenas, G. Herramientas cienciométricas. Experiencia de su aplicación en la Consultoria Biomundi, 2006. [Disponible en: http://www.idea.gov.ve/intempres/Intempres2006/Propuestos%20a%20posters/GEMA_INTEMPRES.pdf ]

[6]. Presuman, R. Ingeniería del Software. Un enfoque práctico. La Habana, Félix Varela, 2005. 601

[7]. Larman, C. UML y PATRONES. Introducción al análisis y diseño orientado a objetos. La Habana, Félix Varela, 2004. Cant Paginas

[8]. Jacobson, Booch, Rumbaugh. El proceso unificado de desarrollo de software. La Habana, Félix Varela, 2004. 438

[9]. Plaza, M., Piñeiro, Y. Predictor: Sistema de descarga y procesamiento automatizado de patentes. Rol Analista de Sistema. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2007.

[10]. Jesús, S. Placencia, M. Predictor: Sistema de descarga y procesamiento automatizado de patentes. Rol Arquitecto. Universidad de las Ciencias Informáticas. Ciudad de la Habana, 2007.

[11]. Saballo, L. Sistema para la descarga y procesamiento automatizado de Patentes. Universidad de las Ciencias Informáticas. Caracas, 2007.

[12]. Sixto. ¿Patrones?, 2003. [Disponible en: http://www.code4net.com/archives/000030.html ]

[13]. Mendoza, M. Metodologías de Desarrollo de Software, 2004. [Disponible en: http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf ]

[14]. Rossi, B. Métricas de Software, 2004. [Disponible en: http://www.itba.edu.ar/capis/rtis/articulosdeloscuadernosetapaprevia/ROSSI-METRICAS.pdf ]

[15]. Erich Gamma, R. H., Ralph Johnson, John Vlissides, ForeWord by Grady Booch. Desing Patterns, Elements of Reusable Object-Oriented Software, 1994.

[16]. Piug. Patent Information Vendor Sites, 2006. [Disponible en: http://www.piug.org/vendor.html ]

[17]. UAM. Patentes. Bases de Datos Internacionales, 2006. [Disponible en: http://www.ibt.unam.mx/biblioteca/patentes.htm ]

[18]. OEPM. esp@cenet, 2006. [Disponible en: http://www.oepm.es/internet/bases_datos/espa.htm ]

[19]. esp@cenet. Introducción a espacenet Version 3, 2006 [Disponible en : http://es.espacenet.com/search97cgi/s97_cgi.exe?Action=FormGen&Template=es/ES/home.hts ]

[20]. esp@cenet. Introduction to esp@cenet, 2006. [Disponible en: http://es.espacenet.com/search97cgi/s97_cgi.exe?Action=FormGen&Template=es/ES/info.hts&INFO=intro ]

[21]. Delphion. PatentLab-II, 2006. [Disponible en: http://www.delphion.com/products/research/products-patlab ]

[22]. CDE. Xerka, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=view&id=10&Itemid=334 ]

[23]. CDE. Matheo Software, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=category&sectionid=13&id=67&Itemid=300 ]

[24]. CDE. Vigilancia de Patentes, 2006. [Disponible en: http://www.cde.es/index.php?option=com_content&task=view&id=20&Itemid=285 ]

[25]. Olmedilla, J. Revisión Sistemática de Métricas de Diseño Orientado a Objetos, 2005. [Disponible en: http://is.ls.fi.upm.es/doctorado/Trabajos20042005/Olmedilla.pdf ]

**GLOSARIO**

- *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.
- *Base Datos EEUU*: Es una base de datos libre que brinda la Oficina de Patentes y Marcas de EEUU.
- *Caso de Uso*: Fragmentos de funcionalidad que el sistema ofrece para aportar un resultado de valor para sus actores.
- *Componente*: Es una parte física y reemplazable de un sistema que conforman un conjunto de interfaces y proporciona la implementación de dicho conjunto.
- *Clase*: Conjunto de objetos que comparten atributos, operaciones, relaciones y semántica.
- *Delfos:* Consultoría del Ministerio de la Informática y las Comunicaciones.
- *Esp@cenet*: Es un sitio Web gratuito en línea para buscar patentes y solicitudes. Fue desarrollado por los EPO junto a los estados miembros de la organización europea de patentes.
- *Gof*: Gang of Four. Banda o pandilla de los cuatro, nombre del equipo conformado por cuatro personas muy destacadas en el tema de los patrones de diseño.
- *HTML*: Lenguaje de marcas o etiquetas.
- *IEEE*: Instituto de Ingenieros Eléctricos y Electrónicos.
- *Ingeniería de Software*: Es una tecnología multicapa en la que se pueden identificar: los métodos (indican cómo construir técnicamente el software), el proceso (es el fundamento de la Ingeniería de Software, es la unión que mantiene juntas las capas de la tecnología) y las herramientas (soporte automático o semiautomático para el proceso y los métodos).
- *Interacción*: Conjunto de mensajes intercambiados entre un conjunto de objetos para alcanzar un propósito específico.
- *Interface*: Clase que proporciona C # para darle solución al problema de la herencia múltiple. Solamente contiene métodos que serán definidos en aquellas clases que la implementen.
- *Patente*: 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 (como representante por ejemplo) por un espacio limitado de tiempo (generalmente 20 años desde la fecha de aprobación).
- *POO*: Programación Orientada a Objetos.

- *Predictor*: Nombre dado al sistema por la Consultoría del Ministerio de la Informática y las Comunicaciones.
- *ProgressBar*: Componente perteneciente a la plataforma .NET
- *Proxy*: es un programa intermediario que actúa a la vez como servidor y cliente para realizar demandas de otros clientes.
- *Requerimiento o Requisito*: Condición o capacidad que tiene que tener el sistema para satisfacer un contrato o documento formal.
- *Requerimientos No Funcionales*: Los requerimientos no funcionales son propiedades o cualidades que el producto debe tener.
- *Requerimientos Funcionales*: Los requerimientos funcionales son capacidades o condiciones que el sistema debe cumplir.
- *RUP:* Proceso Unificado de Rational.
- *Subsistema*: Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos.
- *XML*: Lenguaje de marcas extensible.