Universidad de las Ciencias Informáticas
Facultad 3

Título: Sistema para la descarga y procesamiento automatizado de Patentes

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO INFORMÁTICO.

Autor: Luis Ernesto Saballo López.

Tutor: Msc. Rolando González Hernández
Lic. Rolan Robert Bullain Dieguez

Consultante: Ing. Rudel Cárdenas Díaz.

Caracas, Distrito Federal, Venezuela.
Mayo, 2007.

AGRADECIMIENTOS

A mis tutores Rolan y Rolando por guiarme en esta investigación, brindarme sus conocimientos, tiempo y más aún por ser buenos amigos.

A Marisela por ser tan buena persona y depositar tanta confianza en mi.

A Rudel por su asesoramiento, y su amistad.

A Yarina por su guía, apoyo, comprensión y su confianza depositada en todos nosotros.

A David, Ernesto, Oscar y Geykel por sus ideas y recomendaciones en esta investigación.

A Yanela por su amistad y ayuda en todo momento

A Irina y Michael por darme ánimo y ser tan comprensiva.

A Lizandra por su asesoramiento, apoyo, comprensión a toda hora.

A Nemury, Ameirys y Rolan por todo este tiempo cocinando y compartiendo juntos en Venezuela.

A mis compañeros del futbol por jugar en las tardes después del trabajo y poder liberar el estrés.

A los profesores por su dedicación y contribuir a nuestra formación profesional y revolucionaria.

A la Universidad de las Ciencias Informáticas, por educarnos y permitir hacer nuestros sueños realidad.

A mis compañeros de estudio por haber compartido estos 5 años de universidad.

A mis padres por la confianza, el apoyo y cariño que siempre me han dado, y por ser el mejor ejemplo para mí.

A mi hermano por toda la alegría que me trasmite.

A Yaniet por darme su amor y comprensión.

A mis abuelitos y tíos y primos, por todo el cariño que siempre me dan.

A Dania y Ramon por ser los mejores suegros del mundo.

A Pedro, Ivonne y Sayda por el apoyar el proyecto Delfos y su confianza en mí.

A todos los que preguntaron ¿Cómo va la tesis?

DEDICATORIA

A mis adorados padres por ser mis guías y ejemplo.

A mi novia por acompañarme en todos estos años.

A mis queridos familiares, en especial a mis abuelos y tíos que tanto me quieren.

A mis amistades.

A todas las personas que quiero y me quieren de verdad.

RESUMEN

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

En la actualidad, este proceso se realiza de forma manual, lo cual trae incorporado demora y errores, influyendo de forma negativa en la calidad y eficiencia del proceso.

En este trabajo se hace un estudio detallado de la situación antes planteada y se brinda una herramienta informática que automatiza el proceso de descarga y depuración del las páginas web, con la finalidad de obtener la información relevante de las patentes, definida por los usuarios y elevando exponencialmente: calidad, eficiencia, resultado y por consiguiente, se minimizan los errores. Los flujos de diseño e implementación del sistema son recogidos con el fin de explicar y documentar la investigación realizada.

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

INDICE DE TABLAS

INTRODUCCIÓN.

Muchos son los años que lleva la humanidad en su proceso de desarrollo, en el cual ha sido necesario organizar y establecer procesos formalizados sobre el estado de cada uno de los descubrimientos que se han venido realizando, tanto de base científico-tecnológico como otras categorías, así como la necesidad de tener registrado el autor del descubrimiento con el campo de acción del mismo, su fecha de creación entre otros campos. Producto de esta carencia de organización, control de datos oficiales y legales surgió la necesidad de certificar o emitir un derecho de autoría de todo aquello que constituya una innovación o quiera ser comercializado, es lo que conocemos hoy como patente[1](Hernández 2006).

El uso de patentes como indicador de innovación ha sido exhaustivamente estudiado y ha alcanzado un gran nivel de madurez en la actualidad debido a que las patentes son documentos que contienen información muy valiosa desde el punto de vista legal, técnico y comercial. Las patentes son de gran importancia, no solo para determinar indicadores de innovación de un país determinado, sino para establecer estrategias de negocios de las compañías tecnológicas y políticas de desarrollo en países menos desarrollado. Se considera que el 80% de la información que aparece en las patentes no aparece publicado en otras fuentes(Escorsa P 2001), por lo que constituyen una fuente ideal para la vigilancia tecnológica y la inteligencia competitiva.

Nuestro país en especial la Consultaría del Ministerio de informática, con su nombre abreviado Delfos, Centro Coordinador del Sistema de Información del Ministerio de la Informática y las Comunicaciones de Cuba, se encarga de ejercer la vigilancia tecnológica permanente en diferentes temáticas de interés para el desarrollo de las tecnologías de la informática y las comunicaciones (TIC). Actualmente el entorno en el que un trabajador de Delfos desarrolla su trabajo, puede incluir tanto el acceso a Bases de datos de patentes (BDP), como el acceso a otras bases de datos (BD) bibliográficas, normalmente de tipo científico–tecnológico que se suelen complementar entre ellas.

[1] Una patente es un conjunto de derechos exclusivos garantizados por un gobierno o autoridad al inventor de un nuevo producto (material o inmaterial) susceptible de ser explotado industrialmente para el bien del solicitante de dicha invención (como representante por ejemplo) por un espacio limitado de tiempo (generalmente 20 años desde la fecha de aprobación).

Por otro lado, estas BD pueden estar situadas tanto en un puesto de la red local (por ejemplo en una base de datos propia o en una base de datos comercial en CD), como en un sitio Web.

En el caso de las BDP estén ubicadas en Internet, el investigador se conecta al sitio Web correspondiente y descargan las páginas Web de cada uno de los documentos de Patentes que le interese, si lo que se quiere es hacer un estudio de tendencia o de mercado o cualquier otro tipo de análisis, se necesitarían descargar grandes volúmenes de páginas Web de Internet sobre patentes, y si esto no esta respaldado por un mecanismo que realice el trabajo de manera automática. El análisis es casi imposible o se necesitarían muchas horas y recursos humanos para hacer esta tarea de manera manual. Es por esto que se propone un sistema que posea entre otras funcionalidades, la de descarga de patentes de manera automática. Dándole como entrada el criterio de búsqueda y la BDP a la cual se desea el usuario conectar.

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

El proceso de descarga de patentes en esta consultoría no esta automatizado, y el proceso de filtrado para homogenizar la informacion y extraer datos relevantes es a través de macros[2] que se le aplican a cada una de las paginas Web descargadas. Esto trae consigo demoras y en ocasiones, se producen incumplimientos en la realización de dichas tareas. Por este motivo, DELFOS propuso a la Universidad de las Ciencias Informáticas, específicamente a la Facultad 3, el desarrollo de un sistema

[2] Macros: Conjunto de comandos que sirven para automatizar tareas que se ejecutan de forma repetitiva, y que a pesar de ser fáciles de realizar, pueden llegar a ser tediosas o llevarnos mucho tiempo; casi todos los programas ofimáticos incluyen módulos que permiten la creación y ejecución de macros.

informático de descarga y procesamiento de grandes volúmenes de información considerando las patentes como fuente fundamental.

Dada la situación antes descrita se platea la siguiente situación problémica: Actualmente en el ministerio de Informática y las Comunicaciones de Cuba, los especialistas hacen manualmente la de descarga patentes publicadas en bases de datos de Internet, lo cual es un proceso lento, engorroso, susceptible a error. Una vez que la información ha sido descargada, se le aplican diferentes tipos de filtros, también de una forma manual, para depurar la información y extraer datos bibliográficos de las patentes.

Lo cual permite plantear el siguiente problema: La ausencia de una herramienta que permita la descarga y procesamiento automático de patentes desde distintas bases de datos ubicadas en Internet, provoca demora e incumplimiento en el trabajo de los especialistas del Ministerio de Informática.

El problema planteado se enmarca en el objeto de estudio: La gestión de información disponible en los documentos de patentes publicados en Bases de Datos de Internet.

Para resolver el problema planteado se define, como objetivo general de este trabajo realizar el modelos de diseño, implementación y desarrollar una aplicación que permita de una forma rápida y sencilla la descarga y procesamiento de patentes desde distintas bases de datos ubicadas en Internet.

Cuyos objetivos específicos son:

- Realizar el Modelo Diseño del sistema.
- Realizar el Modelo de implementación.
- Implementar el modulo de configuración del sistema
- Implementar el módulo de descarga de información de Internet.
- Implementar un módulo de reporte para mostrar el resultado de las operaciones de búsqueda.

- Desarrollar un sistema con una interfaz amigable para el proceso de descarga y filtrado de manera automática y rápida.

El campo de acción determinado por el objetivo planteado, para darle cumplimiento a este trabajo es el siguiente: los datos bibliográficos de las patentes.

Esta investigación está basada en la hipótesis de que el desarrollo de una aplicación que automatice el proceso de gestión de información de patentes publicadas en las BD disponibles en Internet, mejorará significativamente el proceso de búsqueda, hace más rápida la descarga y homogeneización, lo cual repercute favorablemente en la calidad y los tiempos de respuesta de los estudios planificados.

Para un correcto desempeño de esta investigación se definen las siguientes tareas:

- Realizar una investigación del estado del arte sobre la base de datos de patentes publicadas en Internet.
- Estudiar los conceptos de Ingeniería de Software necesarios para hacer un modelo de diseño que me permita hace la aplicación lo mas robusta posible.
- Realizar una investigación sobre las técnicas de descarga de páginas de Internet.

CAPÍTULO 1: Fundamentación teórica.

Conocer que patentes están vigentes en el mercado, es fundamental antes de desarrollar nuevos productos, con una considerable inversión que luego podría ser bloqueada. En concordancia con esto se ha desarrollado numerosas herramientas informáticas que permiten descargar y analizar patentes de manera automática de Internet. Es por eso que en este trabajo se propone una solución para la creación de una herramienta para la gestión de patentes, y específicamente en este epígrafe se abordaran temas de interés y conceptos necesarios para la compresión del sistema en los capítulos posteriores.

1.1 Patentes.

Según la Agenda de la Política Exterior de los Estados Unidos de América, una patente es una concesión legal emitida por un gobierno que permite al inventor excluir a otras personas de fabricar, utilizar o vender un invento, declarado como propio, durante el plazo de vigencia de la patente. El Acuerdo sobre los Aspectos de los Derechos de Propiedad Intelectual (ADPIC) estipula que el plazo de vigencia de las patentes solicitadas después del 7 de junio de 1995, es de 20 años a partir de la fecha de solicitud. Para recibir protección de patente, un invento debe mostrar de manera clara que se puede patentar (proceso, maquinaria, artículo manufacturado), que es original y novedoso, que no es obvio y que es útil(USIS 1998).

La utilización de la información sobre patentes se ha extendido a muchas actividades comerciales tácticas y estratégicas, a la investigación y a las actividades de formulación de políticas a nivel nacional o institucional. La información sobre patentes incluye no sólo el contenido de los documentos de patente publicados, sino también las informaciones bibliográficas, los certificados de inventor, los certificados y modelos de utilidad. Se trata de la recopilación de documentos más completa y mejor clasificada sobre tecnologías nuevas e innovadoras. Mediante el análisis realizado a patentes es posible obtener:

- Los principales competidores, colaboradores actuales y potenciales.
- Los movimientos de intereses de los citados competidores evaluando la mayor o menor importancia que conceden a una tecnología o a una línea de investigación y desarrollo.

- Las estrategias de patentes utilizadas por los participantes y las oportunidades y amenazas de las estrategias alrededor de las mismas.

- Descubrir nichos económicos, y estrategias de desarrollo.

- Hasta el presente, se han publicado unos 40 millones de patentes en el mundo, en todos los ámbitos tecnológicos posibles. Y cada año se publica aproximadamente un millón. Algunos de los parámetros que usualmente se pueden determinar para estudiar el estado de la tecnología basado en información de patentes son: la evolución de la cantidad de patentes por año, los países de origen y los de destino de la tecnología, los principales objetos de patentes, las relaciones entre estos parámetros, así como el ciclo de vida de la tecnología.

1.2 Búsqueda y Análisis de Patentes.

Los análisis de patentes, son estudios que se basan en la búsqueda, compilación y análisis de la información de patentes en un sector o temática y tiempo determinado. Estos estudios revisten notable importancia por el gran valor propio de la información de patentes desde el punto de vista comercial, legal y técnico.

Los estudios o análisis de patentes pueden ser utilizados ante la necesidad de: patentar una solución, caracterizar la actividad tecnológica de un sector específico, conocer con mayor exactitud el estado de la técnica para un producto dado o vigilar la actividad de posibles competidores en una línea de investigación, detectar empresas punteras y hacer un balance de su capacidad de innovación tecnológica, así como inferir cuáles son las tendencias y líneas de investigación de mayor desarrollo en un momento dado.

1.2.1 Herramientas de búsqueda, recuperación y análisis de patentes.

Sistemas como el propuesto en este trabajo han aparecido constantemente en Internet aunque con diferentes características y funcionalidades, entre los más destacados que utilizan las grandes compañías dedicadas a la Vigilancia Tecnológica y a la Inteligencia Competitiva- tenemos:

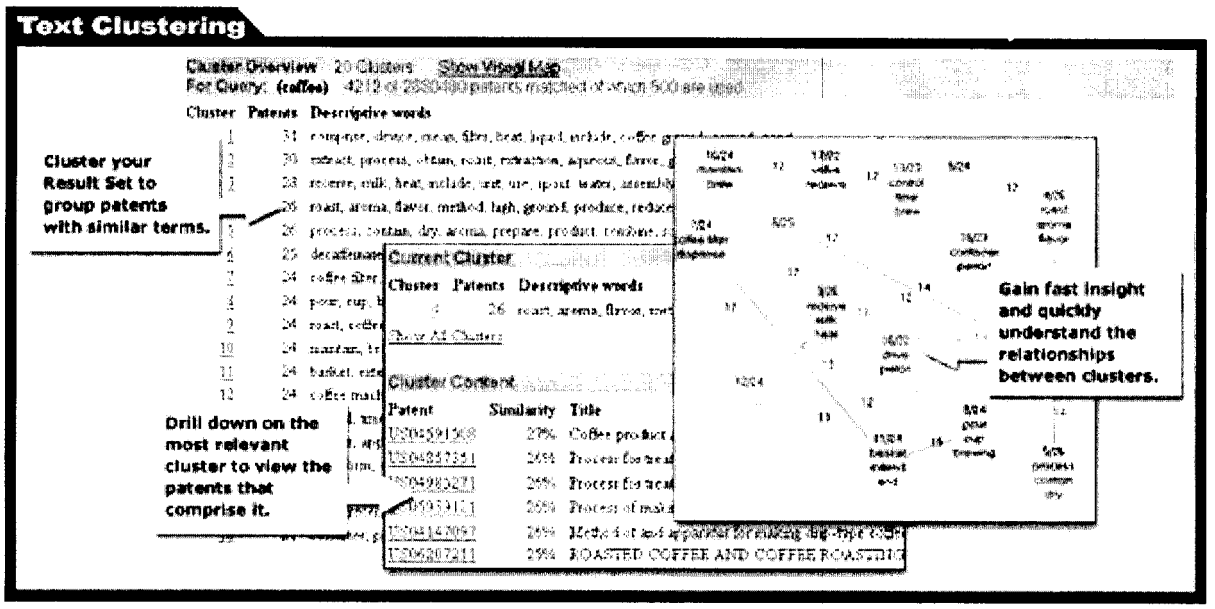
Matheo Patent: Es hasta el momento el más semejante al software de descarga que venimos desarrollando. Realiza la búsqueda, recuperación y análisis de patentes de las bases de datos de la Oficina de Patentes de los EE.UU. (USPTO) y de la Oficina Europea de Patentes (EPO). Tiene una interfaz amigable y posee diversas funcionalidades para la recuperación y análisis de patentes (Software 2006).

PatentHunter 3.0: Busca, descarga y realiza otras funciones como por ejemplo, mostrar el documento en formato pdf, HTML y permite imprimirlo. Su versión demo se encuentra disponible en (PatentWizard 2004).

BizInt Smart Charts: Carga la información obtenida de Bases de Datos de patentes como WPI, CA/CAplus, MicroPatent, Delphion y otras. Permite cargar esa información en una hoja electrónica, reformatearla y hacerle análisis estadísticos (Solutions 2006).

Aureka: En su versión ThemeScape los temas se representan visualmente en mapas con aspecto cartográfico, identificando los conceptos predominantes y sus relaciones; además se pueden comparar compañías, competidores o tecnologías. En su versión Citation Tree construye un árbol de citas a partir de una patente seleccionada (MicroPatent 2005).

Delphion-Text Clustering: Es una función en línea que extrae los términos más relevantes del texto de los resultados de una búsqueda, analiza sus relaciones y los presenta en una mapa(Corporation 2007), tomado del documento "Análisis y diseño de un proyecto de Gestión del Conocimiento en una pyme del sector textil" disponible en (Catalunya).



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

VantagePoint *VantagePoint 4.0:* Software para el análisis de información científico-técnica y de patentes desarrollado por Search Technologies. Permite la visualización de relaciones mediante matrices de co-ocurrencia, mapas tecnológicos y la creación de tesauros para la reducción de datos. Además, realiza análisis estadísticos multidimensionales para identificar grupos y relaciones entre conceptos, autores, países (Technology 2007).

Delfos CONSULTORIAS EMPRESARIALES *Predictor:* Nombre dado al sistema por la consultaría del ministerio de informática. Es un sistema que realiza búsquedas de patentes en BDP de Internet, procesa el contenido de dichas patentes y permite cargar esa información en una hoja de calculo de Excel, reformatearla y hacerle

análisis estadísticos. También genera un fichero para hacer análisis en Procite otro sistema de trabajo con patentes que utilizan en la consultaría.

No.	Software	Proveedor(País), Precio	Ultima versión, Utilidades
1	Matheo Patent	Matheo Software, (Francia), Primera licencia:€600/año; Segunda licencia:€480/año; 3-5 años: €420/año	v. 8.0, busca recupera y analiza patentes de USPTO[3] y Esp@cenet[4]
2		Patent Wizard, (EUA), Subscripción anual (1 licencia) USPTO (69 USD) y EPO (99 USD). Paquete profesional 5 licencias (349 USD)	v. 3.5, busca y descarga y maneja patentes y solicitudes de USPTO y EPO
3	PatentLab-II	Thomson(EUA) Gratuito, se cobra por cantidad de registros de descargas y conexión a Delphion (235 USD/mes)	Permite la visualización de indicadores relacionales
4	PM Manager	Wips Co.,(Corea del Sur), software gratuito supscripcion a WIPSGLOBAL por (200 USD/mes) ó (2000 USD/anual)	v. 4, busca, descarga y analiza información de USPTO, EPO. PCT, Japón y Corea del Sur

Tabla 1: Comparación de herramientas para la gestión de patentes.

1.3 Selección de herramientas para desarrollar el sistema.

Las herramientas son importantes en el desarrollo de cualquier proyecto, ya que facilitan la construcción de los mismos y la selección del conjunto de herramientas para desarrollarlos forma parte de todo proceso de desarrollo y debe hacerse cuidadosamente debido a que una inadecuada elección de las mismas podría dificultar el cumplimiento de los objetivos del negocio o dominio en cuestión que se hallan establecido, trayendo consigo perdida de tiempo y dinero, a la organización.

1.3.1 Tendencias de los lenguajes de programación de sistemas.

Las generaciones de los lenguajes de programación, ha seguido dos tendencias fundamentales:

[3] United State Patent and Trademark Office : Oficina de patentes y marcas de EUA
[4] 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

- El desplazamiento del centro de atención de la programación de pequeños sistemas hechos por una persona a los medianos y grandes sistemas, hechos por equipos de desarrollo de software(Booch 1996)

- La evolución de los lenguajes, un desplazamiento desde los lenguajes que dicen al computador qué hacer, o lenguajes imperativos, hacia lenguajes que describen las abstracciones clave en el dominio del problema (lenguajes declarativos)(Booch 1996).

Lenguajes de primera generación (1954-1958)(Booch 1996):

- FORTRAN 1 Expresiones matemáticas.
- ALGOL 58 Expresiones matemáticas.
- Flowmatic Expresiones matemáticas.
- IPLV Expresiones matemáticas.

Los lenguajes de la primera generación se utilizaron principalmente para aplicaciones científicas y de ingeniería, y su vocabulario fue matemático casi por completo. Así, los lenguajes como el FORTRAN 1 se desarrollaron para que el programador pudiera escribir fórmulas matemáticas, liberándole de esta forma de algunas de las complicaciones del lenguaje ensamblador o del lenguaje máquina. Esta primera generación de lenguajes de alto nivel representó por lo tanto un paso de acercamiento al espacio del problema, y un paso de alejamiento de la máquina que había debajo.

Lenguajes de segunda generación (1959-1961)(Booch 1996):

- FORTRAN 2 Subrutinas, compilación separada.
- ALGOL 60 Estructura en bloques, tipos de datos.
- COBOL Descripción de datos, manejo de ficheros.
- Lisp Procesamiento de listas, punteros, recolección de basura.

Los lenguajes de segunda generación, marcaron las abstracciones algorítmicas. Por esta época, las máquinas eran cada vez más y más potentes, y el abaratamiento en la industria de los computadores significó que podía automatizarse una mayor variedad de problemas, especialmente para aplicaciones comerciales. En este momento, lo principal era decirle a la máquina lo que debía hacer: lee primero estas fichas personales, ordénalas después, y a continuación imprime este informe. Una vez más, esta nueva generación de lenguajes de alto nivel acercaba a los desarrolladores aun más hacia el espacio del problema y los alejaba de la máquina subyacente.

Lenguajes de tercera generación (1962-1970)(Booch 1996):

- PL/ 1 FORTRAN + ALGOL + COBOL.
- ALGOL 68 Sucesor riguroso del ALGOL 60.
- Pascal Sucesor sencillo del ALGOL 60.
- Simula Clases, abstracción de datos.

Los lenguajes de tercera generación a finales de los sesenta, especialmente con la llegada de los transistores y la tecnología de circuitos integrados, el coste del hardware de los computadores había caído de forma dramática, pero su capacidad de procesamiento había crecido casi exponencialmente. Ahora podían resolverse problemas mayores, pero eso exigía la manipulación de más tipos de datos. Así, lenguajes como el ALGOL 60 y posteriormente el Pascal evolucionaron soportando abstracción de datos. El programador podía describir el significado de clases de datos relacionadas entre sí (su tipo) y permitir que el lenguaje de programación apoyase estas decisiones de diseño. Esta generación de lenguajes acercó de nuevo el software un paso hacia el dominio del problema, y lo alejó otro paso de la máquina.

El Hueco Generacional (1970- 1980):

Los setenta ofrecieron un frenesí de actividad investigadora en materia de lenguajes de programación, con el resultado de la creación de diferentes lenguajes cada uno con sus propios dialectos. "Se inventaron muchos lenguajes diferentes, pero pocos perduraron" (Peter 1981). En gran medida, la tendencia a escribir programas más y más grandes puso de manifiesto las deficiencias de

los lenguajes más antiguos; así, se desarrollaron muchos nuevos mecanismos lingüísticos para superar estas limitaciones. Sólo algunos de estos lenguajes han sobrevivido, mientras que otros como Fred, Chaos o Tranquil, se quedaron en el camino, y encontrar literatura de ellos se hace bastante difícil. Sin embargo, muchos de los conceptos que introdujeron encontraron su camino en otros sucesores. Así, hoy en día existen Smalltalk (un sucesor revolucionario de Simula), Ada (un sucesor del ALGOL 68 y Pascal, con contribuciones de Simula, Alphard y CLU), CLOS (que surgió del Lisp, LOOPS y Flavors), C++ (derivado de C y Simula), y Eiffel (derivado de Simula y Ada).

Algunos de los lenguajes mas populares en la actualidad:

C++: Es un lenguaje de programación (LP), diseñado a principios de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C, sus principales características son el soporte para programación orientada a objetos, puede manipular directamente la memoria de la computadora, no tienen muchas verificaciones automáticas, no da soporte para interfaces de implementación, tiene. Entornos de desarrollos (IDE) mas comunes que dan soporte a este lenguaje. Visual C++ .NET de Microsoft, GCC para software libre, Borland C++ Builder de Borland

Java: Es un LP orientado a objetos desarrollado por Sun Microsystems a principios de los 90. A diferencia de los lenguajes de programación convencionales, que generalmente están diseñados para ser compilados a código nativo, Java es compilado en un bytecode que es interpretado (usando normalmente un compilador JIT), por una máquina virtual Java. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos mucho más simple y elimina herramientas de bajo nivel como punteros, da soporte a interfaces de implementación, y posee un modelo de objetos mejor definido que C++.

C#: Es un LP dentro de la plataforma Microsoft .Net es un lenguaje de programación orientado a objetos, desarrollado y estandarizado por Microsoft como parte de su plataforma .NET. Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma .NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes Object Pascal más conocido por su IDE(Delphi). El 7 de noviembre de 2005 acabó la beta y salió la versión 2.0 del lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables,

interfaces de implementación. Ya existe la versión 3.0 de C# en fase de beta destacando los tipos implícitos y el LINQ (Language Integrated Query).

En la actualidad existen los siguientes compiladores para el lenguaje C#:

- Microsoft .NET FrameWork SDK incluye un compilador de C#, pero no un IDE.
- Microsoft Visual C#, IDE por excelencia de este lenguaje, versión 2002, 2003 y 2005.
- #develop, es un IDE libre para C# bajo licencia LGPL, muy similar a Microsoft Visual C#.
- Mono, es una implementación GPL de todo el entorno .NET desarrollado por Novell. Como parte de esta implementación se incluye un compilador de C#.
- Delphi 2006, de Borland Software Corporation.
- dotGNU Portable.NET, de la Free Software Foundation

1.3.2 Entornos de desarrollo.

Un entorno de desarrollo integrado o en inglés Integrated Development Environment (IDE) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE es un entorno de programación que ha sido empaquetado como un programa de aplicación, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI[5]. Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes. El leguaje Visual Basic por ejemplo puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Word.

Los IDEs proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Visual Basic, Object Pascal, entre otros. Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, que

[5] GUI: Graphics User Interface

mediante pluggins[6] se le puede añadir soporte de lenguajes adicionales, o el Microsoft Visual Studio.NET, que es unos de los más modernos y sofisticados.

1.3.3 Estructuras de Datos, Tipos de Datos Abstractos y Algoritmos.

Para procesar información en un computador es necesario hacer una abstracción de los datos que tomamos del mundo real, abstraernos en el sentido de que se ignoren algunas propiedades de los objetos reales, o se simplifiquen, y a través de los algoritmos dar solución a los problemas.

1.3.3.1 Estructuras de datos.

Una estructura de datos es un conjunto de variables de un determinado tipo agrupadas y organizadas de alguna manera para representar un comportamiento. Lo que se pretende con las estructuras de datos es facilitar un esquema lógico para manipular los datos en función del problema que haya que tratar y el algoritmo para resolverlo. En algunos casos la dificultad para resolver un problema radica en escoger la estructura de datos adecuada.

1.3.3.2 Tipos de Datos Abstractos.

Los tipos abstractos de datos (TAD) permiten describir una estructura de datos en función de las operaciones que pueden efectuar, dejando a un lado su implementación.

Los TAD mezclan estructuras de datos junto a una serie de operaciones de manipulación. Incluyen una especificación, que es lo que verá el usuario, y una implementación (algoritmos de operaciones sobre las estructuras de datos y su representación en un lenguaje de programación), que el usuario no tiene necesariamente que conocer para manipular correctamente los tipos abstractos de datos.

Se caracterizan por el encapsulamiento, esconde su interior, y proporciona una interfaz de comunicación que le permite integrarla al resto de los componentes de un sistema. Esto permite aumentar la complejidad de los programas, pero manteniendo claridad suficiente para no desbordar a los desarrolladores.

[6] Pluggins: Modulo de programa que añade características y funcionalidades nuevas a un sistema.

1.3.3.3 Algoritmos.

Un algoritmo es un conjunto ordenado y finito de operaciones que permite hallar la solución de un problema, también es un método y notación en las distintas formas de cálculo(Española 1992).

Un algoritmo en términos de informática es un procedimiento computacional bien definido que toma como entrada un valor, o conjunto de valores, y produce como salida otro valor, o conjunto de valores(Cormen 2002).

La importancia del conocimiento de los algoritmos viene dado por el uso apropiado de los mismos, para conocer a que velocidad funcionara nuestro software, es necesario entender los detalles de los mismos para poder predecir si habrá casos en los que el software no funcionara de manera rápida o si se producirán resultados inadecuados.

1.3.4 Metodologías de desarrollo de Software.

Todo desarrollo de software es riesgoso y difícil de controlar, pero si no llevamos una metodología de por medio, lo que obtenemos es clientes insatisfechos con el resultado y desarrolladores aún más insatisfechos. Sin embargo, muchas veces no se toma en cuenta el utilizar una metodología adecuada, sobre todo cuando se trata de proyectos pequeños de dos o tres meses. Lo que se hace con este tipo de proyectos es separar rápidamente el aplicativo en procesos, cada proceso en funciones, y por cada función determinar un tiempo aproximado de desarrollo (Pressman 2002).

Cuando los proyectos que se van a desarrollar son de mayor envergadura, ahí si toma sentido el basarnos en una metodología de desarrollo, y empezamos a buscar cual sería la más apropiada para nuestro caso. Lo cierto es que muchas veces no encontramos la más adecuada y terminamos por hacer o diseñar nuestra propia metodología, algo que por supuesto no esta mal, siempre y cuando cumpla con el objetivo(Ivar Jacobson 2000). Muchas veces realizamos el diseño de nuestro software de manera rígida, con los requerimientos que el cliente nos solicitó, de tal manera que cuando el cliente en la etapa final (etapa de prueba), solicita un cambio se nos hace muy difícil realizarlo, pues si lo hacemos, altera muchas cosas que no habíamos previsto, y es justo éste, uno de los factores que ocasiona un atraso en el proyecto y por tanto la incomodidad del desarrollador por no cumplir con

el cambio solicitado y el malestar por parte del cliente por no tomar en cuenta su pedido. Obviamente para evitar estos incidentes debemos haber llegado a un acuerdo formal con el cliente, al inicio del proyecto, de tal manera que cada cambio o modificación no perjudique al desarrollo del mismo.

Por experiencia, muchas veces los usuarios finales, se dan cuenta de las cosas que dejaron de mencionar, recién en la etapa final del proyecto, pese a que se les mostró un prototipo del software en la etapa inicial del proyecto. Los proyectos en problemas son los que salen del presupuesto, tienen importantes retrasos, o simplemente no cumplen con las expectativas del cliente.

Para dar una idea de qué metodología podemos utilizar y cual se adapta más a nuestro medio, mencionaré tres de ellas de las que considero las más importantes, tal como: RUP, XP y MSF.

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

- Inicio, El Objetivo en esta etapa es determinar la visión del proyecto.
- Elaboración, En esta etapa el objetivo es determinar la arquitectura óptima.
- Construcción, En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial.
- Transición, El objetivo es llegar a obtener el release del proyecto.

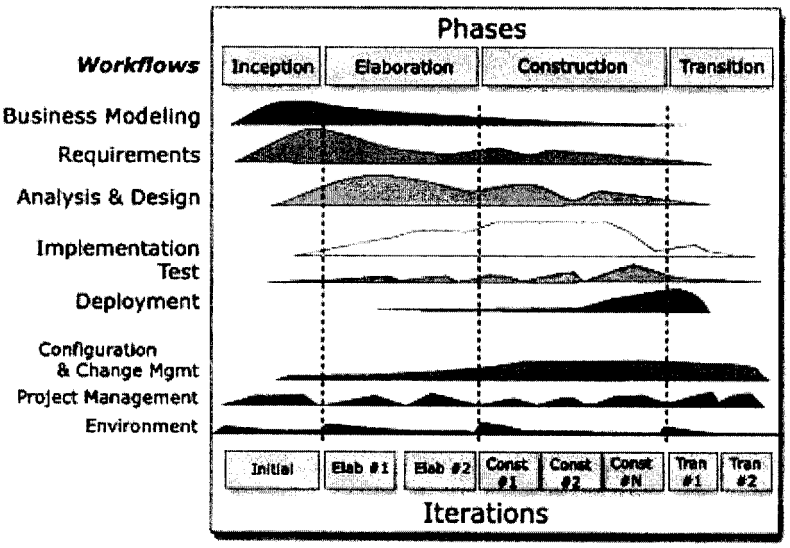


Figura 1: Fases e Iteraciones de RUP.

Cada una de estas etapas es desarrollada mediante el ciclo de iteraciones, la cual consiste en reproducir el ciclo de vida en cascada a menor escala. Los Objetivos de una iteración se establecen en función de la evaluación de las iteraciones precedentes. Vale mencionar que el ciclo de vida que se desarrolla por cada iteración, es llevada bajo dos disciplinas:

Disciplina de Desarrollo

- Ingeniería de Negocios: Entendiendo las necesidades del negocio.
- Requerimientos: Trasladando las necesidades del negocio a un sistema automatizado.
- Análisis y Diseño: Trasladando los requerimientos dentro de la arquitectura de software.
- Implementación: Creando software que se ajuste a la arquitectura y que tenga el comportamiento deseado.
- Pruebas: Asegurándose que el comportamiento requerido es el correcto y que todo los solicitado esta presente.

Disciplina de Soporte

- Configuración y administración del cambio: Guardando todas las versiones del proyecto.
- Administrando el proyecto: Administrando horarios y recursos.
- Ambiente: Administrando el ambiente de desarrollo.
- Distribución: Hacer todo lo necesario para la salida del proyecto

Es recomendable que a cada una de estas iteraciones se les clasifique y ordene según su prioridad, y que cada una se convierte luego en un entregable al cliente. Esto trae como beneficio la retroalimentación que se tendría en cada entregable o en cada iteración.

Los elementos del RUP son:

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

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

Extreme Programing (XP)

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

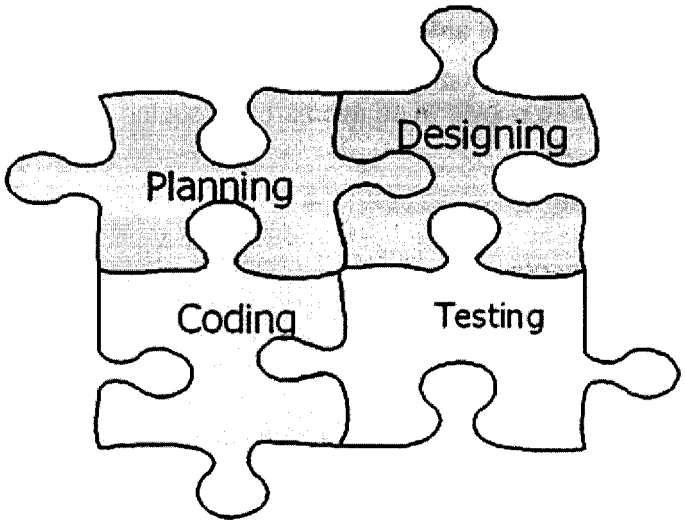


Figura 2: Metodología Extreme Programing.

Características de XP, la metodología se basa en:

Pruebas Unitarias: se basa en las pruebas realizadas a los principales procesos, de tal manera que adelantándonos en algo hacia el futuro, podamos hacer pruebas de las fallas que pudieran ocurrir. Es como si nos adelantáramos a obtener los posibles errores.

Refabricación: se basa en la reutilización de código, para lo cual se crean patrones o modelos estándares, siendo más flexible al cambio.

Programación en pares: una particularidad de esta metodología es que propone la programación en pares, la cual consiste en que dos desarrolladores participen en un proyecto en una misma estación de trabajo. Cada miembro lleva a cabo la acción que el otro no está haciendo en ese momento. Es como el chofer y el copiloto: mientras uno conduce, el otro consulta el mapa.

¿Qué es lo que propone XP?

- Empieza en pequeño y añade funcionalidad con retroalimentación continua
- El manejo del cambio se convierte en parte sustantiva del proceso
- El costo del cambio no depende de la fase o etapa

- No introduce funcionalidades antes que sean necesarias
- El cliente o el usuario se convierte en miembro del equipo

Derechos del Cliente

- Decidir que se implementa
- Saber el estado real y el progreso del proyecto
- Añadir, cambiar o quitar requerimientos en cualquier momento
- Obtener lo máximo de cada semana de trabajo
- Obtener un sistema funcionando cada 3 o 4 meses

Derechos del Desarrollador

- Decidir como se implementan los procesos
- Crear el sistema con la mejor calidad posible
- Pedir al cliente en cualquier momento aclaraciones de los requerimientos
- Estimar el esfuerzo para implementar el sistema
- Cambiar los requerimientos en base a nuevos descubrimientos
- Lo fundamental en este tipo de metodología es:
- La comunicación, entre los usuarios y los desarrolladores
- La simplicidad, al desarrollar y codificar los módulos del sistema
- La retroalimentación, concreta y frecuente del equipo de desarrollo, el cliente y los usuarios finales

Microsoft Solution Framework (MSF)

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

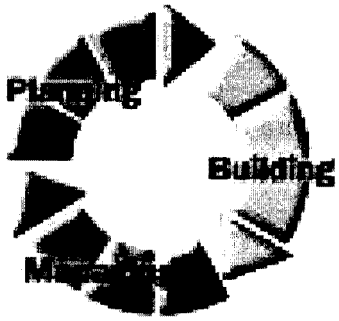


Figura 3: Metodología MSF.

MSF tiene las siguientes características:

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

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el modelo de Aplicación.

- Modelo de Arquitectura del Proyecto: Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.

- Modelo de Equipo: Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.

- Modelo de Proceso: Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.

- Modelo de Gestión del Riesgo: Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.

- Modelo de Diseño del Proceso: Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.

- Modelo de Aplicación: Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

Con este análisis hecho anteriormente sobre metodologías y procesos de desarrollo de software podemos decir que:

- La Metodología RUP es más adaptable para proyectos de largo plazo.
- La Metodología XP en cambio, se recomienda para proyectos de corto plazo.
- La Metodología MSF se adapta a proyectos de cualquier dimensión y de cualquier tecnología.

Para el desarrollo de este sistema se escogió RUP y la notación UML debido a el alcance que nosotros esperamos que tenga este proyecto y la documentación que necesitamos de cada uno de sus artefactos bien, los cuales debe estar bien definidos y documentado para facilitar el posterior desarrollo o a ampliación del sistema con nuevos requisitos en determinados intervalos de tiempo. También jugó un papel fundamental el conocimiento de los integrantes del equipo de desarrollo y demás colaboradores cercanos, sobre esta metodología ahorrando consigo tiempo de estudio e investigación. Por otro lado podemos también aclarar que aunque el proyecto no es grande, su desarrollo no es a corto plazo pues a este se le incorporan constante mente nuevos requisitos funcionales pedidos por el cliente, lo cual mantiene activo el proyecto en un periodo de tiempo considerable de desarrollo, aunque liberando versiones listas para usar, en sus distintas iteraciones.

1.3.4.1 Lenguaje de Unificado y Modelado (UML).

UML es, probablemente, una de las innovaciones conceptuales en el mundo tecnológico del desarrollo de software que más expectativas y entusiasmos haya generado en muchos años (Rumbaugh. 2000). Es un estándar en la industria del software, creado por Grady Booch, James Rumbaugh y Ivar Jacobson.

¿Que es un modelo?

Un modelo en una aplicación informática es una abstracción del software, que especifica desde cierto punto de vista y con determinado nivel de abstracción, detalles del diseño de un sistema (Ivar Jacobson 2000).

Un modelo es una representación de algo en cierto medio. El modelo capta los aspectos importantes de lo que estamos modelando, desde cierto punto de vista, y simplifica u omite el resto. La ingeniería, la arquitectura y muchos otros campos creativos usan modelos (Rumbaugh. 2000).

Un modelo de un sistema software está construido en un lenguaje de modelado, como UML. El modelo tiene semántica y notación y puede adoptar varios formatos que incluyen texto y gráficos. El modelo pretende ser más fácil de usar para ciertos propósitos que el sistema final (Rumbaugh. 2000).

¿Porque Sirven los modelos?

- Para captar y enumerar exhaustivamente los requisitos y el dominio de conocimiento, de forma que todos los implicados puedan entenderlos y estar de acuerdo con ellos.

Ejemplo: Los diversos modelos de un sistema de software pueden capturar requisitos sobre su dominio de aplicación, las formas en que los usuarios lo utilizarán, su división en módulos, los patrones comunes usados en su construcción, etc.

- Para pensar en el diseño del sistema.

Ejemplo: Un diseñador o Arquitecto de software utiliza modelos para visualizar y experimentar con posibles diseños. La simplicidad de crear y modificar modelos pequeños permite un pensamiento creativo e innovación con poco coste. Un modelo de diseño de software ayuda a los desarrolladores a explorar varias arquitecturas y soluciones de diseño fácilmente, antes de escribir el código.

- Para generar productos aprovechables para el trabajo.

Ejemplo: Para generar documentos, declaraciones de escenarios de usos validos, guiones de configuración, glosarios de términos.

1.3.4.2 Patrones de Diseño.

Un patrón de diseño[7] no es más que una solución a un problema de diseño no trivial que es efectiva, en otras palabras, que ya se resolvió el problema satisfactoriamente en ocasiones anteriores y es reusable para que se pueda aplicar a diferentes problemas similares de diseño en distintas circunstancias(Erich Gamma 1994). Los patrones de diseño son de gran importancia ya que nos hablan de como construir software, de como utilizar las clases y los objetos de forma conocida, con el

[7] Los Patrones de Diseño (*Design Patterns*) son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

uso de ellos podemos obtener ventajas tales como tener una forma reutilizar la experiencia de otros desarrolladores. Es una experiencia real, probada y que funciona. Es Historia y nos ayuda a no cometer los mismos errores que ya analizados y probados por otros. Si se hace una caracterización de ellos se puede decir que:

- 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. Ya que se basan en la experiencia acumulada la 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.

1.4 Arquitectura de Software.

Un sistema informático requiere una arquitectura para que los desarrolladores puedan progresar hasta tener una visión común. Un software es difícil de abarcar visualmente porque no existe un modelo de tres dimensiones. Es a menudo único y sin precedentes en determinados aspectos y suele utilizar mezcla de tecnologías existentes. Además debe ser construido para acomodar gran cantidad de clases que sufrirán cambios futuros. A medida que los sistemas se hacen más complejos, "los problemas de diseño van mas allá de los algoritmos y las estructuras de datos para su computación" (Garlan 1996)

¿Qué es una arquitectura?

Es lo que especifica el arquitecto en la descripción de la arquitectura. La descripción de la arquitectura permite al arquitecto controlar el desarrollo del sistema desde la perspectiva técnica (Ivar Jacobson 1997). La arquitectura del software se centra tanto en los elementos estructurales significativos del sistema, como Subsistemas, clase, componentes y nodos, como en las colaboraciones que tienen lugar entre estos elementos a través de las interfaces (Ivar Jacobson 2000).

Los casos de uso dirigen la arquitectura para hacer que el sistema proporcione la funcionalidad y uso deseado. Una arquitectura debe ser completa, pero también debe ser suficientemente flexible como para incorporar nuevas funciones y debe soportar la reutilización del software existente.

¿Cómo se obtiene?

La arquitectura se desarrolla de forma iterativa durante la fase de elaboración, pasando por los requisitos, el análisis, el diseño, la implementación y las pruebas. Utilizamos los casos de usos significativos para la arquitectura y un conjunto de otras entradas para implementar la línea base de la misma, este conjunto de entrada incluye requisitos funcionales y no funcionales del software, entre otros.

¿Cómo se describe?

La descripción de la arquitectura es una vista de los modelos del sistema, los casos de uso, análisis, diseño, implementación y despliegue. La descripción de la arquitectura describe las partes del sistema que es importante que comprendan todos los desarrolladores y otros interesados.

1.5 Solicitudes Web mediante programación (Descargas de Internet automáticas).

Comenzaremos por mencionar brevemente algunos términos muy usados en el tema de solicitudes web que usaremos a lo largo de este trabajo, uno de ellos es:

Http[8], este es un protocolo a nivel de aplicación, sobre el cual se realizan transacciones en la Web, actualmente esta en su versión 1.1 la cual es un estándar para los desarrolladores de aplicaciones, y es mas estable que las versiones anteriores (W3C).

HTML[9], este es un lenguaje de marcas o etiquetas HTML (tags) y su principal uso es describir, y mostrar los textos en la Web

Proxy[10], en el contesto que lo utilizamos es un programa o dispositivo que realiza una tarea acceso a Internet en lugar de otro ordenador. Un proxy es un punto intermedio entre un ordenador conectado a Internet y el servidor que está accediendo. Cuando navegamos a través de un proxy en realidad no estamos accediendo directamente al servidor, sino que realizamos una solicitud al proxy y éste es quien se conecta con el servidor que queremos acceder, devuelve el resultado de la solicitud realizada.

Cuando nos conectamos con un proxy, el servidor al que accedemos en realidad recibe la solicitud del proxy, en vez de recibirla directamente desde nuestro ordenador. Puede haber sistemas proxy que interceptan diversos servicios de Internet. Lo más habitual es el proxy web, que sirve para interceptar las conexiones con la Web y puede ser útil para incrementar la seguridad, rapidez de navegación o anonimato.

Existen varias formas para solicitar una página Web mediante programación, pero la gran generalidad tiene en común que utilizan las potencialidades del protocolo http, sus diferencias radican mayormente en los componentes o clases que se dispongan en la plataforma de trabajo en la que se este desarrollando. Por ejemplo, antes de existir la plataforma .NET Los programadores de Microsoft Visual 6.0 resolvían el problema de las solicitudes Web a través de las interfaces de programación de aplicaciones (APIs) de WinInet (Microsoft 2003).

Haciendo uso del .NET framework, se puede utilizar el espacio de nombre (namespace[11]) **System.Net** que proporcionan la clase **WebRequest** para encapsular una solicitud de un recurso de

[8] Http es un acrónimo de Hyper Text Transfer Protocol
[9] Html es un acrónimo de Hyper Text Marckup Languague
[10] Proxy es un programa intermediario que actúa a la vez como servidor y cliente para realizar demandas de otros clientes.
[11] Namespace: La palabra clave **namespace** se utiliza para declarar un ámbito. Este ámbito permite organizar el código y proporciona una forma de crear tipos globalmente únicosMicrosoft(MSDN). from http://msdn2.microsoft.com/es-es/library/z2kcy19k(VS.80).aspx.

Internet, y la clase **WebResponse** para representar los datos devueltos. Con estos se puede obtener una secuencia que representa la respuesta a una solicitud determinada. Una vez obtenida la secuencia, la lectura de la respuesta es igual que leer un archivo de texto local o con otro origen.

En otras plataformas o lenguaje que aún no dispongan de estos componentes mencionados, como PHP, el programador tendría que implementar u obtener de algún lugar su propia clase HttpRequest, muy semejante en principio a las que fueron creadas por Microsoft en el .NET FrameWork, cuyo contenido fundamentalmente sea:

- Dirección IP del usuario: la dirección IP que indica la clase es la real del usuario, aunque éste acceda a través de algún proxy.
- Información GET, POST y cookies:

1.5.1 Programación Multi-hilo.

Las solicitudes web mediante programación pueden tratarse de muchas maneras, pero en todas es recomendable hacer uso de programación multi-hilo, para permitir que el usuario pueda hacer otras cosas en la aplicación, mientras esta el proceso de descarga en otro hilo ejecutándose.

Un hilo o thread como se dice en Ingles y abordado en literaturas incluso de habla hispana, no es más que un flujo de control secuencial dentro de un programa(Kay Septiembre 2003). Un thread es un flujo de control que puede tener sus propias estructuras de datos y puede compartir los mismos datos con otros hilos o la aplicación que lo creo.

Un hilo no puede correr por sí mismo, se ejecuta dentro de un programa. Se pueden programar múltiples hilos de ejecución para que corran simultáneamente en el mismo programa. La utilidad de la programación multihilos resulta evidente. Por ejemplo, un navegador Web puede descargar un archivo de un sitio, y acceder a otro sitio al mismo tiempo. Si el navegador puede realizar simultáneamente dos tareas, no tendrá que esperar hasta que el archivo haya terminado de descargarse para poder navegar a otro sitio.

En determinadas ocasiones, cuando desarrollamos aplicaciones Windows hemos necesitado hacer algún proceso en paralelo al proceso que se está ejecutando en ese momento. Pero para ser

mas certeros, en un computador con un sólo procesador (CPU), no se pueden ejecutar dos aplicaciones al mismo tiempo, ya que el procesador sólo puede realizar el trabajo de una a la vez. Entonces, ¿Como es el multiproceso en un computador con un procesador? La respuesta es muy simple. El procesador ejecuta cada aplicación un tiempo muy breve, pero muchas veces por segundo, entonces nos da la impresión de que se están ejecutando todas las aplicaciones al mismo tiempo. Ese tiempo que se ejecuta una aplicación se conoce como tiempo del procesador (o rebanada de tiempo - TimeSlice), y con ejecutar la aplicación nos referimos al hecho de ejecutar el hilo en que corre esa aplicación.

Si tenemos un computador con más de un procesador, y nuestra aplicación utiliza mas de un hilo, podría ocurrir que nuestros hilos si se ejecuten en forma simultánea, pero eso dependerá de los ciclos del procesador, de la cantidad de hilos que se estén ejecutando en ese momento y de la prioridad de cada uno, entre otras opciones. Al tener mas procesadores, si la utilización de hilos es correcta, deberán notarse resultados en la mejora del rendimiento de nuestra aplicación. Entonces, si tenemos un computador con un sólo procesador, ¿que ganamos utilizando hilos, si al final el procesador se tendrá que repartir entre mas procesos? Para esta pregunta, la respuesta debe enfocarse desde los puntos de vista de Rendimiento y Sensación. Con respecto al rendimiento, como esperamos, no hay una mejora perceptible ya que el procesador debe repartirse en más procesos (como veremos mas adelante en la aplicación). Sin embargo, la sensación del usuario que utiliza la aplicación mejora. El podría seguir haciendo otras cosas mientras la aplicación ejecuta en otro hilo un requerimiento largo y costoso.

CAPÍTULO 2: Características del sistema.

A través de este capítulo se describe el objeto de estudio, y como darle solución a los principales problemas existentes en la consultaría informática del MIC relacionadas con la descarga de patentes de Internet y el procesamiento de las mismas, se realiza la propuesta y caracterización del sistema basándose en los conceptos mencionados en el capitulo anterior y el análisis del sistema desarrollado por (Piñeiro 2007), donde se han plasmado con suficiente grado de detalle todos los aspectos necesarios para entrar en el tema que vamos a ver a continuación y poder realizar los flujos siguientes de diseño e implementación.

2.1 Arquitectura.

La arquitectura del sistema tiene gran importancia en la construcción del mismo debido a que describe los elementos arquitectónicos que son más importantes en el sistema, estos elementos son: subsistemas, dependencias, interfaces, colaboraciones, entre otros. La importancia de estos elementos, reside en el hecho de que guían el trabajo en el sistema a través del ciclo de vida del proyecto, todos ellos describen los cimientos del sistema, que son necesarios para comprenderlo, desarrollarlo, y producirlo económicamente (Ivar Jacobson 2000).

La arquitectura de software abarca decisiones importantes sobre:

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

2.1.1 Patrón de arquitectura utilizado en el sistema.

Para el desarrollo de este sistema se definió la arquitectura tres capas por las ventajas que esta ofrece:

- Proporciona una escalabilidad, capacidad de administración y utilización de recursos mejorados.
- Cada capa es un grupo de componentes que realiza una función específica.
- Se puede actualizar una capa sin recompilar otras capas.

El contenido de cada una de las capas mencionadas en correspondencia con la solución del sistema esta dividido de la siguiente manera:

- La capa de presentación (Figura 4 epígrafe 2.1.1), que en este caso esta formada por los Componentes de IU[12], los cuales pueden ser vistos como la parte con la cual interactúa el usuario. Ya sean ventanas que contenga controles de usuarios o cualquier otro tipo de control que tenga una interfaz grafica. Los componentes de proceso de IU podríamos asociarlos a clases de tipo Interfaz en UML. Es decir estos encapsulan lógica de navegación y control de eventos de la interface.
- La capa de negocios (Figura 4 epígrafe 2.1.1), encapsula lógica, descripción de los procesos, estructuras o representaciones que controlen el funcionamiento del negocio, las entidades empresariales que representan objetos que fueron modelados anteriormente y que van a ser manejados o consumidos por toda la aplicación.
- La capa de acceso a datos (Figura 4 epígrafe 2.1.1), contiene clases que interactúan con la base de datos y permiten, utilizando los procedimientos almacenados, realizar todas las operaciones con la base de datos de forma transparente para la capa de negocio, permitiendo la migración hacia otro Sistema Gestor de Bases de Datos sin muchas complicaciones.

[12] IU: Interfaz de Usuario, Interfaz con la que interactúa los usuarios, generalmente esta compuesta por formularios, controles, entre otros, como veremos mas adelante en cap3 en el diseño e implementación del sistema

La capa de acceso a datos esta concebida en el sistema, pero no en el marco de esta investigación, así como la creación de una Base de Datos centralizada con los datos de las patentes que se han descargado de internet con el objetivo de obtener reportes mas personalizados y rápidos, así como preguntar al usuario si desea consultar la Base de Datos central antes de comenzar a descargar patentes de internet, estas ideas estarán presente en las recomendaciones para darle seguimiento al sistema y seguir ampliando sus capacidades.

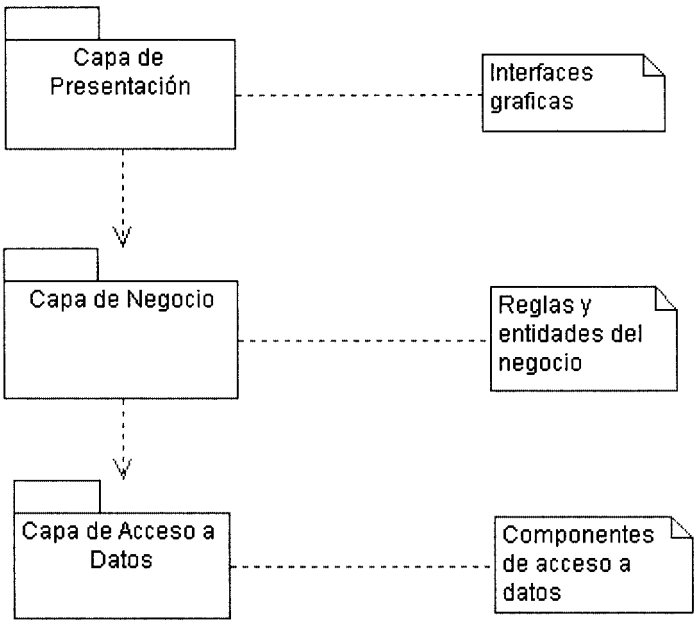


Figura 4: Arquitectura 3 capas, definas en el sistema.

2.1.2 Organización del desarrollo centrado en la arquitectura.

La organización del desarrollo de un sistema centrado en la arquitectura es una de las características fundamentales de RUP, esto se debe a que los casos de usos véase (Piñeiro 2007) solamente no son suficientes para conseguir un sistema de trabajo (Ivar Jacobson 2000).

En la capa de presentación internamente se estableció una arquitectura que permitiera la actualización del sistema una vez implantado en el entorno del cliente véase Cap. 2 epígrafe 2.5, este

aspecto de la arquitectura facilitará las tareas de soporte e integración de cada uno de los subsistemas que mencionaremos a continuación.

El sistema esta dividido en cuatro partes o subsistemas fundamentales:

1. Configuración.
2. Descarga o solicitudes Web
3. Parseo o procesamiento del HTML
4. Reportes.

El subsistema de parseo es tratado en (Castellanos. 2007). En esta investigación nos concentraremos en los otros 3 subsistemas, de manera que se pueda tener una perspectiva ó visión del sistema completo, necesaria para controlar su desarrollo.

Subsistema de configuración: Es el encargado de controlar la configuración de los distintos aspectos del sistema:

- Configuración de la conexión a Internet.
- Configuración de las Bases de datos a las que se conectará el sistema.
- Configuración de la apariencia y colores de la aplicación.
- Configuración de los datos de los usuarios del sistema.

Subsistema de descarga: Es el encargado de hacer las solicitudes Web, llevando a cabo todo el proceso de:

- Creación de hilos de ejecución de las descargas y gestión de los mismos cambiando su estado a pausa, ejecución ó cancelación según desee el usuario.

- Construcción de direcciones URL[13] a partir de parámetros de búsqueda y filtros que se deseen utilizar.

- Almacenamiento de la información descargada de Internet en la carpeta de trabajo seleccionada por el usuario.

Subsistema de reportes: Es el encargado de mostrar los reportes de proyectos de descargas anteriores, a partir de:

- La selección del proyecto de interés del usuario, mostrando los datos obtenidos de cada patente, como resultado de procesamiento del HTML de las paginas descargadas, este proceso de minería es llevado a cabo por el subsistema de procesamiento o parseo(Castellanos. 2007).

- La selección de un criterio de búsqueda por el cual se hallan descargado patentes anteriormente.

2.2 Entorno de desarrollo y Lenguaje de Programación Seleccionado.

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

- No existió un requerimiento por parte del cliente sobre la definición de una plataforma específica.

- La necesidad de una plataforma de trabajo productiva con una buena Biblioteca de clases, que incluyera componentes para el trabajo con hilos y componentes de conexión a internet, así como determinada estructuras de datos básicos para la construcción del sistema.

- La necesidad de un lenguaje dentro de la plataforma que soportara los conceptos de POO necesarios, para el desarrollo de la misma.

[13] Significa ***U**niform **R**esource **L**ocator*, es decir, localizador uniforme de recurso. Es una secuencia de caracteres, de acuerdo a un formato estándar, que se usa para nombrar recursos, como documentos e imágenes en Internet, por su localización.

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

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

2.3 Estilo de programación.

Un estilo de programación no es más que un conjunto de reglas o normas usadas para escribir código y que incluye una gran gama de aspectos dentro del proceso de codificación y es uno de los temas más discutidos entre los programadores. Mientras unos están convencidos de que es un simple problema de estética, otros compiten, como los productores modernos de compiladores, poniendo cada vez más detalles para facilitar la escritura y lectura del código e introducen ciertas normas de estilo.

Estos puntos son importantes pero hay una justificación con más relevancia: la forma de escribir los programas, debe ser una representación exacta de las estructuras del modelo que se define. Por supuesto, que el nivel que se logre al cumplir esta regla depende de las estructuras que defina o disponga el lenguaje de programación, Cap. 1 epígrafe 1.3.1 para ampliar más. El mejor estilo es el que más aporte a la eficiencia, legibilidad y rapidez del proceso de desarrollo y el que potencie los programas más robustos y fáciles de usar (Rodríguez 2006).

Grady Booch, en su libro Object Oriented Design with Applications[14], enuncia como una de la causas de la complejidad del software *"... la dificultad de administrar el proceso de desarrollo, más miembros en un grupo de desarrollo significa mayor complejidad en las comunicaciones... cuando se trabaja en grupo, el reto mayor es mantener la unidad e integridad"*. Se necesita una comunicación eficiente entre los miembros de un grupo de desarrollo, para esto es necesario que todos usen un código de comunicación común.

[14] Redwood City, California. 1991, The Bejamin/Cummings Publishing Company, Inc. p.4

Un buen estilo asegura muchos detalles, algunas de las más relevantes son: los programas más comprensibles, más fácil extender o mantener los módulos de un sistema, las tecnologías desarrolladas son más fáciles de usar en varias plataformas de desarrollo.

Las estructuras usadas en la codificación deben corresponderse exactamente con las estructuras obtenidas en el proceso de diseño del modelo. Esto no es posible exactamente por la mala calidad de los lenguajes de programación existentes con respecto a la implementación que hacen de las estructuras que definen las metodologías de diseño más elaboradas (Rodríguez 2006). Razones por lo que es tan dañino el hecho de que la mayoría de los programadores basen su aprendizaje en los lenguajes y no en métodos generales de diseño. Veamos algunos ejemplos.

Si se le pregunta a una persona que no sea especialista en la ciencia de la computación que explique el procedimiento para encontrar un documento de patente en un listado de ellas, probablemente responda: *"reviso todas las patentes hasta que alguna se corresponda con la que ando buscando"*. Note que la frase *"hasta que"* define el elemento de control del algoritmo, Cap 1 epígrafe 1.3.3.3 para ampliar más. Esta es una idea aceptable, probablemente un programador hubiera dicho: *"recorro las patentes mientras la actual sea distinta a la buscada o no se haya llegado al final de la lista (lista de patente), si la encuentro devuelvo el índice, y si no, menos uno"*. Más elaborada para codificar. Sin embargo, si le pedimos a un programador inexperto que codifique estas ideas, probablemente el resultado sería:

```
public int Buscar(Patente elemento, List<Patente> lista_Elementos)
{
      int resultado = -1;
      for (int i = 0; i < lista_Elementos.Count ; i++)
            if (lista_Elementos[i] == elemento)
            {
                  resultado = i;
                  break;
            }
      return resultado;
}
```

Las ideas originales no se parecen a este resultado. El **for,** en C# es mas flexible que el de otros lenguajes como Pascal pero la idea principal sigue siendo la misma un ciclo de variable de control y significa semánticamente que se va a repetir algo un número determinado de veces, y no *hasta que* ocurra algo. Aunque en el caso de C#, como en C++ o Java y otros pueda usarse como tal hasta que

suceda algo, sin embargo en este caso esto no se cumple porque un **break,** rompe con la estructura definida anteriormente. Si se tratara de traducir exactamente estas instrucciones a lenguaje natural no se obtendrían resultados coherentes. Las instrucciones que rompen estructuras de programación son una mala práctica de codificación, deben usarse lo menos posible, porque introducen efectos no deseados en la comprensión directa de los algoritmos. Otras instrucciones que causan un efecto similar son: **goto**, **exit** y **continue**. No es un problema de eliminarlos por capricho, siempre es posible demostrar que el uso de estas estructuras es una mutilación de las ideas originales(Rodríguez 2006). Reescribamos el algoritmo anterior:

```
public int Buscar(Patente elemento, List<Patente> lista)
{
      int resultado = -1;
      int i = 0;
      while (lista[i] != elemento && i < lista.Count)
            i++;
      if (i < lista.Count)
            resultado = i;
      else
            resultado = -1;
      return resultado;
}
```

Esta versión es mucho más fácil de entender: nos paramos en la primera patente, mientras la actual no es la que ando buscando y queden patentes, me muevo para la siguiente; si la encontré entonces devuelvo su índice, de lo contrario devuelvo menos uno.

Escribamos poniéndole a cada frase su origen en el código: nos paramos en la primera patente (i = 0), mientras (**while**) la actual no es la que ando buscando (list**[i] !=** elemento) y (**and**) queden patentes (i **<** lista.Count), me muevo para la siguiente (i++); si (**if**) la encontré (i **<** list.Count) entonces establezco su índice (resultado **=** i), de lo contrario (**else**) establezco menos uno (resultadp **=** -1), y devuelvo el resultado establecido. Sin comentarios.

2.3.1 Estándar de codificación.

Un estándar de codificación completo comprende todos los aspectos de la generación de código, debe reflejar un estilo armonioso, como si un único programador hubiera escrito todo el código del sistema. Al comenzar un proyecto de software, es necesario establecer un estándar de codificación

para asegurarse de que todos los programadores del proyecto trabajen de forma coordinada. Cuando el proyecto de software incorpore código fuente previo, o bien cuando realice el mantenimiento de un sistema de software creado anteriormente, el estándar de codificación debería establecer cómo operar con la base de código existente.

- Idioma: Se debe emplear un solo idioma y se propone el español, las palabras no se acentuarán.

- Palabras Reservadas: Las palabras reservadas van en minúsculas sin excepción alguna.

- Indentación: En el caso nuestro IDE (VS.NET 2005) se encarga de ajustar los espacios de forma automática. Los inicios ({) y cierre (}) de bloque deben estar alineados debajo de la declaración a la que pertenecen y es a gusto utilizarse o no en el caso de que exista una sola instrucción. Nunca colocar { en la línea de un código cualquiera, esto requiere una línea propia. El cuerpo de declaraciones compuestas (declaraciones como if, while, for, etc.) debe anidarse a una profundidad de 4 espacios.

- Líneas en blanco: Coloque una línea en blanco antes y después de la declaración de una estructura o de una clase.

- Uso de espacios en blanco: Los signos lógicos y de operación deberán estar separados por un espacio antes y después del mismo, ejemplo: `int Total = 15 - 8 + 9;` no se debe utilizar entre el nombre de un método y el paréntesis abierto, `void HacerAlgo(int Posicion);` entre el cast y la expresión, ejemplo: `float Promedio = (float)(5 + 4) / 2;` después del corchete abierto y antes del cerrado de un arreglo, ejemplo `int Cantidad = arrElementos[i].`

- Comentarios de línea: Consiste en un // seguido de un texto. Debe incluirse un espacio simple entre el // y el texto. Se debe evitar comentar cada línea de código. Es mejor colocar el comentario precediendo al bloque de líneas de código. Si el comentario se aplica a un grupo de instrucciones debe estar seguido de una línea en blanco, si se refiere sólo a la siguiente instrucción, se suprime la línea. Este tipo de comentario también puede seguir al código que referencia, debiendo aparecer en la misma línea.

- Declaraciones de clases: Las directivas de visibilidad deben estar identadas con la definición de la clase .Estas directivas deben declararse en el orden siguiente: private, protected, pulbic.

- Variables locales: Las variables deben ser declaradas justo antes de ser utilizadas. Cuando sea posible, la inicialización de las variables locales ocurrirá en el momento de su declaración. Se debe evitar la declaración dentro del cuerpo de un ciclo.

- Declaraciones condicionales: Las declaraciones if - else no deben poseer más de 4 niveles de profundidad, de tenerlos no deben estar en una línea sino repartidos en igual número de líneas como niveles tenga, esto se explica porque es más fácil leer una declaración if - else de arriba para abajo que de izquierda a derecha. La cláusula else, siempre debe estar alineada con la cláusula if.

```
// Se sugiere utilizar cuando las condiciones sobrepasan los márgenes
if (condition1 &&
    condition2 &&
      condition3)
{
.....
}
else
```

- Declaraciones switch: Las expresiones constantes dentro de las declaraciones switch deben estar ordenadas numérica o alfabéticamente dependiendo del tipo al que pertenecen.La cláusula default deberá alinearse con la cláusula case

- Declaraciones for, foreach: El código de inicialización debe realizarse en la declaración del mismo a menos que sean necesario declararlo fuera para darle más tiempo de vida a la variable de control.

- Declaraciones while: Las declaraciones while deben usarse en lugar de for cuando el lenguaje natural a la hora de pensar en algo que se repita lo indique.

- Declaraciones do ...while: El código de la declaración está acotado por las palabras reservadas: do.. while. Lo utilizaremos cuando necesitemos explícitamente que se cumpla al menos una vez ciclo post condicional.

- Regla para identificadores: Los nombres deben formarse a partir de palabras en español únicamente. En el caso de variables y atributos en minúscula y en los nombres de métodos o propiedades la primera letra en Mayúscula en caso de ser una palabra compuesta la primera letra de la que empieza a continuación de la anterior, ejemplo: CambiarEstado(...)

2.4 Análisis de posibles implementaciones, componentes o módulos ya existentes.

El análisis de posibles implementaciones de componentes para el desarrollo del sistema es de gran importancia, permite ahorrar tiempo de trabajo, obtener uno o varios componentes totalmente independientes que pueden ser utilizados en el sistema; lo que significa que se tendrá algunas piezas claves del mismo sin pasar por su proceso de desarrollo y solo restará integrarlas con el resto del sistema, lo cual será tratado en el Cap.2 epígrafe 2.5, entre ellas están:

Componente *Outlook* es el nombre que se le da al ensamblado que se utiliza como menú izquierdo en la interfaz gráfica principal para mostrar las distintas opciones del sistema, el nombre viene dado por su parecido al original creado por Microsoft en su producto Outlook, cliente de correo incluido en su paquete Office. Este componente proporciona al sistema una gran flexibilidad y provee una agradable apariencia gráfica, tiene la posibilidad de ser personalizable fácilmente por los desarrolladores. Esta forma de menú es muy usada en estos días, con implementaciones diferentes, por los desarrolladores de aplicaciones Windows[15]

Componente *TreeListView* es la componente que se utiliza para mostrar reportes, el nombre viene dado por la mezcla de los controles básicos TreeView y ListView y con esta unión se logran potencialidades gráficas más personalizadas y especializadas, para llevar al usuario final la mayor satisfacción y comodidad posible, véase el Cap. 3 epígrafe 3.2.1 sobre interfaces de usuario.

[15] Es el término utilizado para referirse a una aplicación que fue diseñada para ejecutarse en el entorno operativo de WINDOWS y que no puede ejecutarse fuera de él. Todas las aplicaciones WINDOWS tienen una filosofía estandarizada pues utilizan convenios similares para organizar los menús, el estilo de un cuadro de diálogo, el uso del ratón o mouse, entre otros.

Componente *Parser* es el nombre que se le da al ensamblado que se encarga del procesamiento del HTML de las páginas Web. Es donde están los algoritmos para el depurado y extracción de los datos bibliográficos de las patentes, y el cual se explica a fondo en (Castellanos. 2007).

2.5 Estrategias de integración de los componentes.

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

1. Primeramente se tiene en un fichero XML de configuración llamado config.XML, del cual se cargan en tiempo de ejecución[16], los controles de usuarios o plug-ins, que se deseen que el sistema pueda iniciar cuando el usuario lo solicite, ver código fuente correspondiente a lo ya mencionado en Anexo1.

2. Luego en el evento Load del Formulario principal donde se desean mostrar los controles, se lee el fichero Config.XML usando un DataSet[17] para ello, y el método ReadXml de dicha clase, luego iteramos cada uno de los DataRow del DataSet, llamando al método AddPlugin con el nombre y el camino de cada uno de los Plugin correspondiente a cada DataRow, ver código fuente correspondiente a lo ya mencionado en Anexo 2.

3. Implementación del método AddPlugin del paso anterior, ver Anexo 3

4. Implementación de la clase base Plugin, ver Anexo 4

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

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

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

[16] Tiempo de Ejecución: Momento en que la aplicación esta corriendo.
[17] DataSet: Clase del Framework .NET para el manejo de datos en memoria, que pueden ser Tablas, esquemas, etc.

- Ponerle un nombre descriptivo al control de usuario que se este creando.
- Adicionar la directiva using para PlugInLib.
- Cambiar la clase base del control de usuario System.Windows.Forms.UserControl por PlugIn.
- Conectar cualquier evento que se quiera enviar a la aplicación principal.
- Adicionar las redefiniciones de métodos que sean necesarios para la llamadas desde la aplicación principal hacia el Plugin.
- Construir la Interfaz de usuario en el control de usuario como cual otro control (a su gusto).

6. Implementación de la descripción anterior, ver Anexo 5

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

2.6 Estructuras de datos apropiadas para la implementación de estos algoritmos.

Las Estructuras de Datos se pueden definir como la organización de la información que permite un determinado lenguaje de programación. Cada estructura posee sus propias características de almacenamiento y recuperación de los datos, para ampliar mas véase Cap. 1 epígrafe 1.3.3.1, otro aspecto de gran importancia de la selección de las estructuras de datos correctas, que se utilicen para manejar la información en la computadora, es la influencia que estas tienen en el desempeño de los algoritmos, para ampliar mas véase Cap. 1 epígrafe 1.3.3.3.

En la implementación de este sistema se utilizan estructuras de datos disponibles en la biblioteca de clases base (BCL[18]) del FrameWork[19] 2.0, como listas genéricas, estás se usan en el almacenamiento de colecciones de objetos, esta estructura está presente en los componentes o

[18] BCL: Base Class Library.
[19] FrameWork: Plataforma de trabajo

clases que gestionan listas o colecciones de objetos como por ejemplo el modulo de "ProcesamientoHTML" la clase "CC_Resultado_Procesado" contiene una colección de "CE_Patente", quedando de la forma siguiente:

```
namespace Predictor.Negocio.ProcesamientoHTML
{
    public class CC_Resultado_Procesado
    {
        private List<CE_Patente> listaPatentes;
        ...
    }
}
```

Las listas constituye la estructura de dato medular del sistema. Esta estructura de dato, ya implementada por el FrameWork de .NET es muy parecida a otras listas que puedan tener otros IDEs en su biblioteca de componentes, pues es una estructura de datos que los desarrolladores crean siguiendo determinados estándares y principios, esto facilita su uso, y provee las funcionalidades que podamos necesitar en un momento determinado, como son adicionar, eliminar, insertar elementos de las mismas, por mencionar algunas entre las muchas que tienen.

Otra estructura de dato utilizada en el sistema de forma implícita son los árboles, a través del componente TreeListView en el subsistema de reporte, para poder mostrar a través de los nodos[20] del árbol de reporte los distintos datos bibliográficos de las patentes, facilitando la visualización de estos por parte de los usuarios del sistema. También tenemos como estructura de dato, el uso de ficheros textos, para almacenar las secuencias de caracteres, que son extraídas de las páginas Web descargadas, así como la información de las patentes lista para analizar, los datos de configuración de conexión a Internet entre otros.

[20] Un nodo es la unidad sobre la que se construye el árbol y puede tener cero o más nodos hijos conectados a él.

CAPÍTULO 3: Diseño e Implementación del sistema.

Durante el ciclo de desarrollo iterativo es posible pasar a la fase de diseño, una vez terminados los documentos del análisis véase en (Piñeiro 2007). Durante el flujo de diseño se logra una solución lógica que se funda en el paradigma orientado a objetos. Su esencia es la elaboración de diagramas de interacción, que muestran gráficamente como los objetos se comunican entre ellos, con el fin de cumplir con los requerimientos. El advenimiento de los diagramas de interacción nos permite dibujar diagramas de clases de diseño que resumen la definición de las clases (e interfaces) implementables en software y luego en la codificación, empezamos con el resultado del diseño y se implementa el sistema en términos de componentes, es decir ficheros de código fuente, script, ejecutables, entre otros elementos (Ivar Jacobson 2000).

3.1 Modelo de diseño del sistema.

El modelo de diseño describe la realización física de los casos de uso centrándose en como los requerimientos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar. Además el modelo de diseño sirve de abstracción de la implementación del sistema, y es de ese modo utilizado como una entrada fundamental de las actividades de implementación (Ivar Jacobson 2000).

En este trabajo el modelo de diseño esta compuesto por un sistema de diseño que contiene entre otros aspectos, cada uno de los Subsistemas de diseño como se muestra a continuación:

- *Subsistema de Autenticación*: Contienes clases de diseño relacionadas con la autenticación de los usuarios del sistema
- *Subsistema de Configuración:* Contiene clases de diseño relacionadas con la configuración del sistema.
- *Subsistema de Búsqueda:* Contiene clases de diseño relacionadas con la Descarga de información de internet.

- *Subsistema Común:* Contiene clases de diseño comunes para varios Subsistemas de diseño, por lo que se decidió crear este Subsistema aparte, evitando ligar o acoplar fuertemente otros Subsistemas entre ellos.

- *Subsistema Reportes:* Contiene clases de diseño relacionadas con los reportes de una búsqueda en internet.

- *Subsistema de Procesamiento HTML:* Contiene las clases de diseño relacionadas con el procesamiento del código HTML de las paginas web descargadas de internet.

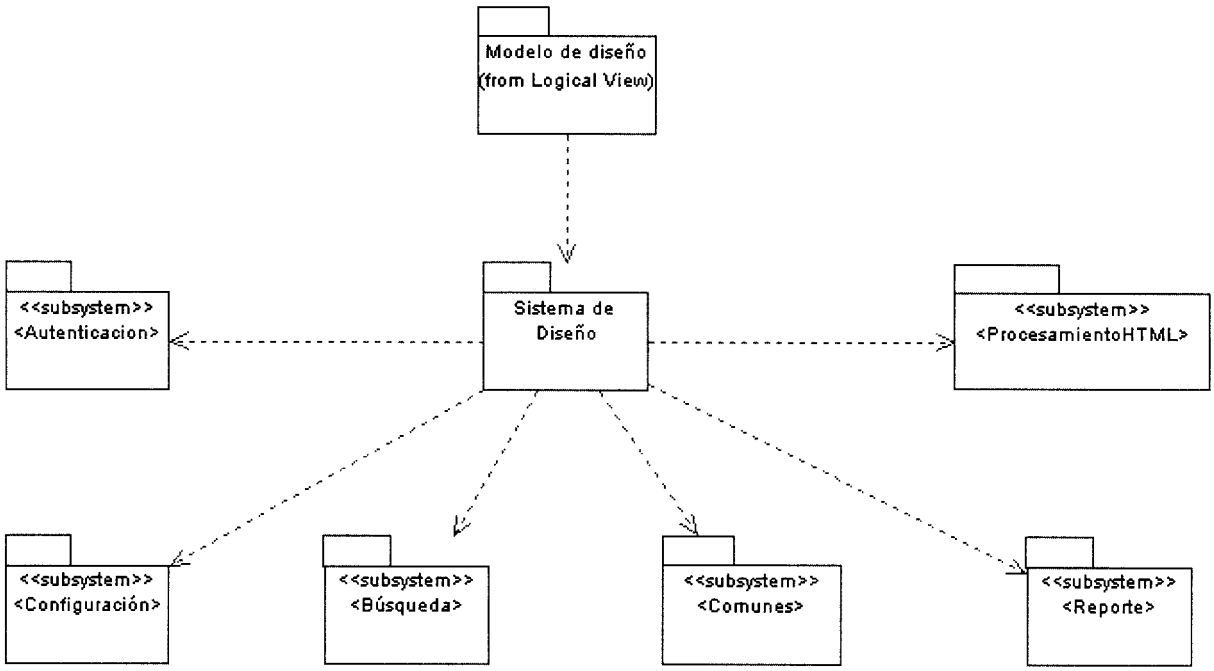


Figura 5: Modelo de Diseño del Sistema.

3.1.2 Diagramas de interacción (Secuencia).

Antes de iniciar el diseño lógico de como funcionará un sistema, es necesario investigar y definir su comportamiento como una "caja negra"[21]. El comportamiento del sistema es una descripción de lo

[21] Caja Negra: Sabes cuales son sus entradas y las salidas que debe tener, no sabemos como lo hace

que hace, sin explicar la manera en que lo hace. Una parte de dicha descripción es un diagrama de secuencia del sistema. Los diagramas de interacción son los más importantes desde el punto de vista de la preparación de un buen diseño y exigen gran dedicación y esfuerzo creativo.

Para elaborarlos hay que aplicar los principios de la asignación de responsabilidades y utilizar los patrones de diseño (Larman 1999). En este trabajo se realizaron los diagramas de interacción (secuencia) para cada uno de los casos de uso del sistema[22], tal y como se muestran en los anexos.

Cada diagrama de secuencia que abordaremos en este trabajo describe, la interacción de un actor con un caso de uso específico y el curso de los eventos internos generados por la acción del actor.

3.1.2.1 Diagrama de Secuencia del Caso de Uso Autenticación.

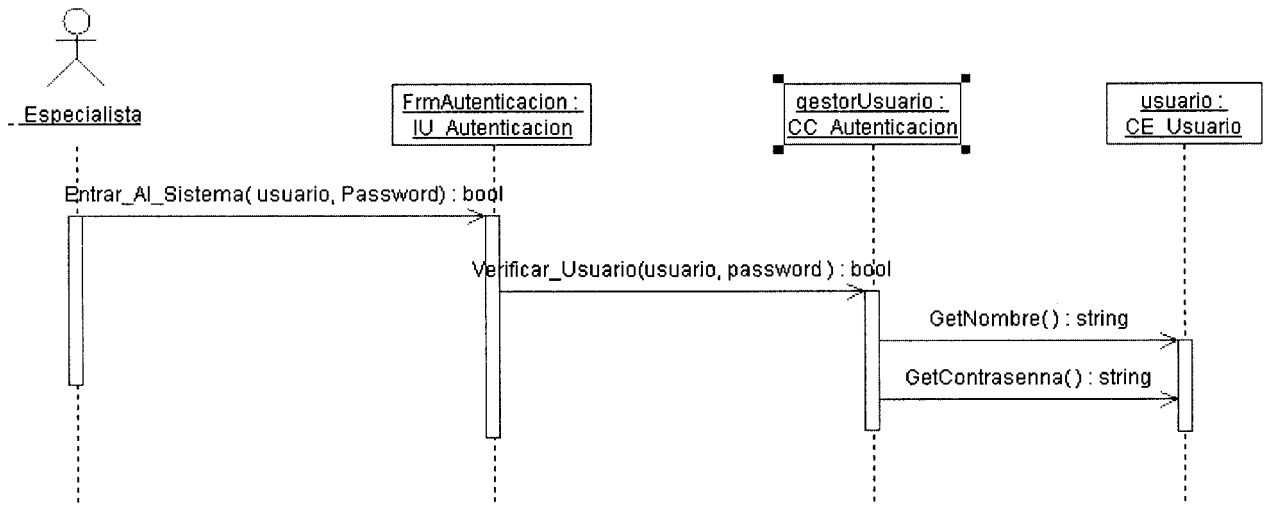


Figura 6: Diagrama de secuencia del caso de uso Autenticación.

[22] Caso de Uso: Es una descripción de un proceso de principio a fin relativamente amplia, descripción que suele abarcar muchos pasos o transacciones y normalmente no es un paso ni una actividad individual del proceso.

3.1.2.2 Diagrama de Secuencia del Caso de Uso Configuración (Internet).

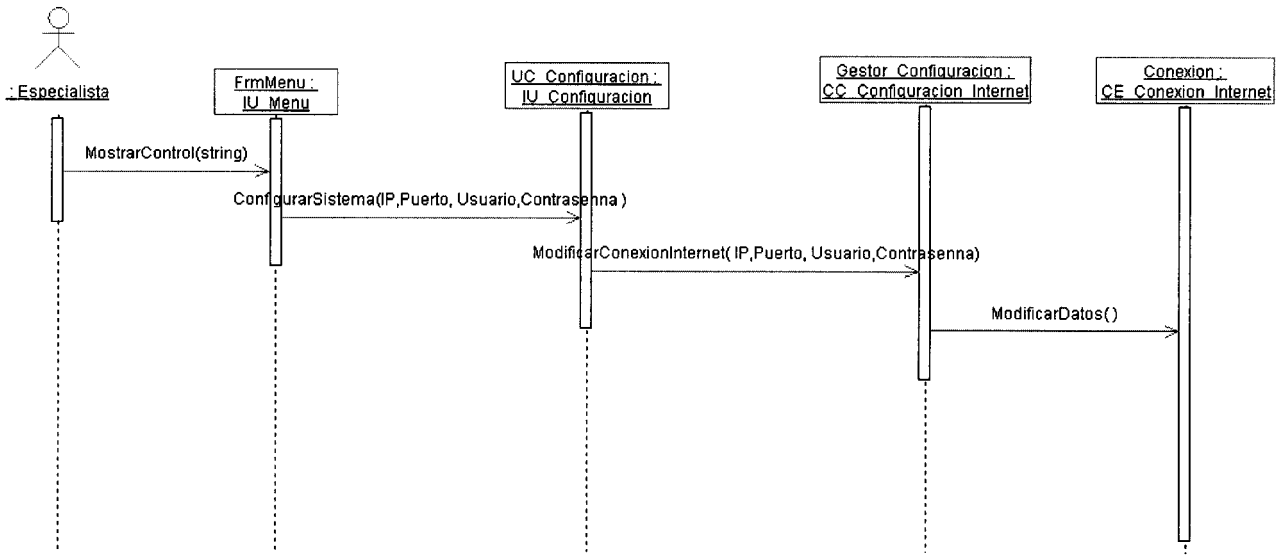


Figura 7: Diagrama de Secuencia del Caso de Uso Configuración (internet).

3.1.2.3 Diagrama de Secuencia del Caso de Uso Búsqueda Básica.

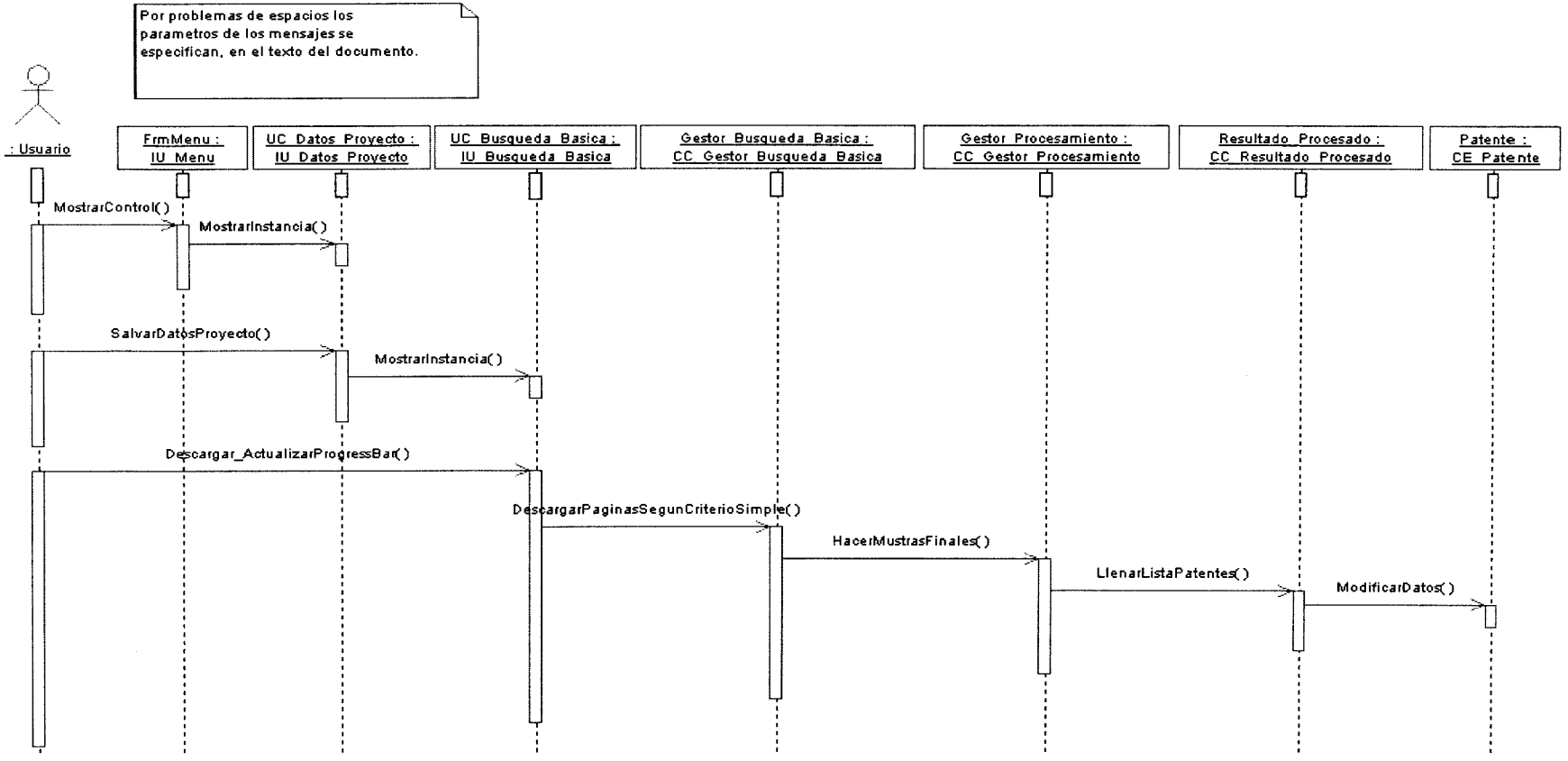


Figura 8: Diagrama de Secuencia del Caso de Uso Búsqueda Básica.

Descripción de los mensajes en el diagrama de secuencia del caso de uso Búsqueda Básica:

Nota: Los parámetros de los mensajes no se colocaron en el diagrama por falta de espacio, y se mencionan a continuación en la descripción de cada uno de los mensajes.

- *MostrarControl:* El usuario selecciona en el menú principal el proceso que desea realizar, este mensaje recibe como parámetro el nombre del control de usuario que desea cargar según el proceso seleccionado, en este caso es IU_Datos_Proyecto.

- *MostrarInstancia:* Muestra una instancia de la Interfaz de usuario correspondiente.

- *SalvarDatosProyecto:* El usuario entra los datos del nuevo proyecto a realizar, este mensaje recibe como parámetro, el nombre del proyecto, título, comentarios y carpeta de trabajo, luego se muestra la interfaz de usuario IU_Búsqueda_Básica.

- *DescargarActualizarProgressBar:* El usuario comienza el proceso de descarga de patentes de internet

- *DescargarPaginasSegunCriterioSimple:* Lleva a cabo la gestión de las descargas de las paginas de internet, recibe como parámetros el Servidor de Base Datos y el criterio.

- *HacerMuestrasFinales:* Una vez descargadas las paginas web, se procesa el HTML de las paginas generando 2 ficheros muestras finales para que sean cargados en otras aplicaciones como son Excel y Procite, y el cliente pueda hacer sus análisis.

- *LlenarListaPatentes:* Llena una colección de patentes con los datos de las mismas que fueron extraido de las paginas descargadas

- *ModificarDatos:* Representa las Propiedades de acceso (Set) para establecer los valores de cada uno de los atributos de las patentes.

3.1.2.4 Diagrama de Secuencia del Caso de Uso Búsqueda Avanzada.

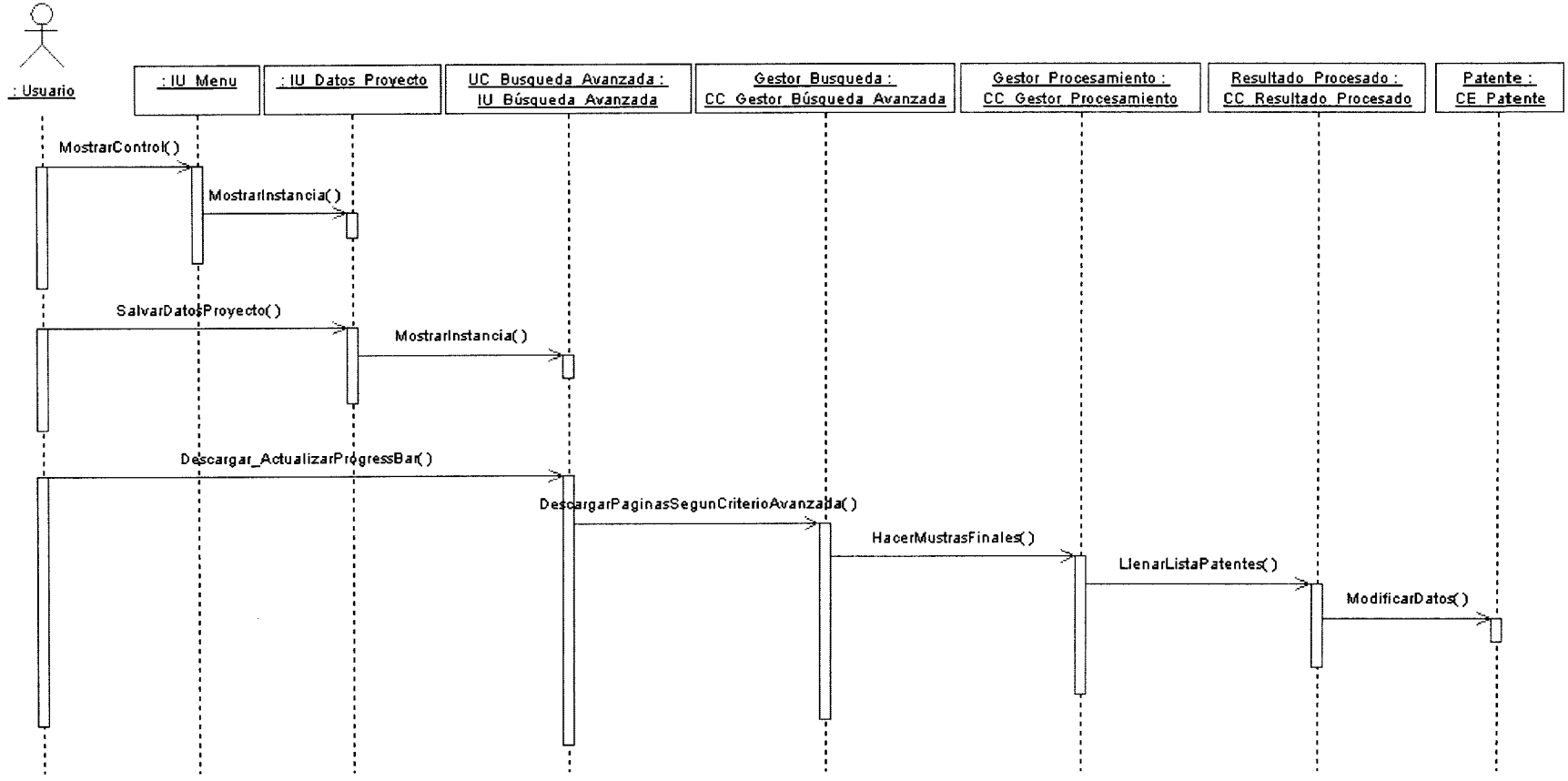


Figura 9: Diagrama de Secuencia del Caso de Uso Búsqueda Avanzada.

Descripción de los mensajes en el diagrama de secuencia del caso de uso Búsqueda Avanzada:

Es importante aclarar que los parámetros de los mensajes no se colocaron en el diagrama por falta de espacio, y se mencionan a continuación en la descripción de cada uno de los mensajes.

- *MostrarControl:* El usuario selecciona en el menú principal el proceso que desea realizar, este mensaje recibe como parámetro el nombre del control de usuario que desea cargar según el proceso seleccionado, en este caso es IU_Datos_Proyecto.

- *MostrarInstancia:* Muestra una instancia de la Interfaz de usuario correspondiente.

- *SalvarDatosProyecto:* El usuario entra los datos del nuevo proyecto a realizar, este mensaje recibe como parámetro, el nombre del proyecto, título, comentarios y carpeta de trabajo, luego se muestra la interfaz de usuario IU_Búsqueda_Básica.

- *DescargarActualizarProgressBar:* El usuario comienza el proceso de descarga de patentes de internet

- *DescargarPaginasSegunCriterioAvanzada:* Lleva a cabo la gestión de las descargas de las paginas de internet, recibe como parámetros el servidor de Base Datos, el criterio, fecha de publicación, clasificación internacional, y solicitante, estos 3 últimos parámetros sus valores puede ser nulos.

- *HacerMuestrasFinales:* Una vez descargadas las paginas web, se procesa el HTML de las paginas generando 2 ficheros muestras finales para que sean cargados en otras aplicaciones como son Excel y Procite, y el cliente pueda hacer sus análisis.

- *LlenarListaPatentes:* Llena una colección de patentes con los datos de las mismas que fueron extraido de las paginas descargadas

- *ModificarDatos:* Representa las Propiedades de acceso (Set) para establecer los valores de cada uno de los atributos de las patentes.

3.1.2.5 Diagrama de Secuencia del Caso de Uso Procesamiento de Patentes.

En este diagrama, está incluido en la Búsqueda de Patente Básica y Avanzada, pero el usuario puede inicializarlo sin pasar antes por algunas de estas búsquedas, véase (Piñeiro 2007)

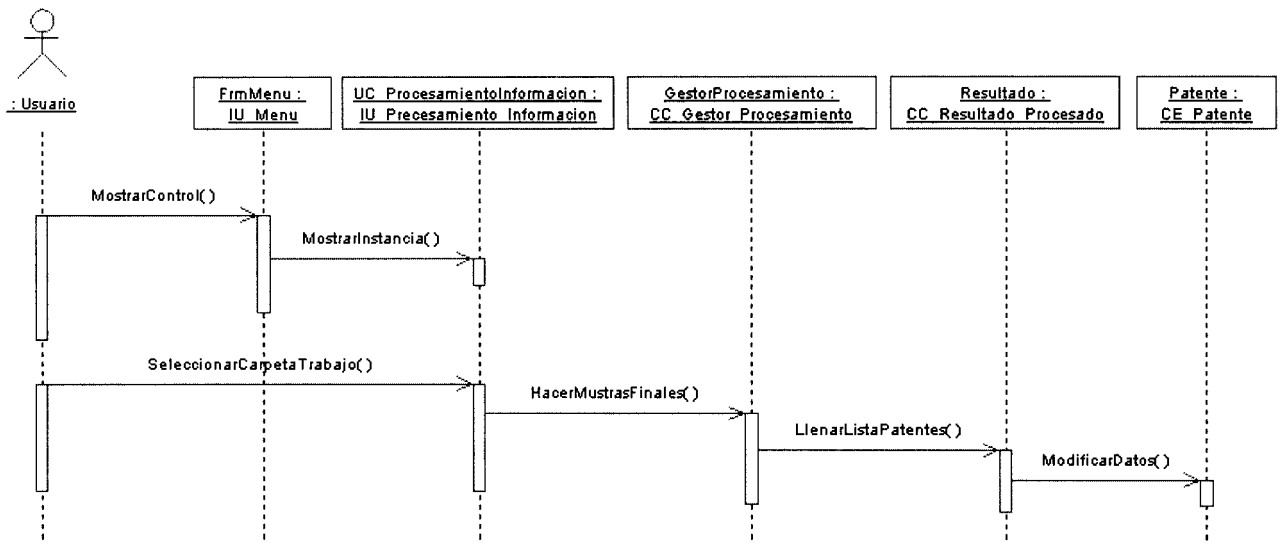


Figura 10: Diagrama de Secuencia del Caso de Uso Procesamiento de Patentes.

3.1.2.6 Diagrama de Secuencia del Caso de Uso Reportes.

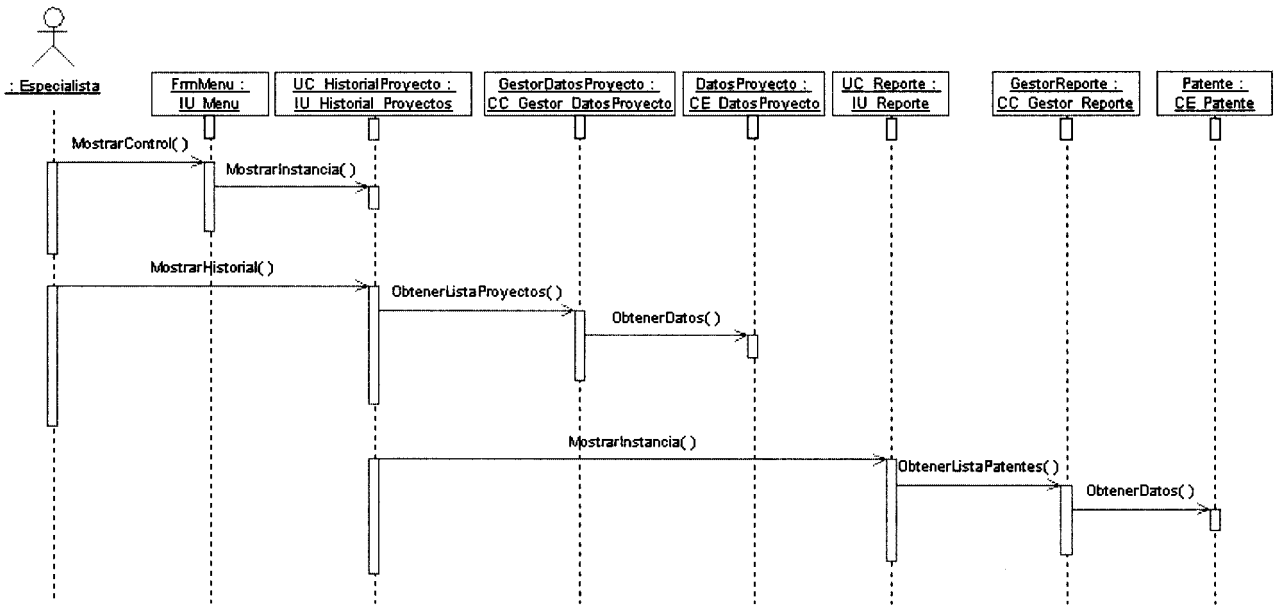


Figura 11: Diagrama de Secuencia del Caso de Uso Reportes.

Descripción de los mensajes en el diagrama de secuencia del caso de uso Procesamiento de Patentes:

Los parámetros de los mensajes no se colocaron en el diagrama por falta de espacio, y se mencionan a continuación en la descripción de cada uno de los mensajes.

- *MostrarControl:* El usuario selecciona en el menú principal el proceso que desea realizar, este mensaje recibe como parámetro el nombre del control de usuario que desea cargar según el proceso seleccionado, en este caso es IU_Procesamiento_Información.

- *MostrarInstancia:* Muestra una instancia de la Interfaz de usuario correspondiente.

- *SeleccionarCarpetaTrabajo:* El usuario especifica donde están los elementos que se desean procesar.

- *HacerMuestrasFinales:* Una vez descargadas las paginas web, se procesa el HTML de las paginas generando 2 ficheros muestras finales para que sean cargados en otras aplicaciones como son Excel y Procite, y el cliente pueda hacer sus análisis.

- *LlenarListaPatentes:* Llena una colección de patentes con los datos de las mismas que fueron extraido de las paginas descargadas

- *ModificarDatos:* Representa las Propiedades de acceso (Set) para establecer los valores de cada uno de los atributos de las patentes.

Descripción de los mensajes en el diagrama de secuencia del caso de uso Reportes:

Los parámetros de los mensajes no se colocaron en el diagrama por falta de espacio, y se mencionan a continuación en la descripción de cada uno de los mensajes.

- *MostrarControl:* El usuario selecciona en el menú principal el proceso que desea realizar, este mensaje recibe como parámetro el nombre del control de usuario que desea cargar según el proceso seleccionado, en este caso es IU_Datos_Proyecto.

- *MostrarInstancia:* Muestra una instancia de la Interfaz de usuario correspondiente.

- *MostrarHistorial:* Muestra el historial de todos los proyectos.

- *ObtenerListaProyectos:* Devuelve el listado de todos los proyectos.

- *ObtenerDatos (Proyecto):* Representa las Propiedades de acceso (Get) para devolver los valores de cada uno de los atributos de los proyectos.

- *ObtenerListaPatentes:* Devuelve el listado de todas las patentes.

- *ObtenerDatos (Patente):* Representa las Propiedades de acceso (Get) para devolver los valores de cada uno de los atributos de las patentes.

3.1.3 Fundamentación de la utilización de patrones de diseño.

Los diseñadores de software especialistas en el paradigma orientado a objetos van formando un amplio repertorio de principios generales y de expresiones que los guían al crear software. A estos principios se les llama patrones. En la terminología de objetos, el patrón es una descripción de un problema y su solución, que recibe un nombre y puede emplearse en otros contextos; en teoría, indica la manera de utilizarlo en circunstancias diversas. Muchos patrones ofrecen orientación sobre cómo asignar las responsabilidades a los objetos ante determinada categoría de problemas (Larman 1999).

Los patrones no se proponen descubrir ni expresar nuevos principios de la ingeniería del software. Ellos codifican el conocimiento, las expresiones y los principios ya existentes, cuanto más analizados y generalizados, mucho mejor véase en el Cap. 1 epígrafe 1.3.4.2

A continuación se mencionan y justifica el uso de los siguientes patrones GRASP[23] en este sistema y luego algunos patrones del famoso catalogo (Madrid 2003).

Patrones GRASP usados en el sistema:

- *Experto*: Partiendo de que se conoce que una determinada clase posee una determinada información, se definió por este patrón donde colocar en cada clase las funcionalidades que necesitan de esa información, ya que esta clase seria nuestro experto en información (Larman 1999).

  Ejemplo: las clases que contienen una colección de elementos de otra, son las mejores candidatas a contener la definición e implementación de los métodos de manipulación de estos elementos como son las funcionalidades de insertar, buscar, eliminar estos de esta colección (Larman 1999).

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

  a. Contener Y

[23] GRASP: Patrones generales para la asignación de responsabilidades

b. Agregar Y

c. Tiene los datos de inicialización de Y

d. Registra a Y

e. Utiliza a Y

- *Alta Cohesión:* Es necesario controlar la complejidad de cada clase utilizada en el sistema para mantener un buen comportamiento de las mismas, es por esto que las clases que fueron detectadas con muchas funcionalidades, se analizó la pasibilidad de dividir en otras clases estas responsabilidades de manera que se repartieran equitativamente el peso de la complejidad, manteniendo además la coherencia del modelo de clases de diseño (Larman 1999).

- *Bajo Acoplamiento:* Si se desea que nuestro sistema genere componentes reusables, fácil de mantener y entender este es el patrón a tener presente en todo momento, para esto se trabajó cada subsistema de manera independiente (componentes para la descarga de información, procesamiento, configuración y reportes) y se definió una interfaz de comunicación con el exterior en cada uno de ellos (Larman 1999).

- *Controlador:* Se utilizó para la comunicación entre cada una de las capas del sistema, ejemplo de esto fue la creación de una clase Fachada que se encargara de la comunicación entre los eventos externos del sistema en la capa de presentación y los componentes de la capa de negocio (Larman 1999).

Patrones del GOF[24], usados en el sistema:

El patrón Fachada (Facade), se usa en este sistema para facilitar una interfaz simple a un subsistema complejo, y para estructurar en capas el sistema, definiendo un controlador que se encargue de gestionar los eventos externos del sistema hacia la capa de negocio.

[24] 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.

El patrón Observador (Observer), se usa en este sistema para definir una dependencia "uno-a-muchos" entre objetos, para que, cuando uno de ellos cambie su estado, todos los que dependan de él sean avisados y puedan actualizarse convenientemente, en otras palabras se usa en el sistema cuando un cambio en un objeto requiere cambiar otros objetos y no se conoce cuántos objetos exactamente lo necesitarán. Cuando un objeto debe poder notificar sus cambios a otros objetos, sin necesidad de saber nada acerca de sus identidades (acoplamiento débil). Ejemplo de esto se tienen en el proceso de descarga de información, donde los distintos controles de la interfaz de usuario (ProgressBar, ListView, Label's, entre otros), cambian su estado constantemente cada vez que se realiza una solicitud web en la capa de negocio, y se hace necesario que estas clases que aseguran la búsqueda en internet en el negocio tengan un bajo acoplamiento con los componentes de la interfaz gráfica para no violar los principios de la arquitectura en capas, recordemos que una capa solo debe referenciar a la inmediatamente inferior y no viceversa, ver mas en Cap. 2 epígrafe 2.1.1.

El patrón Solitario (Singleton), se usa en este sistema para garantizar que una clase determinada sólo tenga una única instancia, proporcionando un punto de acceso global a la misma, esto trae muchas ventajas como: al tener el acceso a la "Instancia única" más controlado, es fácilmente modificable para permitir más de una instancia y, en general, para controlar el número de las mismas.

Los objetivos específicos que se buscan con el uso de estos patrones fueron los mencionados anteriormente y Los objetivos generales que se persiguen con el uso de los mismos en este sistema son:

- Proporcionar elementos reusables en el diseño de software.
- Evitar la reiteración en la búsqueda de soluciones a problemas ya conocidos y solucionados anteriormente.
- Formalizar un vocabulario común entre miembros del equipo.
- Estandarizar el modo en que se realiza el diseño.

3.1.4 Diagramas de Clases de Diseño.

Los diagramas de clases del diseño describen gráficamente las especificaciones de las clases de software y las interacciones en una aplicación, se trata de una perspectiva desde el punto de vista del diseño.

3.1.4.1 Diagrama de clases de Diseño correspondiente al caso de uso Búsqueda Básica.

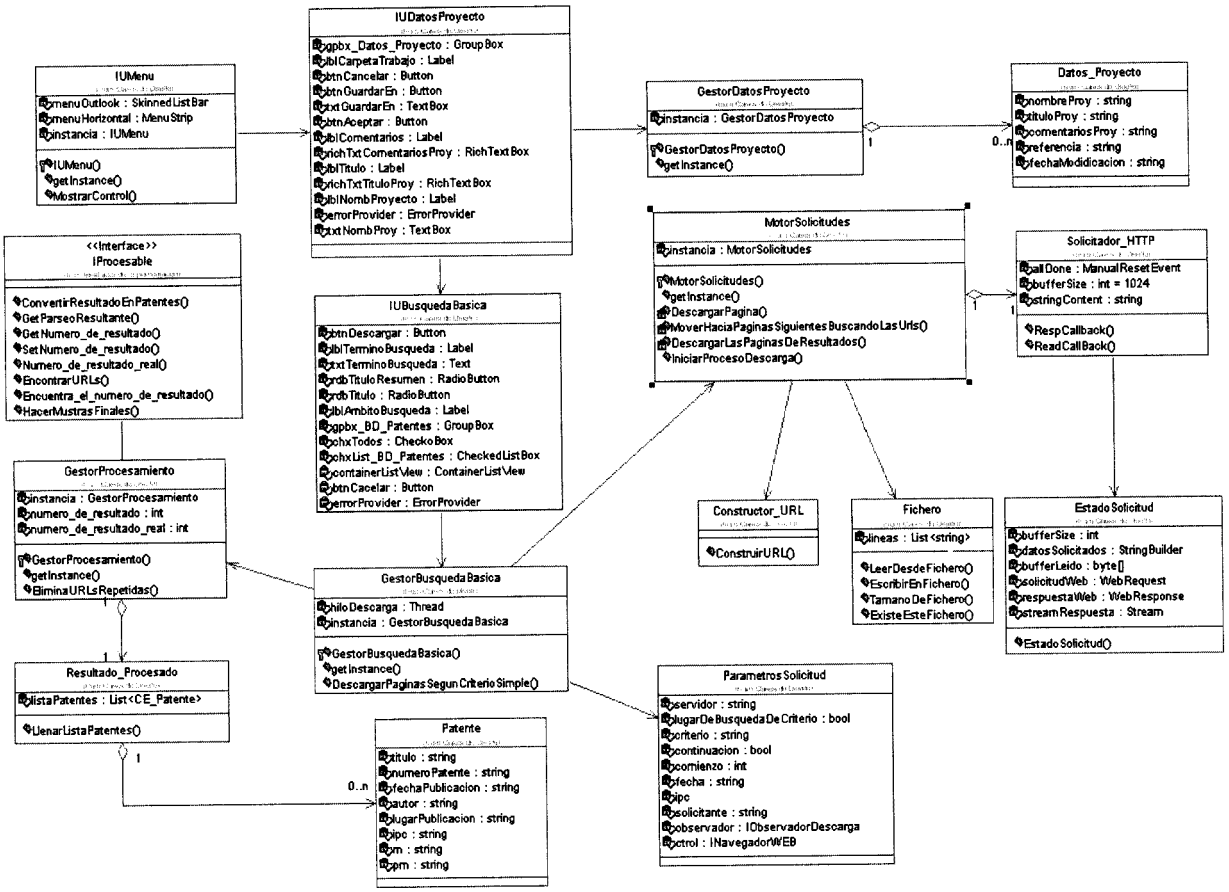


Figura 12: Diagrama de clase s de Diseño correspondiente al caso de uso Búsqueda Básica.

Las propiedades de algunas clases no se colocaron en el diagrama por falta de espacio, y por cada uno de los atributos de estas clases hay una propiedad pública que garantiza la escritura y lectura del atributo, estas clases son las que se mencionan a continuación:

- ParametrosSolicitud
- Patente
- Datos_Proyecto
- EstadoSolicitud

3.1.4.2 Diagrama de clases de Diseño correspondiente al caso de uso Búsqueda Avanzada.

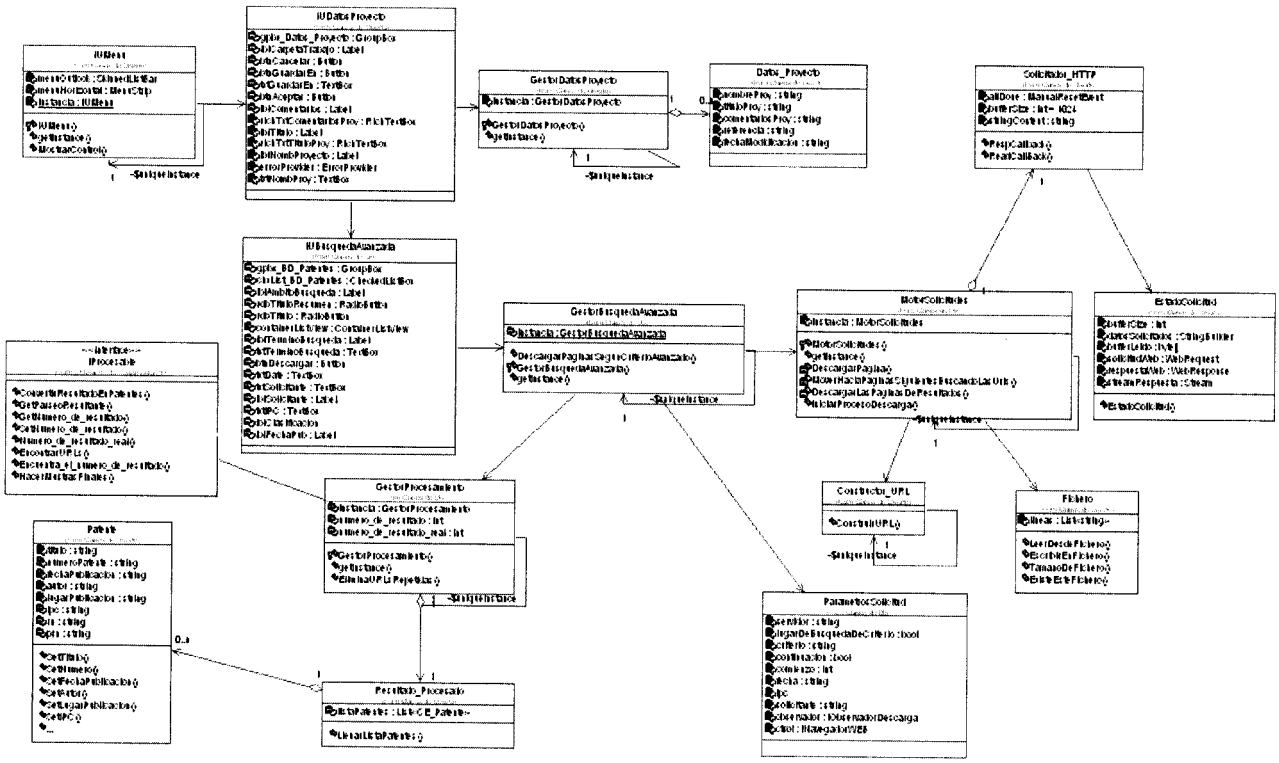


Figura 13: Diagrama de clases de Diseño correspondiente al caso de uso Búsqueda Avanzada.

Las propiedades de algunas clases no se colocaron en el diagrama por falta de espacio, y por cada uno de los atributos de estas clases hay una propiedad pública que garantiza la escritura y lectura del atributo, estas clases son las que se mencionan a continuación:

- ParametrosSolicitud

- Patente
- Datos_Proyecto
- EstadoSolicitud

3.1.4.3 Diagrama de clases de Diseño correspondiente al caso de uso Configuración.

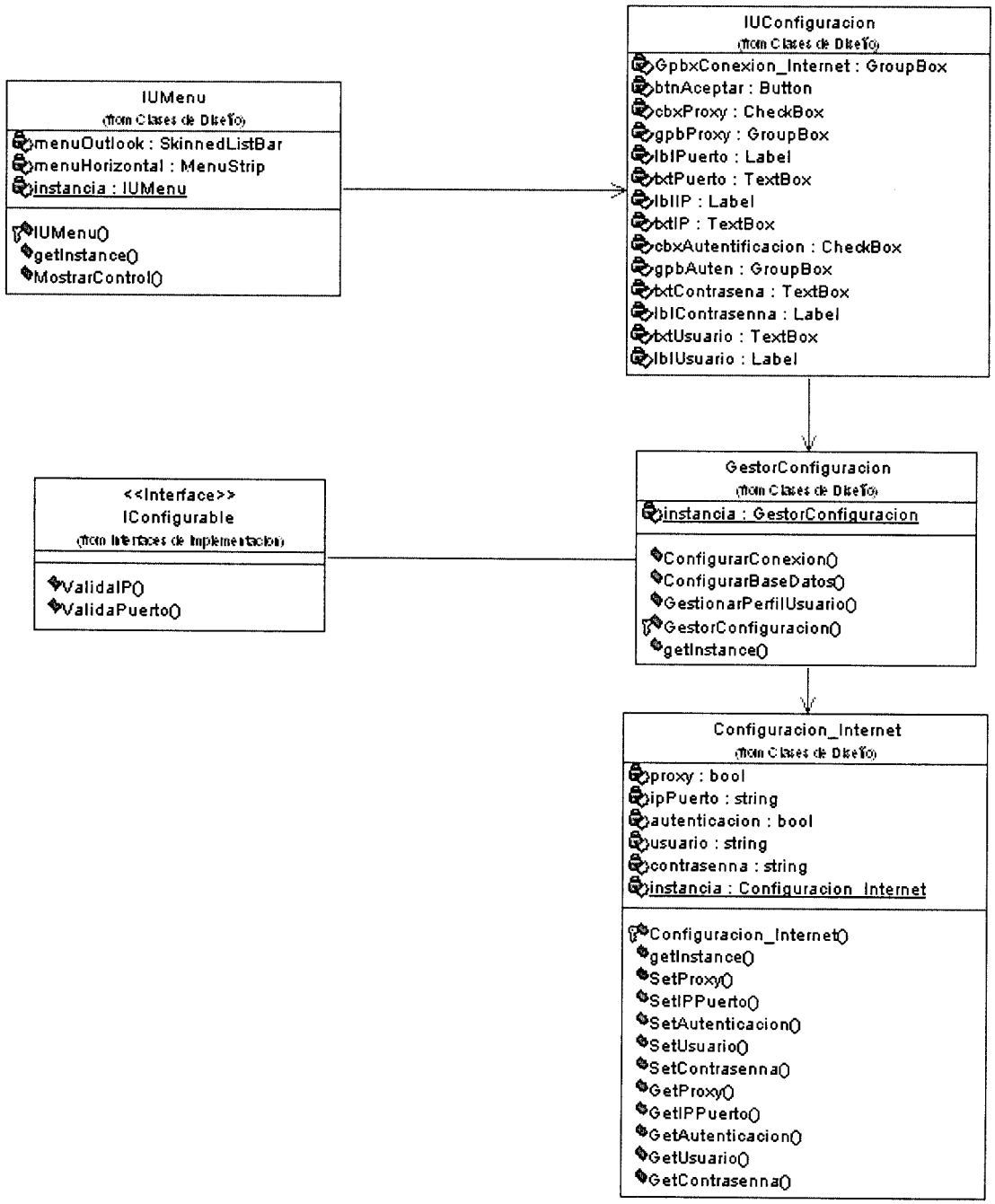


Figura 14: Diagrama de clases de Diseño correspondiente al caso de uso Configuración.

3.1.4.4 Diagrama de clases de Diseño correspondiente al caso de uso Reporte.

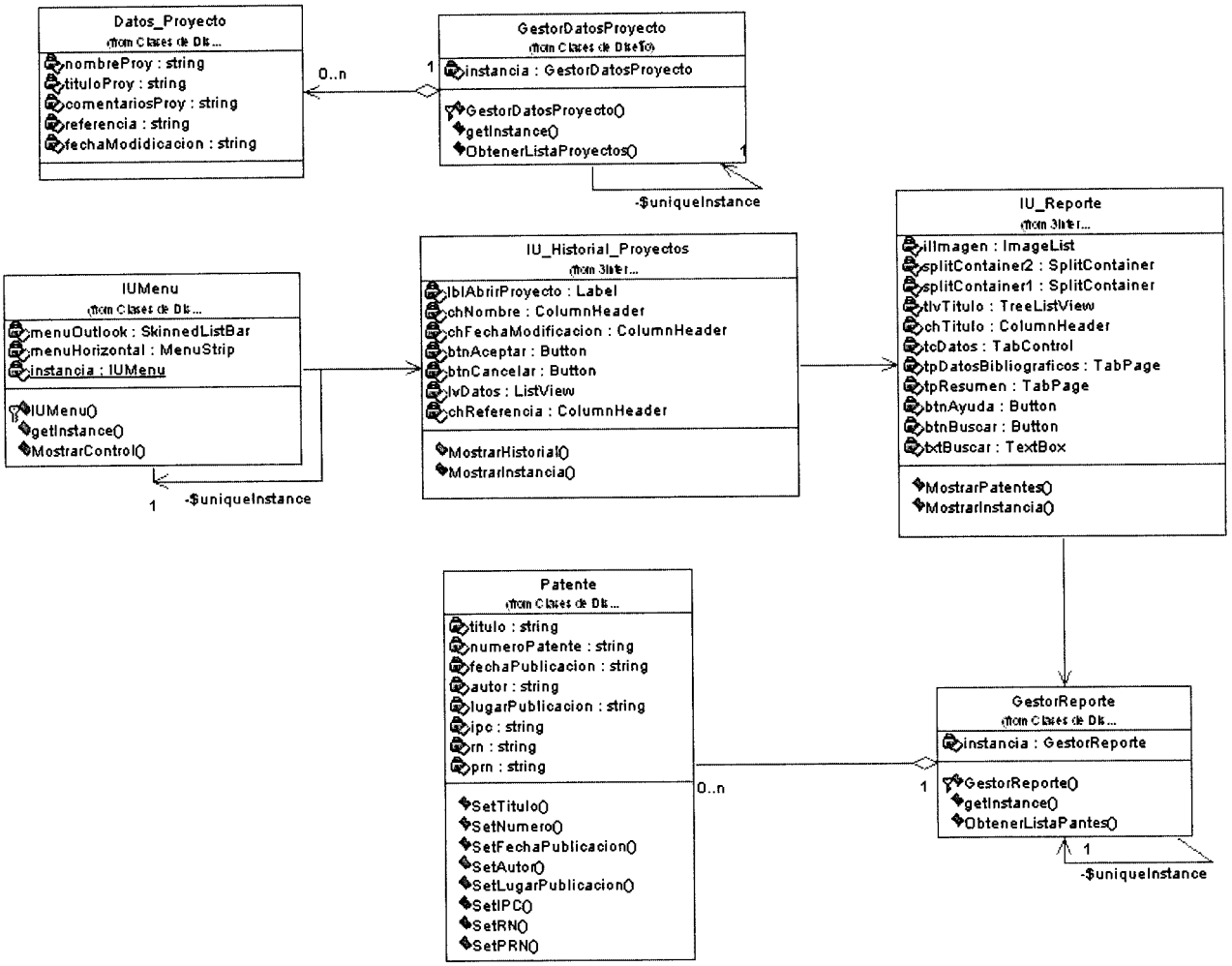


Figura 15: Diagrama de clases de Diseño correspondiente al caso de uso Reporte.

3.2 Modelo de implementación del sistema.

El modelo de implementación describe como los elementos del modelo de diseño (las clases) se implementan en términos de componentes (Ficheros, Ejecutables, etc.), Describe también como se organizan los componentes de acuerdo con los mecanismos de construcción disponibles en el entorno de implementación y en el lenguaje o lenguajes de programación utilizados y como dependen los componentes unos de otros.

Cada Subsistema de implementación posee una traza al Subsistema de diseño donde se encuentra sus clases correspondientes, como se muestra a continuación y los distintos Subsistemas de implementación se pueden ver en el Anexo 6, así como los Subsistemas de diseño se pueden ver en el Cap. 3 epígrafe 3.1

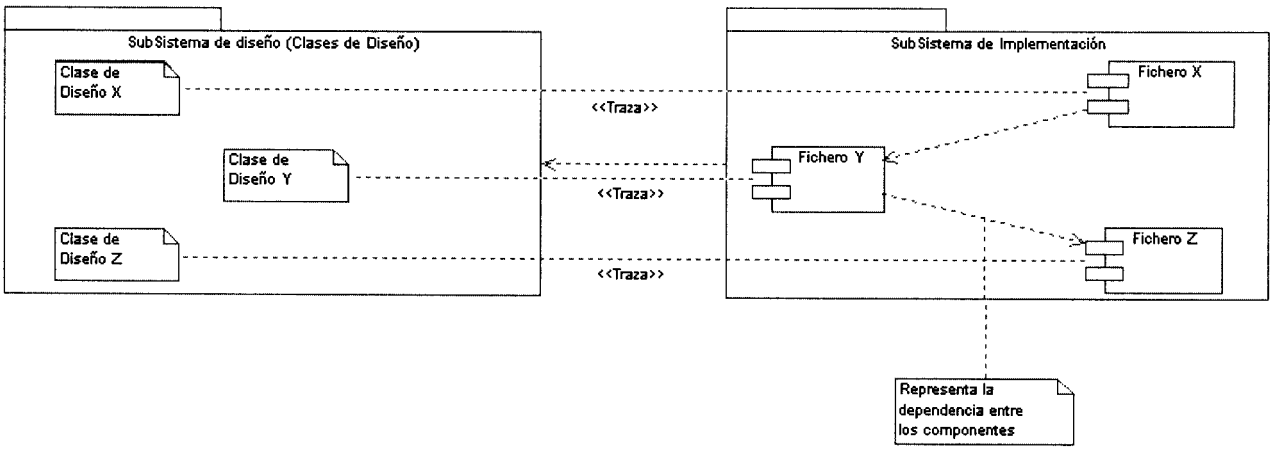


Figura 16: Modelo de implementación del sistema.

3.2.1 Interfaz de Usuario.

El diseño de la interfaz de usuario comienza con la identificación de los requisitos del usuario, de las tareas y del entorno. Una vez identificada las tareas, se crean y se analizan los escenarios del usuario, para definir el conjunto de objetos y de acciones de la interfaz (Pressman 2002). Esto es lo que forma la base para la creación del formato de las pantallas que representan el diseño gráfico y están compuesto por la colocación de iconos, la definición del texto descriptivo en pantalla, títulos de las ventanas y la especificación de elementos principales y secundarios del menú, entre otros.

3.2.1.1 Principios generales para la definición de las interfaces de usuario.

- Definir las de interacción del usuario con la interfaz grafica del sistema de manera que no obligue al usuario a realizar acciones innecesarias y no deseadas.

- Permitir que la interacción del usuario con un determinado proceso del sistema se pueda interrumpir ó deshacer.

- Ocultar al usuario los detalles técnicos, el usuario no tiene que conocer el sistema operativo, las funciones de gestión de archivos, o cualquier otro detalle de la tecnología informática.

- Reducir la carga de memoria del usuario, Siempre que sea posible, el sistema deberá recordar la información pertinente y ayudar a que el usuario recuerde el proceso o escenario de interacción que esta realizando.

- Establecer valores por defecto útiles, el conjunto inicial de valores por defecto tendrá que ser de utilidad para el usuario, pero este también deberá tener la posibilidad de especificar sus propias preferencias o configuraciones. Sin embargo, deberá disponer de una opción de re inicialización que le permita volver a definir los valores por defecto.

3.2.1.2 Interfaces de cada proceso.

Figura 17: Pantalla de Presentación (Splash).

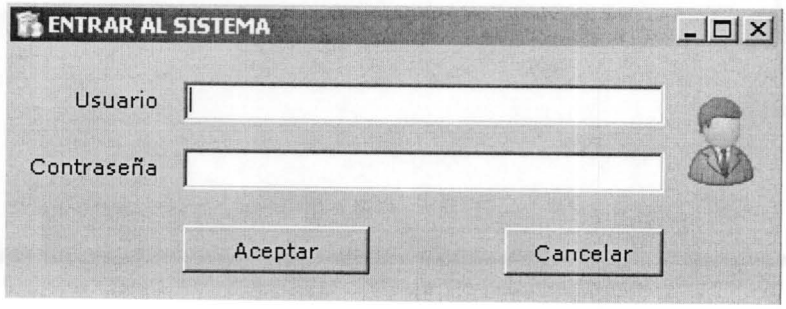


Figura 18: Pantalla de Autenticación.

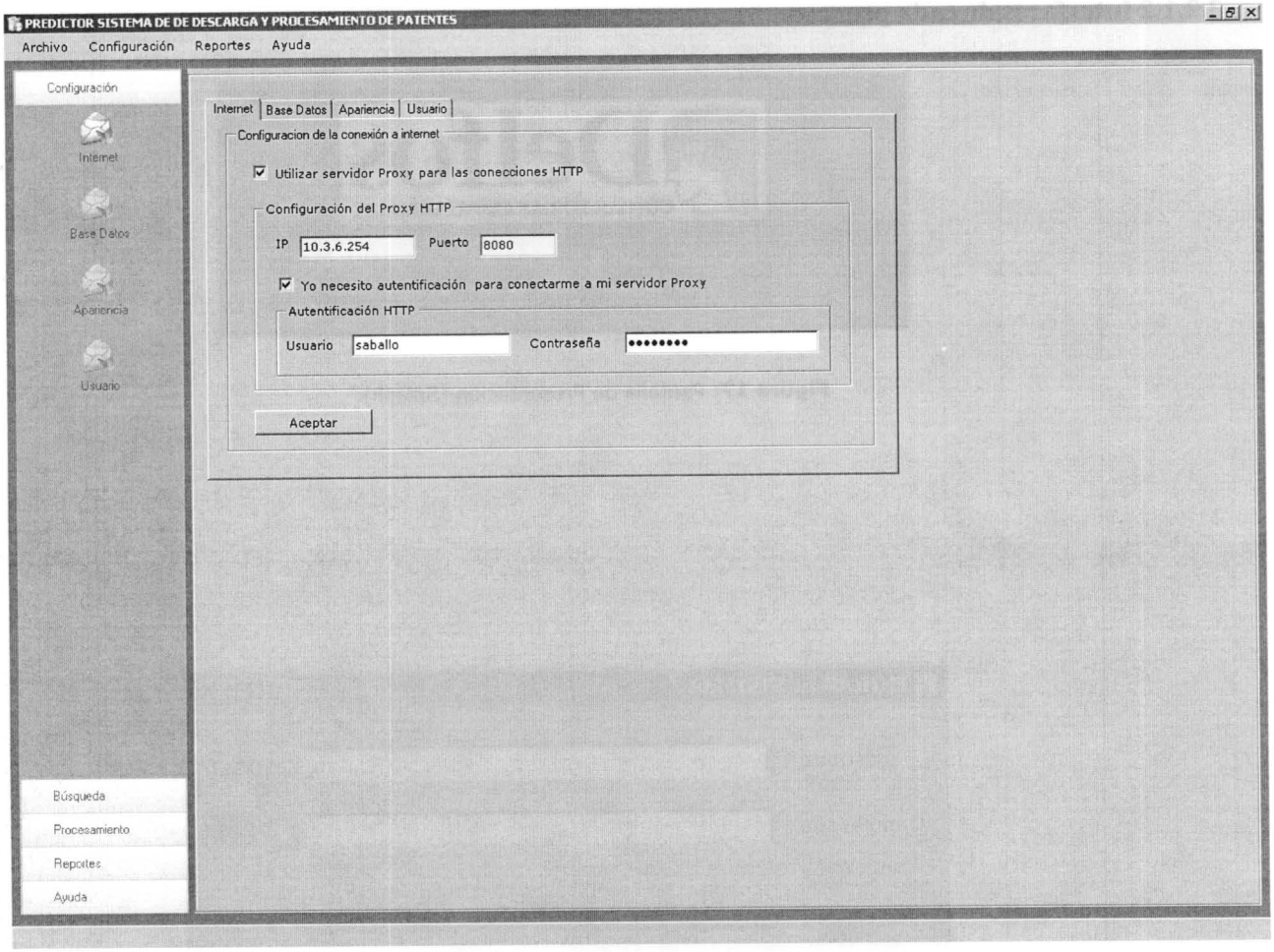


Figura 19: Pantalla de Configuración de la conexión a internet.

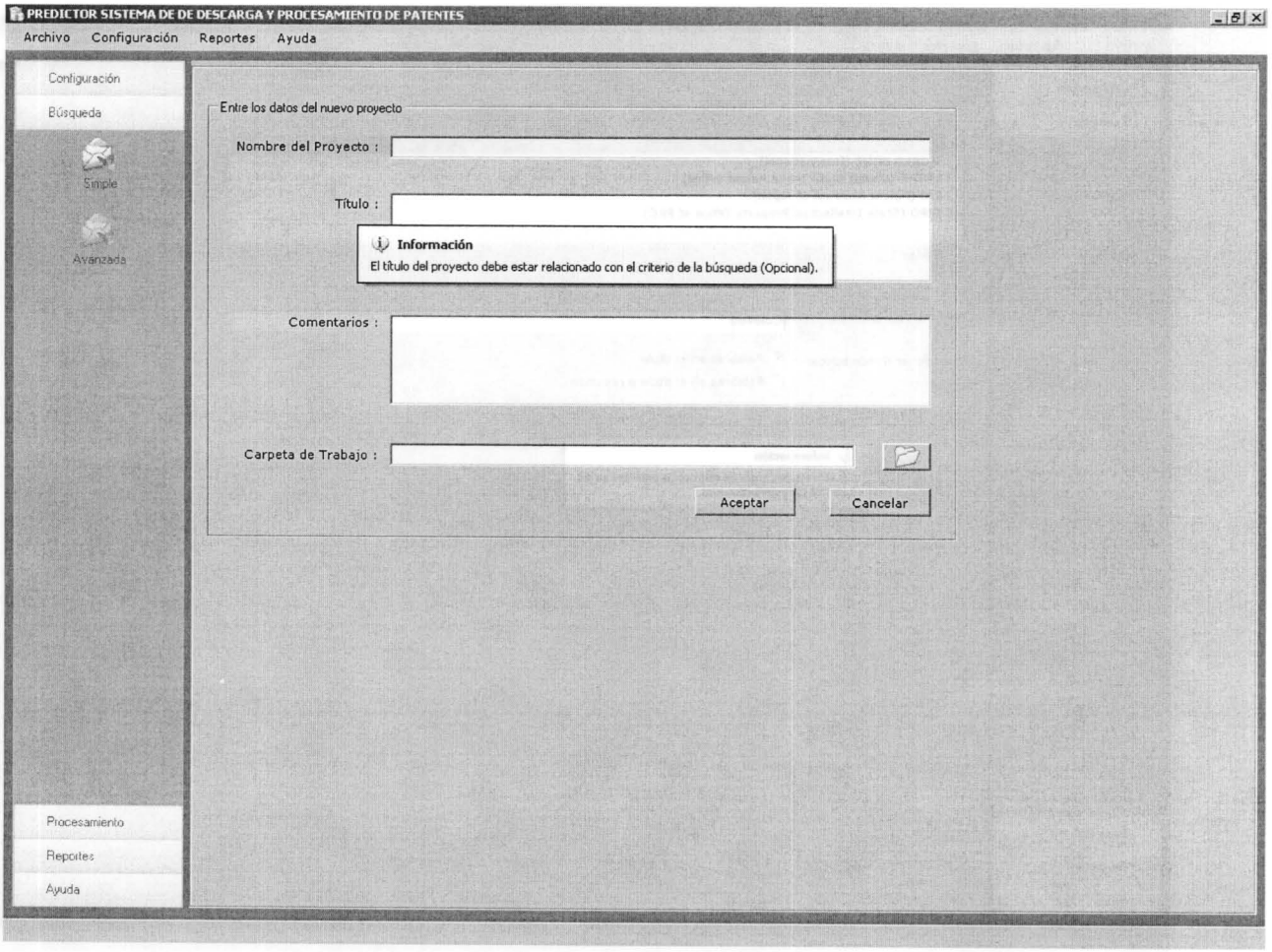


Figura 20: Pantalla de Datos de un Proyecto.

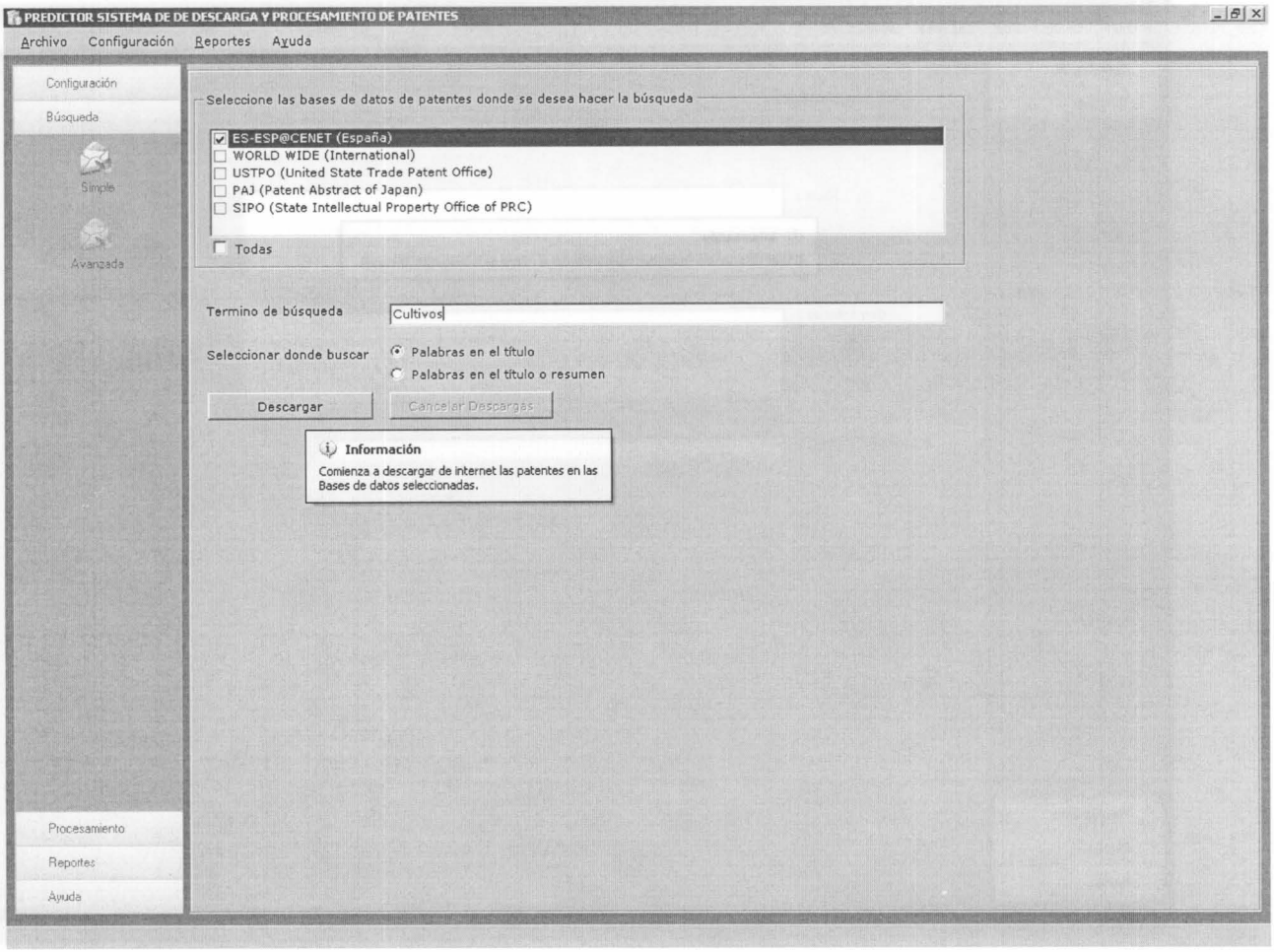


Figura 21: Pantalla de inicio de la Búsqueda Básica.

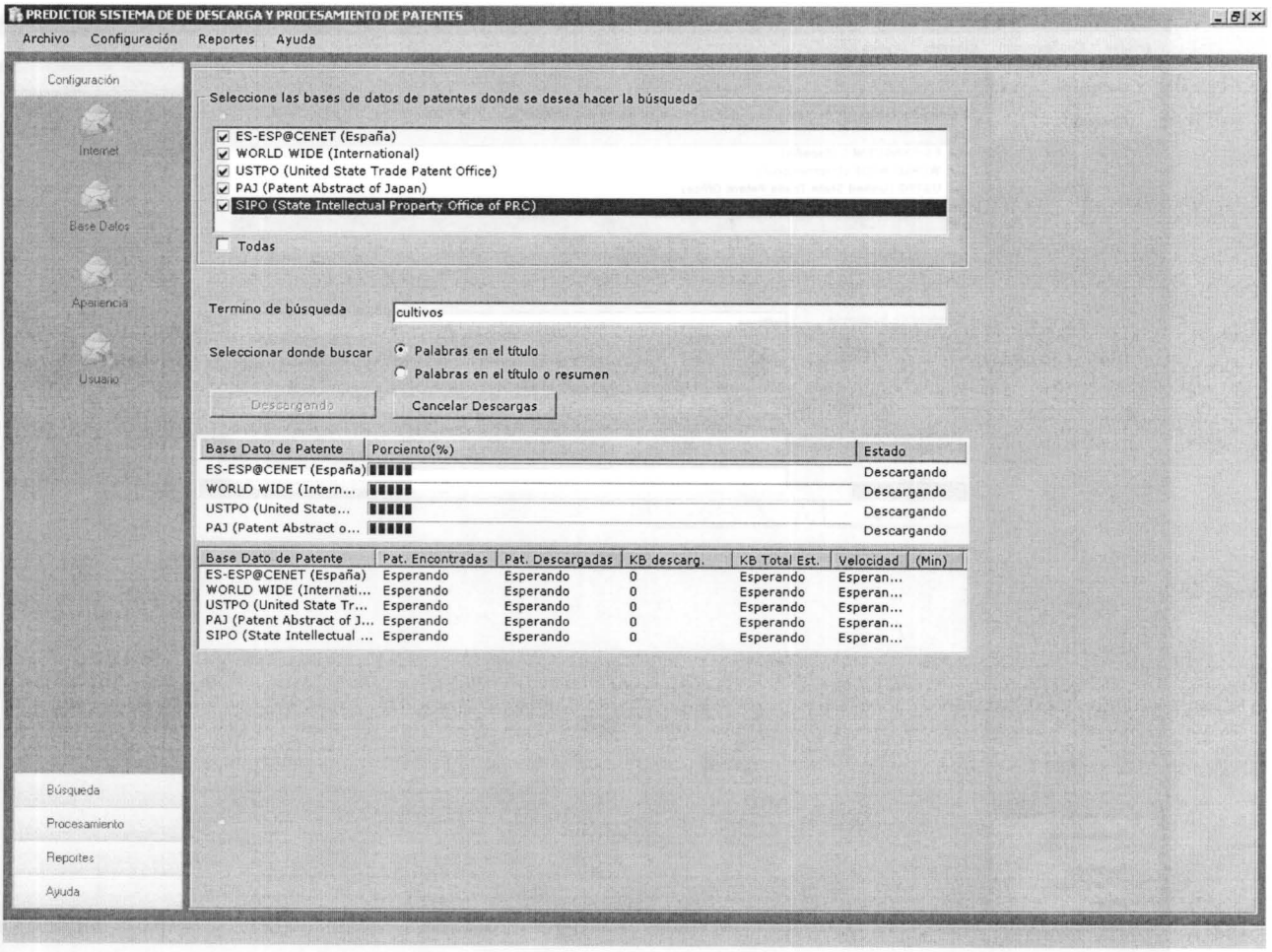


Figura 22: Pantalla de la Búsqueda Básica.

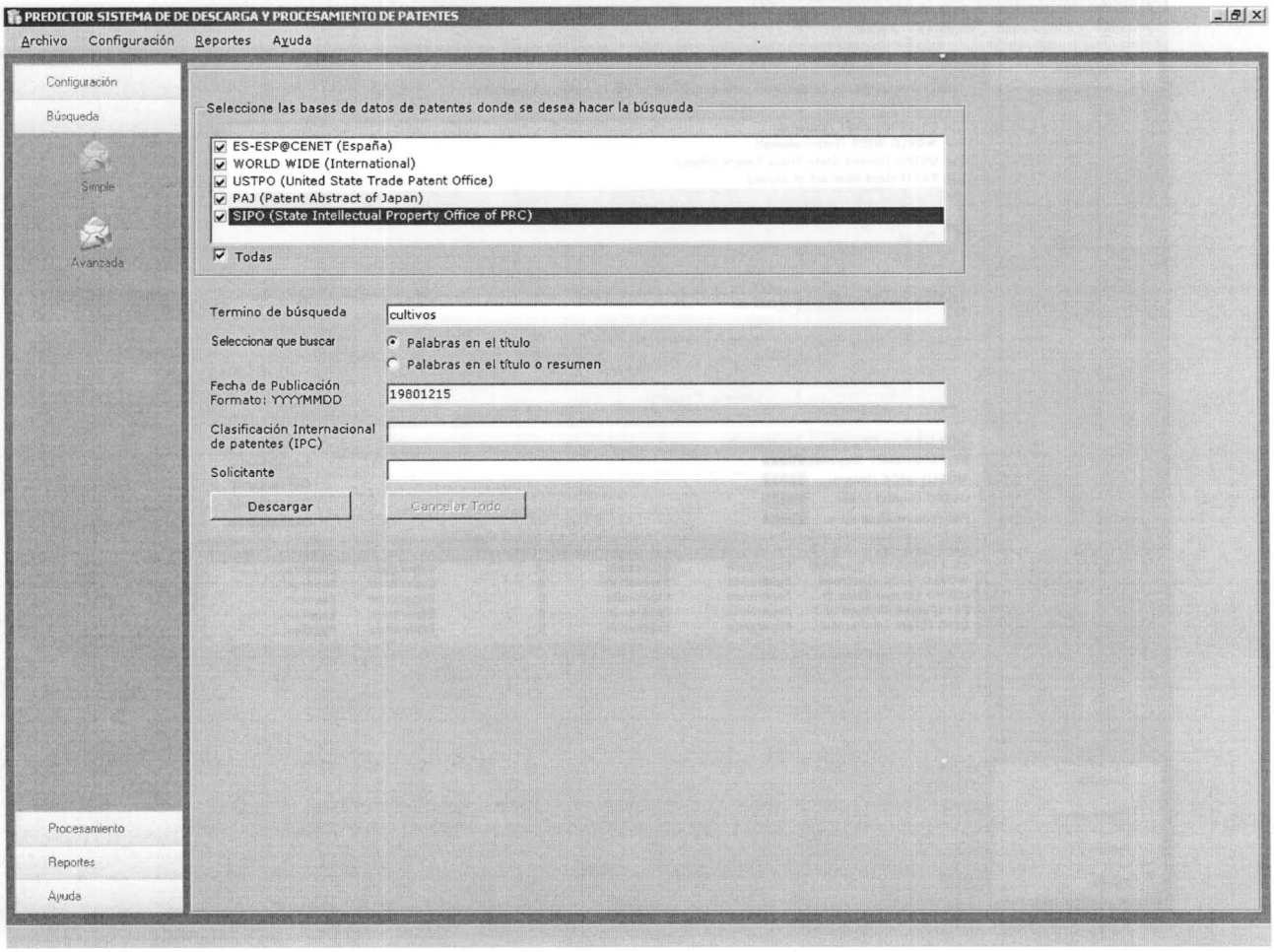


Figura 23: Pantalla de inicio de la Búsqueda Avanzada.

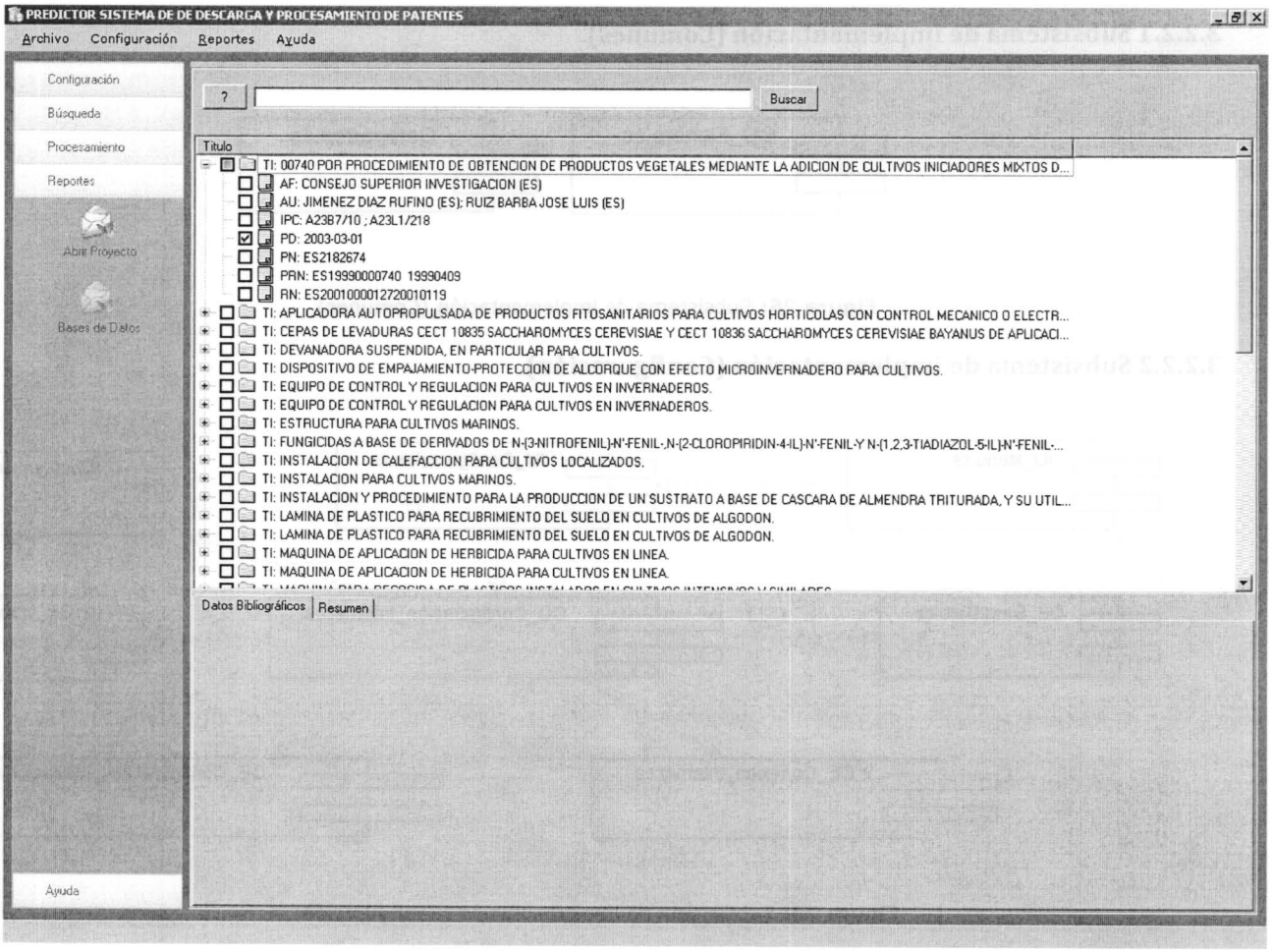


Figura 24: Pantalla de reportes de una Búsqueda.

3.2.2 Diagramas de componentes.

Los diagramas de componentes muestra los componentes con sus dependencias y los artefactos que estos implementan, estos diagramas son útiles para mostrar que componentes son necesarios para compilar un componente determinado.

3.2.2.1 Subsistema de implementación (Comunes).

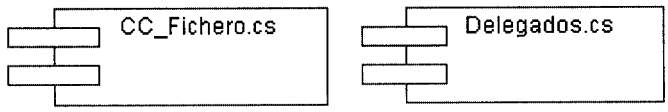


Figura 25: Subsistema de implementación (Comunes).

3.2.2.2 Subsistema de implementación (Configuración).

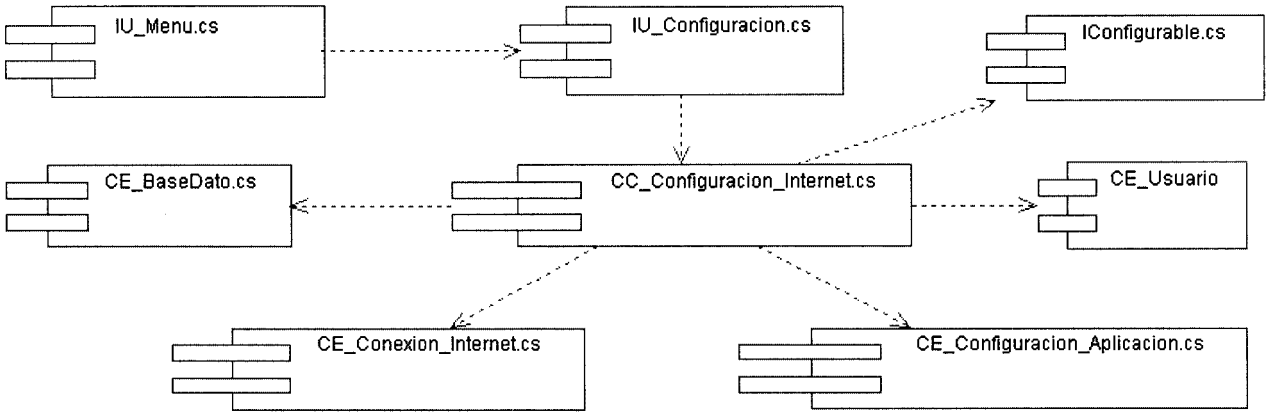


Figura 26: Subsistema de implementación (Configuración).

3.2.2.3 Subsistema de implementación (Descarga).

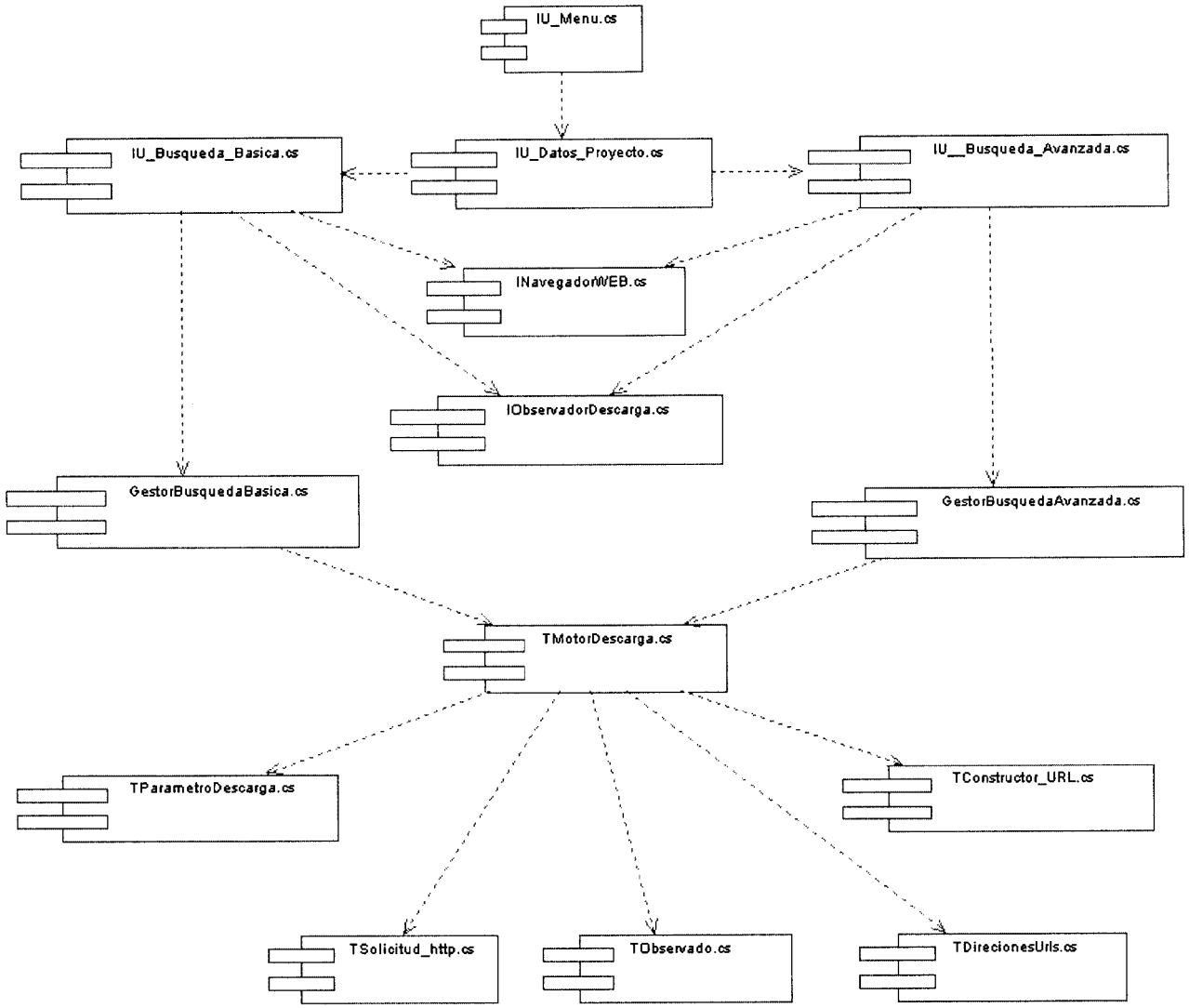


Figura 27: Subsistema de implementación (Descarga o búsqueda en Internet).

3.2.2.4 Subsistema de implementación (ProcesamientoHTML).

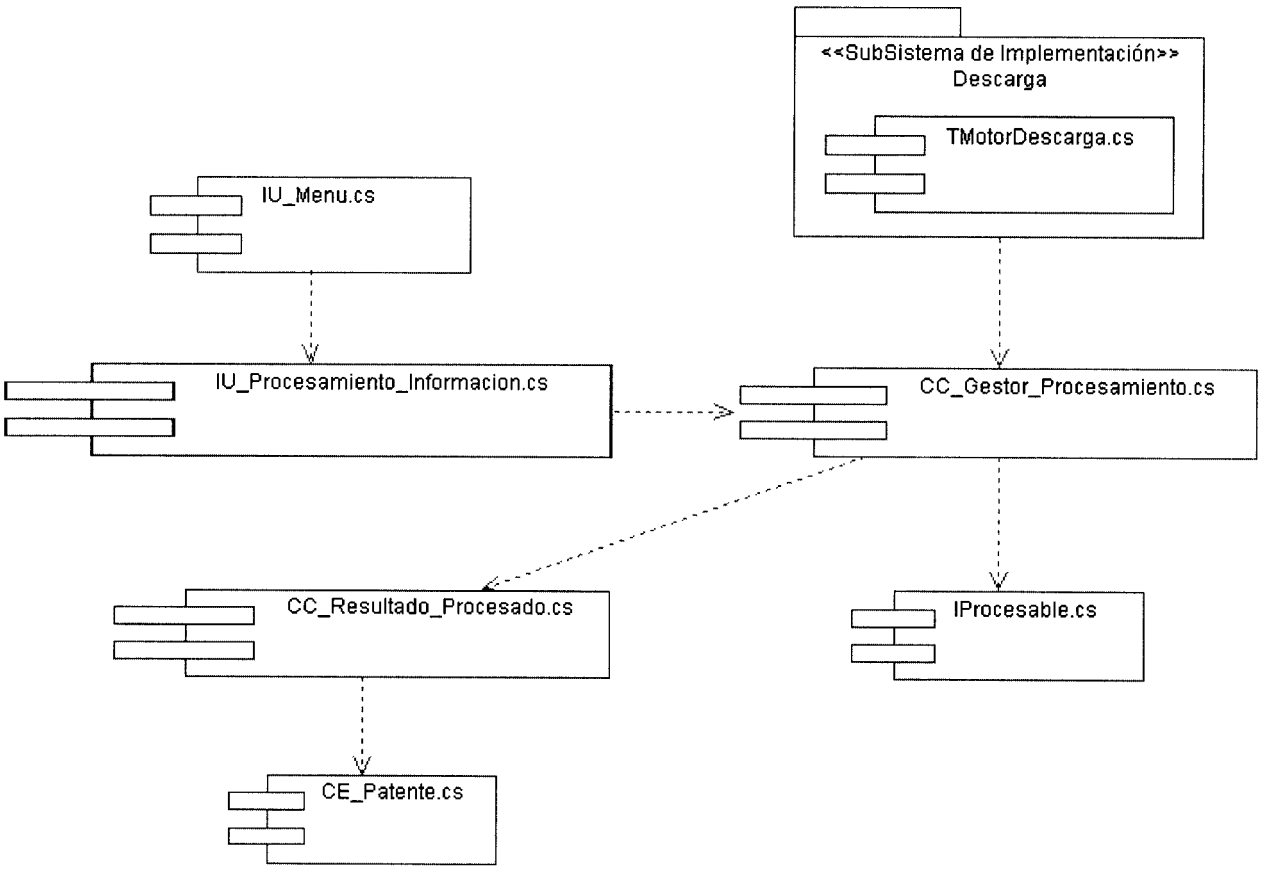


Figura 28: Subsistema de implementación (ProcesamientoHTML).

3.2.2.5 Subsistema de implementación (Reportes).

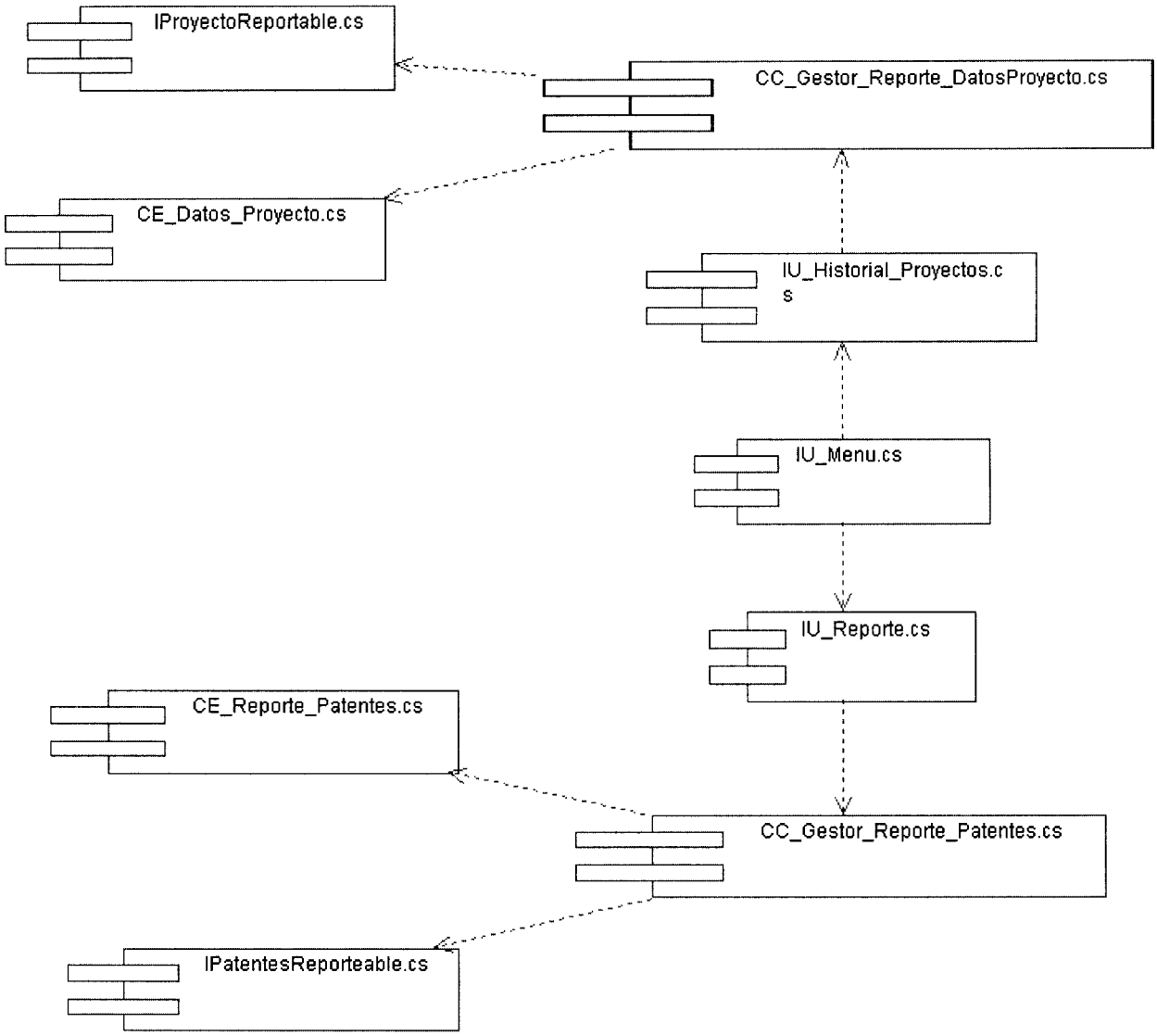


Figura 29: Subsistema de implementación (Reportes).

3.2.2.6 Subsistema de implementación (Autenticación).

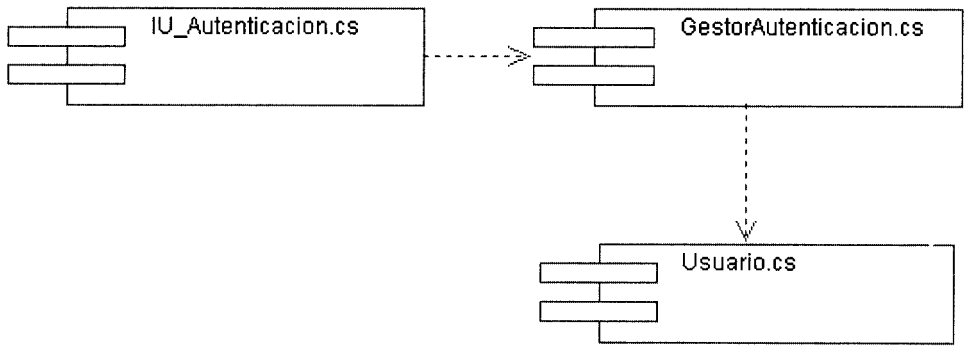


Figura 30: Subsistema de implementación (Autenticación).

CONCLUSIONES.

Con la realización del proyecto se arribaron a las siguientes conclusiones:

1. Se realizaron satisfactoriamente los artefactos de ingeniería planteados en los objetivos específicos para la realización del sistema.

2. Se utilizaron patrones de arquitectura y diseño que dieran lugar a un sistema robusto y fácil de extender, aspectos muy importantes en este tipo de sistemas de solicitudes web y procesamiento de la información descargada.

3. Se crea por primera vez en nuestro país un sistema automatizado para la gestión de patentes, con el cual se satisfacen las necesidades de automatización de la descarga y el procesamiento del HTML de incluido en las patentes, ganando en productividad y tiempo para invertir en otras tareas de análisis de información. Con estos resultados nuestros clientes podrán brindar mejor servicio de consultoría.

RECOMENDACIONES.

Para trabajos futuros:

1. Definir hilos independientes para las descargas de las URLs y la descarga de los contenidos de las patentes.

2. Concluir el módulo de persistencia para optimizar las búsquedas y utilizar estándares, como el XML en el resultado final, para potenciar la interoperabilidad con otros sistemas.

3. Continuar realizando investigaciones que permitan concluir la posibilidad de definir un algoritmo genérico para el procesamiento de la información proveniente de las distintas base de datos de patentes.

4. Migrar la solución hacia software libre o plataformas no propietarias.

GLOSARIO.

Abstracción: Las características esenciales de un objeto que lo distinguen de todos los demás tipos de objetos y proporcionan así fronteras conceptuales definidas con nitidez en relación con la perspectiva del observador, la abstracción es uno de los elementos fundamentales del modelo de objeto (Booch 1996).

Arquitectura: La estructura lógica y física de un sistema, formada por todas las decisiones de diseño estratégicas y tácticas aplicadas durante el desarrollo (Booch 1996).

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 (Kruchten 2000).

Atributo: Una parte de un objeto (Booch 1996).

Capa: Colección de categoría de clases o subsistemas al mismo nivel de abstracción (Booch 1996).

Clase: Representación abstracta de uno o mas objetos, los términos clase y tipo suelen ser (no siempre) equivalentes, son conceptos ligeramente diferentes, en el sentido que la clase describe la estructura y el comportamiento de uno o mas objeto(s) y un tipo solo define su comportamiento (Booch 1996).

Componente: Una parte física y reemplazable de un sistema que proporciona la realización de una o más interfaces (Ivar Jacobson 2000).

Diagrama: La representación gráfica de un conjunto de elementos, usualmente representado como un grafo conectado de vértices (elementos) y arcos (relaciones) (Ivar Jacobson 2000).

Fachada: Una fachada es un paquete estereotipado que no contiene más que referencias a elementos de modelos que pertenecen a otros paquetes. Utilizado para proporcionar una vista "publica" de algunos de los contenidos del paquete (Ivar Jacobson 2000).

Hilo: Un flujo de control ligero que puede ejecutarse concurrentemente con otro hilos en el mismo proceso (Ivar Jacobson 2000).

Interfaz: Una colección de operaciones que son utilizadas para especificar un servicio de una clase u otro componente (Ivar Jacobson 2000).

Instancia: Una manifestación concreta de una abstracción; una entidad sobre la que pueden aplicarse un conjunto de operaciones y que tiene un estado que almacena el efecto de las operaciones; sinónimo de objeto(Ivar Jacobson 2000).

Proceso: Un flujo de control pesado que puede ejecutarse concurrentemente con otros procesos (Ivar Jacobson 2000).

Sistema: Una colección de subsistemas organizados para llevar a cabo un propósito específico y descritos por un conjunto de modelos, posiblemente desde distintos puntos de vistas(Ivar Jacobson 2000).

Subsistema: Una agrupación de elementos, de los que algunos constituyen una especificación del comportamiento ofrecido por los otros elementos contenidos(Ivar Jacobson 2000).

BIBLIOGRAFÍA.

Booch, G. (1996). Análisis y Diseño Orientado a Objetos con Aplicaciones.

Castellanos., Y. (2007). Módulo de procesamiento HTML en documentos de patentes en el sistema Predictor.

Catalunya, U. P. d.

Cormen, T. H. (2002). Introduction to Algorithms.

Corporation, T. T. (2007).

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

Escorsa P, M. R. (2001). De la vigilancia tecnológica a la Inteligencia, Prentice Hall.

Española, R. A. (1992). Diccionario de la real academia española.

Garlan, D. (1996). Software Architecture: Perspectives on an Emerging Discipline, Prentice-Hall.

Hernández, R. G. (2006). La información de marcas como indicador de innovación Tecnológica. Habana.

Ivar Jacobson, G. B., James Rumbaugh (2000). "El proceso unificado de desarrollo de software."

Ivar Jacobson, M. G., and Patrik Jonsson (1997). Software Reuse: Architecture, Process and Organization for Business Success, Addison-Wesley.

Kruchten, P. (2000). The Rational Unified Process An Introduction, Second Edition, Addison Wesley.

Larman, C. (1999). UML y Patrones.

Madrid, U. P. d. (2003). Gang of Four Patterns.

Peter, W. (1981). The Ada Programming Language and Environment, unpublished draft.

Piñeiro, Y. (2007). Sistema de descarga y procesamiento automatizado de patentes. Rol Analista de Sistema.

Pressman, R. (2002). "Ingeniería de Software, Un enfoque práctico 5ta edición."

Rodríguez, M. (2006). Programación Orientada a Objetos.

Rumbaugh., J. (2000). UML Manual de Referencia.

Technology, S. (2007).

USIS, P. E. (1998). **3**.

Kay, P. M. (Septiembre 2003). "MSDN." from http://www.microsoft.com/spanish/msdn/comunidad/mtj.net/voices/art130.asp.

MicroPatent. (2005). from http://www.micropat.com/static/test.htm.

Microsoft. (2003). from http://support.microsoft.com/kb/307023/es.

Microsoft(MSDN). from http://msdn2.microsoft.com/es-es/library/z2kcy19k(VS.80).aspx.

PatentWizard. (2004). from http://www.patenthunter.com/.

Software, M. (2006). from http://www.matheo-software.com/.

Solutions, B. (2006). from http://www.bizcharts.com/index.html.

Thomson, C. from http://www.delphion.com/products/research/products-patlab.

W3C. from http://www.w3.org/Protocols/.

ANEXOS

Anexo 1 Fichero de configuración de PlugIns del sistema.

```
<PlugIns>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
          Name="OurControls.IU_Configuracion"></PlugIn>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
          Name="OurControls.IU_Busqueda_Avanzada"></PlugIn>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
          Name="OurControls.IU_Busqueda_Basica"></PlugIn>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
          Name="OurControls.IU_Reportes"></PlugIn>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
          Name="OurControls.IU_Datos_Proyecto"></PlugIn>
  <PlugIn Path="E:\ExtensibleUI\OurControls\bin\Debug\OurControls.dll"
        Name="OurControls.IU_Procesamiento"></PlugIn>
</PlugIns>
```

Anexo 2 Implementación del evento Load del Formulario Principal.

```
private void Form1_Load(object sender, System.EventArgs e)
{
  DataSet ds = new DataSet();
  ds.ReadXml("Config.xml");
  foreach (DataRow dr in ds.Tables["Plug-In"].Rows)
  {
    AddPlugIn(dr["Path"].ToString(),
    dr["Name"].ToString());
  }
}
```

Anexo 3 Implementación del método AddPlugIn

```
private void AddPlugIn(string Path, string ControlName)
{
        Assembly ControlLib;
        PlugIn NewPlugIn;
        // Cargar el ensamblado.
        ControlLib = Assembly.LoadFrom(Path);

        // Creando el Plugin.
        NewPlugIn = (PlugIn)ControlLib.CreateInstance(ControlName);
        NewPlugIn.Location = new System.Drawing.Point(250, 140);
        NewPlugIn.Dock = DockStyle.Fill;
        NewPlugIn.Visible = true;
           // Limpio el contenedor de los Plugin.
           GpbxContenedor_Controles.Controls.Clear();
        GpbxContenedor_Controles.Controls.Add(NewPlugIn);

        // Set up the ClickHandler
        NewPlugIn.Clicked += new PlugInLib.ClickHandler(Control_Clicked);
}
```

Anexo 4 Implementación de la clase base Plugin.

```
namespace PlugInLib
{
    public delegate void ClickHandler(object sender, EventArgs e);
    public class PlugIn : System.Windows.Forms.UserControl
    {
    // Garantiza que el evento clicked regrese al contenedor.
    public event ClickHandler Clicked;
    protected void DoClick(EventArgs e)
    {
      if (Clicked != null)
        Clicked(this, e);
    }
    // Proporciona un "Caption" que el contenedor puede mostrar.
    protected string m_Caption = "PlugIn";
    public string Caption
    {
      get {  return m_Caption; }
      set  { m_Caption = value; }
    }
    public override string ToString()
    {
      return m_Caption;
    }
    // Proporciona un metodo "TestFunction" que el contenedor puede llamar.
    public virtual void TestFunction() {}
  }
}
```

Anexo 5 Implementación de un plugIn Concreto.

```
namespace OurControls
{
  public class IU_Busqueda_Basica : PlugIn    // <---Cambiar la class Base a PlugIn
  {
    /// <summary>
    /// Variable del diseñador requerida.
    /// </summary>
    private System.ComponentModel.Container components = null;
    public IU_Busqueda_Basica ()
    {
      InitializeComponent();
    }
    protected override void Dispose(bool disposing)
    {
      if (disposing)
      {
        if (components != null)
        {
          components.Dispose();
        }
      }
      base.Dispose(disposing);
    }

   #region Component Designer generated code
    private void InitializeComponent()
    {
      this.Caption = " IU_Busqueda_Basica ";
```

```
        this.Name = " IU_Busqueda_Basica ";
        this.Click += new System.EventHandler(this. IU_Busqueda_Basica _Click);
    }
    #endregion

    // Redefinir los metodos para recivir llamadas desde la aplicacion principal.
    public override void TestFunction()
    {
        // Fue llamada desde la apliación principal.
    }
    // Enviar un click a la apliacion principal.
    private void IU_Busqueda_Basica _Click(object sender, System.EventArgs e)
    {
      DoClick(e);
    }
  }

}
```

Anexo 6 Subsistemas de Implementación.

Figura 31: Subsistemas de implementación.