BIPI: Desarrollo de un Buscador Inteligente para Internet basado en Agentes Móviles

Entrega Final

Licenciatura en Informática

Facultad de Informática

Universidad Nacional de La Plata

Tipo de Investigación:
Investigación Aplicada

Alumnos:
Claudio C. Confeggi
Sebastian Felix

Lugar de Trabajo:
LINTI

Director:
Javier Díaz

Noviembre de 2000

Agradecimientos

A pesar de que sabemos que esta tesis es producto de nuestro esfuerzo a lo largo de un año, no podemos dejar de reconocer que hubieron personas que nos ayudaron de una u otra manera e hicieron posible que llegara a su fin.

Queremos agradecer a quienes aportaron su grano de arena; nos apoyaron, estimularon y tuvieron paciencia en los momentos difíciles. Mil gracias a los que nos brindaron conocimientos y consejos para el desarrollo de este trabajo de grado.

Agradecemos explícitamente a Javier Díaz que como director nos guió y aconsejó en el transcurso de la tesis. A él quien nos empujó a inmiscuirnos en el mundo de los buscadores, y que a partir de allí nos llevó a la investigación de otros temas y tecnologías.

Muchísimas gracias a Daniel que nos presentó a don Latex, sin el cual la redacción y organización del presente documento hubiera sido muy engorrosa, y que nos curó de la adicción a don Word.

Finalmente, agradecemos conjuntamente a aquellas personitas que estuvieron siempre a nuestro lado y se *bancaron* tantas noches y fines de semana sin nuestro tiempo, dedicación y atención. Dos agradecimientos muy especiales a Karina y Carolina.

Claudio C. Confeggi
Sebastian Felix
La Plata, 26 de noviembre de 2000

Prefacio

A fines del año 1997, habiendo concluido la cursada de la carrera de Licenciatura en Informática, debimos orientar nuestros esfuerzos hacia la definición de un tema que se concretara en una tesis. Pero no fue, hasta fines de 1999, que realmente dedicamos nuestro tiempo a esta investigación. Comenzamos reconociendo problemáticas que podían ser satisfechas en el ámbito informático, que nos movilizaran a la adquisición de conocimientos teóricos y técnicos y que, al mismo tiempo, puedan ser profundizadas y resueltas con el desarrollo de una tesis.

Finalmente, basados en el incremento y exceso de información que existe en nuestros tiempos, y la imposibilidad de poder ubicarla correctamente, optamos por el tema de buscadores, presentado por Javier Díaz, que podían ayudar en la resolución de esta problemática en el contexto de Internet. Esto fue sólo el puntapié inicial y, una vez que ahondamos en el tema, vimos la posibilidad de profundizar en él, llevándonos a la diversificación del tópico inicial hacia los metabuscadores y los agentes móviles, ya que apoyarían en la resolución de la problemática y servirían como sustento conceptual para el desarrollo de la aplicación. Además, como el buscador inicialmente no tenía información, se pensó en brindarle la capacidad de aprender del resto de los buscadores.

Fue así que, en Diciembre de 1999, presentamos la propuesta para la resolución de esta problemática a través de un metabuscador denominado BIPI, acrónimo de ***B**uscador **I**nteligente **P**ara **I**nternet*, que se ve realizada en esta investigación aplicada y que, en los capítulos y apéndices siguientes, se define.

Ya en la presentación de la propuesta se vislumbraban cuestiones que debían ser resueltas a lo largo de la investigación y desarrollo de la tesis. Entre estas preguntas se encuentran cuestiones claves como: ¿Cómo definir el metabuscador y que funcionalidad debe incluir? ¿Qué características de los demás buscadores se deben adoptar? ¿Es el concepto de agente móvil el adecuado para el propósito del desarrollo? ¿Qué herramientas brindan la versatilidad necesaria para construir la aplicación? ¿Será útil el metabuscador obtenido y resolverá la problemática inicialmente planteada?

Índice General

1 Introducción **1**
1.1 Esquema de la tesis 2

I Conceptos preliminares **3**

2 Máquinas de búsqueda **5**
2.1 Partes 5
2.1.1 Spider 5
2.1.2 Catálogo 5
2.1.3 Software 6
2.2 Clasificación 6
2.2.1 Motores de búsqueda 6
2.2.2 Directorios 6
2.2.3 Híbridos 7
2.3 Agregado de URLs 7
2.4 Indexación 8
2.4.1 Obtención de URL 8
2.4.2 Recuperación 8
2.4.3 Almacenamiento 8
2.5 Evitando la indexación 9
2.6 Ranqueo y cálculo de relevancia 10
2.6.1 Metatags 11
2.6.2 Spamming 12
2.6.3 Enlaces 13
2.6.4 Frase de búsqueda 14
2.7 Ejemplo: Altavista 15
2.7.1 Partes 15
2.7.2 Clasificación 15
2.7.3 Agregado de URLs 15
2.7.4 Indexación 15
2.7.5 Ranqueo y cálculo de relevancia 15
2.7.6 Evitando la indexación 16
2.8 Comparación de máquinas de búsqueda 16

3 Agentes **19**
3.1 Historia de los agentes 19
3.2 Agentes 20
3.2.1 Características 21

3.2.2 Clasificación . . . 22
3.3 Teoría de agentes . . . 23
3.3.1 Actuación de los agentes . . . 23
3.3.2 Estructura de los agentes 24
3.3.3 Entorno . . . 24
3.4 Sistemas Multiagente . . . 26
3.5 Organización de los sistemas multiagente . . . 27
3.5.1 Las habilidades sociales de un agente individual . . . 27
3.5.2 Organización del sistema . . . 29
3.5.3 Cooperación entre agentes . . . 30
3.5.4 Comunicación entre agentes . . . 30
3.5.5 Dinámica de la organización . . . 31
3.5.6 Comunicar agentes . . . 32
3.5.7 Lenguajes de comunicación 32
3.5.8 ACL . . . 33
3.5.9 Ontologías . . . 34
3.6 Comunicación entre agentes mediante KQML 34
3.6.1 Requisitos de transporte . . . 35
3.6.2 Sintaxis KQML . . . 35
3.6.3 Semántica KQML 35
3.6.4 Performativas . . . 36
3.6.5 Parámetros de las performativas 37

4 Agentes en Internet 39
4.1 Internet y la World Wide Web . . . 39
4.2 Areas de aplicación de los agentes . . . 41
4.3 Tipos de agentes en Internet . . . 43
4.3.1 Agentes que colaboran . . . 44
4.3.2 Agentes que gestionan información . . . 46
4.3.3 Agentes asistentes personales (que adaptan interfaces) 49
4.3.4 Agentes móviles . . . 53
4.4 Comportamiento de los agentes en Internet 54

II BIPI: metabuscador 61

5 Arquitectura 63
5.1 Arquitectura 63
5.2 Cliente . . . 64
5.2.1 Perfiles de Clientes . . . 64
5.3 Servidor de Web . . . 66
5.3.1 Componente Común de Web . . . 66
5.3.2 Componente Administrador de Web 67
5.4 Aplicación . . . 67
5.4.1 Entornos . . . 67
5.4.2 Aplicación Local . . . 67
5.4.3 Aplicación Remota . . . 71
5.4.4 Middleware . . . 72
5.5 Sistema de Gestión de Base de Datos . . . 74

6 Interfaz y funcionalidad **77**
6.1 Home Page 77
6.2 Búsqueda 78
6.3 Agregar URL 79
6.4 Administración de la parametrización 80

III BIPI: motor de búsqueda **83**

7 Modelo de almacenamiento **85**
7.1 Descripción de modelo 85
7.2 Parametrización 86
7.3 Desglose de páginas 86
7.4 Gestión de descriptores 86
7.5 Páginas pendientes 87

8 Indexación de páginas **89**
8.1 Pasos en la indexación 89
8.1.1 Obtención de URL 89
8.1.2 Recuperación 89
8.1.3 Almacenamiento 90
8.2 Evitando la indexación 91
8.3 Reindexación 91
8.3.1 Criterio de Expiración de Páginas 92

9 Búsqueda **93**
9.1 Formato de Búsqueda Estandarizado 93
9.2 Factores en el ranqueo 95
9.2.1 Metatags 95
9.2.2 Spamming 96
9.2.3 Enlaces 96
9.3 Cálculo de relevancia 96
9.3.1 Cálculo completo 97
9.3.2 Búsqueda de subfrase 98
9.4 Presentación de resultados 99
9.5 Diagramas de flujo 99
9.5.1 Búsqueda en el metabuscador 100
9.5.2 Búsqueda en el motor de BIPI 100

IV Notas finales **103**

10 Perspectivas **105**
10.1 Expansión del metabuscador 105
10.1.1 Pasos técnicos de la expansión 105
10.2 Extensión de funcionalidad 105

11 Conclusiones **107**

V Apéndices **109**

A Java **111**
A.1 Introducción 111
A.2 Aplicaciones Java 112
A.3 Java en la red 112
A.4 Java en el servidor: Servlets 112
A.5 Acceso a bases de datos (JDBC) 112
A.6 Multitarea: threads 113

B JSP **115**
B.1 La tecnología JSP 115
B.1.1 Características 115
B.1.2 Beneficios de la tecnología JSP 115
B.2 Conceptos básicos 116
B.2.1 ¿Qué es una página JSP? 117
B.2.2 Aplicaciones web 117
B.2.3 Componentes y contenedores 117
B.2.4 Traducción y pasos de ejecución 117
B.2.5 Rasgos de las JSP 118

C Documentación técnica de la aplicación **119**
C.1 All Packages 119
C.2 Package ar.com.bipi.agents 119
C.3 Package ar.com.bipi.connection 127
C.4 Package ar.com.bipi.consoles 128
C.5 Package ar.com.bipi.servers 132
C.6 Package ar.com.bipi.system 135
C.7 Package ar.com.bipi.types 137

Capítulo 1

Introducción

El rápido crecimiento de Internet, junto con la incapacidad para documentar y catalogar toda la información existente en ella, lleva a que buscar datos específicos sobre la World Wide Web sea un esfuerzo que muchas veces supera los beneficios que proporciona la información encontrada.

Esta necesidad de búsqueda sugiere la construcción de interfaces sencillas y de fácil manejo para los usuarios. Interfaces que realicen una abstracción y oculten las diferentes técnicas necesarias para realizar determinadas tareas en Internet.

Los desarrolladores han entrado en esta tendencia, construyendo navegadores de web cada vez más completos y versátiles que asisten al usuario en su navegación por la World Wide Web. Sin embargo, esto no ha sido suficiente para permitir una rápida localización de la información, y se han desarrollado otras herramientas que facilitan realizar tareas específicas de búsqueda sobre la World Wide Web. La evolución en la creación de estas nuevas herramientas debería llevar a que las mismas se desarrollen pensando en las personas que las usarán, de manera de poder brindar un servicio personalizado en la búsqueda de la información.

Para la tarea de búsqueda de información en Internet se han desarrollado buscadores que proveen una gran variedad de métodos para encontrar la información requerida. La gran mayoría de estos buscadores se basan en la idea de recolectar información de la World Wide Web y posteriormente catalogarla de manera que, cuando un usuario ingresa al buscador en busca de información específica, revisa sus catálogos para devolver enlaces que direccionan a dicha información. La estaticidad de estos buscadores lleva al usuario a pensar que no obtuvo todos los resultados posibles en la búsqueda. Este pensamiento hace que muchos usuarios accedan a otros buscadores en busca de la información deseada; mas aún, hay muchos libros y artículos, al igual que helpers, que recomiendan esta estrategia de búsqueda a través de distintos buscadores.

Esta problemática llevó a que diseñáramos un modelo de `metabuscador` que finalmente fue plasmado en un aplicativo que lo soportara. Este desarrollo aprovecha algunos motores de búsqueda existentes, encargándoles la realización de búsquedas, para luego concentrar los resultados y así proveer un resultado consolidado más exacto. Pero nuestra investigación fue más allá y nos condujo a la construcción de un motor de búsqueda propietario que contiene ciertas características de los motores de búsqueda del mercado, llevando a que este motor forme parte del conjunto de motores del metabuscador. De esta forma decimos que BIPI es un metabuscador y un motor de búsqueda a la vez.

El motor de búsqueda debía contener un conjunto de páginas que pudiera entregar como resultado al metabuscador, por ello es que se lo hizo "inteligente". En realidad lo que hace este motor es anexar a su conocimiento aquellas páginas que le entrega el metabuscador, provenientes de resultados que entregan otros motores ante búsquedas realizadas sobre él.

Posteriormente este motor de búsqueda fue capaz de aprender a través del usuario, ya que éste puede proveerle páginas.

Los rasgos de la arquitectura del metabuscador (distribución, necesidad de comunicación y autonomía de las partes, paralelismo de acciones, independencia de plataforma, etc.) enfocó nuestra atención en las características de los agentes, más específicamente en los ***agentes móviles*** y su utilización en la conformación de ***sistemas multiagente***, que concuerdan con la arquitectura definida. La unión de estos elementos, junto con el ***lenguaje KQML***, propio de la comunicación de agentes, llevó a que consiguiéramos consolidar el modelo planificado.

1.1 Esquema de la tesis

La tesis contiene una división en partes. Luego de esta introducción comienza el desarrollo de la Parte I en que se trata una serie de temas cuyos conceptos son necesarios para la comprensión general del trabajo. En el Capítulo 2 se explica el concepto de máquina de búsqueda, las partes que contiene y su funcionamiento, cerrando con el ejemplo concreto de Altavista. Luego, en el Capítulo 3 se detallan las características de los agentes, su comportamiento y sus interrelaciones en sistemas multiagente, usado como introducción para el desarrollo del Capítulo 4 en donde se trata a los agentes en un ámbito más específico: Internet; y entre ellos a los agentes móviles.

Una vez introducidos los conceptos de la parte anterior, definimos el metabuscador en la Parte II. En el Capítulo 5 se detalla su definición arquitectónica; su división en capas, las interrelaciones entre ellas, los agentes que funcionan en su interior y las relaciones entre tales agentes. En el Capítulo 6, en cambio, nos enfocamos en explicar la funcionalidad que provee la interfaz de acceso al metabuscador, tanto la interfaz pública como la de administración.

Una vez descrito el metabuscador, fue necesario detallar el comportamiento del motor de búsqueda propietario, por ello, en la Parte III, se pueden encontrar los temas que hacen al mismo. En el Capítulo 7 se define cómo, el motor de BIPI, gestiona el almacenamiento del conocimiento; continuando en el Capítulo 8 con la descripción del método de obtención de información que incrementa la Base de Datos. Por último, en el Capítulo 9 se detallan las etapas que se suceden en la realización de una búsqueda, desde su solicitud hasta la generación de resultados.

Como cierre, en la Parte IV, se describen las perspectivas futuras del metabuscador y del motor de búsqueda, así como las conclusiones generales de la tesis.

Finalmente, podemos ubicar en la Parte V, todos aquellos temas que hacen a las herramientas utilizadas y definidas. En el Apéndice A se describen las características del lenguaje Java, en el Apéndice B se detalla la tecnología JSP y por último en el Apéndice C se incluye la reseña técnica de clases que componen el metabuscador de BIPI. Este último apéndice es de importancia en la tesis, pero no está incluido en el cuerpo principal debido a que en sí no provee datos de investigación, sino la concreción de la misma.

Parte I

Conceptos preliminares

Capítulo 2

Máquinas de búsqueda

Existen muchas acepciones para el término Máquina de Búsqueda, en inglés "Search Engine", dependiendo de la forma en que éstas agregan páginas en sus catálogos o índices. Todas realizan la misma función, mantienen la información necesaria de cada página en índices para su posterior recuperación ante una solicitud de búsqueda por parte del usuario.

Gran parte de las páginas que conforman la World Wide Web, aquellas escritas en HTML, se componen de elementos que permiten especificar ciertas características de ellas mismas. Entre estos elementos se encuentran el título, los metatags (palabras clave, descripción, etc.) y el texto de su contenido (cuerpo). Estos descriptores establecen la diferencia entre las distintas páginas y son utilizados por las máquinas de búsqueda en sus indexaciones y búsquedas.

La forma en que las distintas máquinas de búsqueda ranquean las páginas almacenadas en sus catálogos, a partir de los descriptores anteriormente mencionados, es un criterio de diferenciación. El ranking o relevancia que se da a cada página se determina en la especificación del modelo de parámetros de cada máquina de búsqueda. Es importante que este modelo sea mantenido dinámicamente a fin de que se adapte a las necesidades del momento y evitar el aprendizaje de su estructura de búsqueda por parte de los suscriptores.

2.1 Partes

Las máquinas de búsqueda tienen tres elementos principales que detallamos a continuación.

2.1.1 Spider

También llamado *crawler*. El spider visita una página web, la lee, y luego sigue los enlaces a otras páginas. Esto es lo que uno a veces llama que un sitio sea "spidered" o "crawled". El spider regresa al sitio indexado a intervalos regulares, cada uno o dos meses, en búsqueda de cambios.

2.1.2 Catálogo

Todo lo que el spider encuentra va dentro de la segunda parte de una máquina de búsqueda: el índice. El índice, algunas veces llamado catálogo, es como un libro gigante que contiene una copia de cada página web que el spider encuentra. Si una página web cambia, entonces este libro es actualizado con nueva información.

A veces puede tomar tiempo para que las nuevas páginas o los cambios que el spider encuentra sean agregados al índice. Por ello, una página web puede ser "spidered" pero aún no "indexada". Hasta que es indexada, o sea, agregada al catálogo, esta página no estará disponible para aquellas búsquedas realizadas con la máquina de búsqueda.

2.1.3 Software

Este es el programa que recorre las millones de páginas almacenadas en el índice para encontrar coincidencias a partir de una búsqueda, y rankear los resultados de acuerdo a lo que él cree que es lo más relevante.

2.2 Clasificación

Los buscadores se dividen principalmente en dos tipos distintos: los *Motores de búsqueda* y los *Directorios*. Ejemplos clásicos de ambos son HotBot y Yahoo, respectivamente.

La diferencia principal entre ellos radica en el modo en que recolectan e indexan los sitios en sus Bases de Datos.

2.2.1 Motores de búsqueda

Poseen un programa robot denominado ***spider*** o ***crawler*** que recorre la Web leyendo los distintos sitios publicados y extrayendo de ellos la información necesaria para armar sus índices. En teoría, estos robots, recorren la Web siguiendo los enlaces que van encontrando en su camino y, de esta forma, leen e indexan todas las páginas por las cuales pasan (no siempre indexan una página por la que pasan, ver Sección 2.5).

En la práctica, sin embargo, las posibilidades de que un robot acceda a un sitio determinado, simplemente siguiendo enlaces, son bastante reducidas. Sumado a ello tampoco les resulta rentable mantener grandes Base de Datos y los últimos reportes al respecto indican que, en el mejor de los casos, sólo cubren hasta un 16% de las páginas publicadas en la red.

Debido a los grandes volúmenes de información, sin clasificación previa, que recolectan estos motores, pueden producirse búsquedas en las que los resultados no se encuentran relacionados con el tópico de búsqueda, ya que el tópico puede estar fuera de contexto.

Si se cambia el contenido de una página ya indexada, eventualmente el spider encontrará los cambios cuando revisite la página, pudiendo afectar el ranqueo de la misma (en base al título, cuerpo y otros elementos). Debido a este comportamiento se dice que son máquinas dinámicas.

2.2.2 Directorios

Ordenan los sitios de sus Bases de Datos en un sistema de categorías. Algunos permiten total libertad al usuario respecto del tipo de categoría en la cual listar su sitio y otros tienen un proceso de revisión editorial previo, por lo que la información provista puede sufrir modificaciones.

Los directorios no recorren las páginas con un robot para extraer información de ella. El único material que utilizan para armar sus índices es el que el usuario provee al momento de la registración y, eventualmente, puede sufrir alguna modificación en los directorios que revisan los sitios antes de incluirlos en sus listados. Cuando se hacen los envíos a los directorios, entonces, conviene tener en cuenta una buena escritura de la descripción y de los títulos, sin importar en estos casos que tan bien hayamos diseñado el texto de nuestra página, o sus metatags.

Otro punto a tener en cuenta aquí es el de la correcta elección de la categoría en la cual queremos agregar nuestro sitio. Debido a sus características, los directorios pueden ser consultados no solamente mediante el formulario de búsqueda, sino también navegando a través de sus categorías. Conviene ser lo más preciso posible, a efectos de facilitar al navegante el acceso a nuestra información.

Esta metodología de categorización de páginas lleva a que los directorios frecuentemente provean muchos más resultados acertados que los motores de búsqueda.

Al contrario de las máquinas de búsqueda puras, un cambio en el contenido de una página ya indexada, tendrá pocas probabilidades de que sea reflejado en el catálogo del directorio, llevando a que los resultados de búsquedas sean inexactos por contenido o categoría (los sitios buenos, con buenos contenidos, ocasionalmente serán reindexados). De esta forma el ranqueo de una página no se actualizará cuando ésta sea modificada, por lo que se denomina a los directorios máquinas con clasificación estática.

2.2.3 Híbridos

Como el nombre lo indica, este tipo de máquina de búsqueda, combina características de los dos tipos anteriores. Por lo tanto son motores de búsqueda que trabajan conjuntamente con directorios. Usualmente son sitios revisados y rankeados.

El hecho de llegar a ser incluido en una máquina de búsqueda es usualmente una combinación de suerte y calidad. Por esos motivos, este tipo de máquina de búsqueda, asocia la potencia de búsqueda de un spider de una máquina de búsqueda pura, con la posibilidad de agregado manual de URLs y categorización de los directorios; para que todos los sitios tengan la posibilidad de ser listados.

Por ello, con este tipo de buscador, no debemos esperar que el spider pase a visitarnos, sino que, por el contrario, debemos "pedirle" al robot que visite nuestro sitio. Para lograr esto han puesto a disposición de los usuarios la posibilidad de informarles cual sitio debe ser visitado mediante el agregado manual de URLs (ver Sección 2.3).

2.3 Agregado de URLs

Esta opción se encuentra presente en la mayoría de los buscadores en una página llamada "Agregar URL". En general, sólo debe indicarse la URL a visitar, dado que esta es toda la información que necesita el robot. En algunos casos, será necesario consignar la dirección de correo electrónico, por si surgen necesidades de comunicación.

Esta es una característica muy importante de ciertos buscadores ya que permite que un usuario le indique al buscador en qué momento debe indexar la página. Cuando un usuario hace uso de esta función, el crawler inmediatamente buscará la página y la llevará a las máquinas de indexación para que la agreguen al índice. De esta forma, en vez de esperar que el spider ubique esta página al azar, quizá luego de muchos meses de procesamiento, el usuario puede controlar cuando una página es indexada.

Debemos recordar que no se requiere ninguna autorización en especial para agregar una página, a no ser que se trate de un directorio (como Yahoo!) en el cual debemos probar que somos quien decimos ser. Aquí simplemente sugerimos páginas para que el crawler busque. Él traerá cualquier texto que encuentre en tal dirección. Todo lo que sabe lo encontrará en esa página, no de lo que nosotros le digamos.

Algunos buscadores proveen además la función para la eliminación de URLs de páginas no existentes. Si nosotros llegamos a una página que no está disponible veremos el "Error 404", que significa que no existe tal página -no que el crawler no pudo acceder a la página por algún problema transitorio, sino que la página con tal URL no existe en el servidor.

Si una consulta nos lleva ocasionalmente a una página no existente deberíamos hacer uso de la función "Agregar URL" y colocar la dirección de la página que produjo el error. Si la página está en el índice, el buscador la quitará. Lo que estamos haciendo es forzar al indexador a revisitar la URL y, al descubrir que no existe, la elimina del catálogo.

2.4 Indexación

Este proceso lo realiza uno de los tres componentes de las máquinas de búsqueda: el spider. Consiste en la búsqueda, recuperación y almacenamiento de la página en el catálogo del buscador; por esta razón, las máquinas de búsqueda puras o híbridas son las únicas que realizan el proceso completo. Los directorios simplemente van a hacer un agregado de URL, ya que el índice es mantenido por los usuarios.

2.4.1 Obtención de URL

Con este paso se inicia el proceso de indexación. Consiste en la obtención de la URL de la página a indexar. Esta URL proviene de lo que llamaríamos la lista de "URLs pendientes". La lista es actualizada de diferentes maneras dependiendo del tipo de máquina de búsqueda. Si se trata de un motor que posee un spider que se encarga de la "visita" de sitios, éste recolecta URLs y las almacena en la lista. Si por el contrario, se trata de un directorio, algún usuario inserta la URL a ser recuperada. Este último comportamiento también se da en los híbridos a través de la funcionalidad "Agregar URL" (ver Sección 2.3).

2.4.2 Recuperación

Una vez que se obtiene una URL, el componente software se encarga de recuperar el contenido de la página que la URL indica. Este contenido sigue su flujo, a la etapa siguiente, sin modificaciones.

Esta recuperación se produce en las máquinas de búsqueda puras, en los híbridos y en los directorios que revisan los sitios antes de incluirlos en sus índices.

2.4.3 Almacenamiento

Este proceso compone la indexación en sí (o sea, es la indexación propiamente dicha) y se alimenta de la página recuperada en la etapa anterior. El almacenamiento puede producirse de diferentes formas dependiendo del buscador. En general existen dos métodos de almacenamiento.

Almacenamiento plano

Mediante este método la página se almacena como texto plano, o sea que no sufre modificaciones y se introduce "as is" (tal cual) en el catálogo (base de datos, sistema de directorios, etc.).

Las ventajas de este método son:

- **Se conserva el concepto de "unidad de página"**, ya que un desglose de la misma estaría basado en una subjetividad establecida por el programador y la estructura del medio de almacenamiento.
- **Se evita la pérdida de tiempo** consumido por un procesamiento adicional previo al almacenamiento de la página.

Desglose

La página recuperada se divide en diferentes partes. Estas partes por lo general son: el título, el cuerpo, los enlaces, los metatags, etc. Las mismas son almacenadas en forma separada en una estructura establecida en el medio de almacenamiento.

La ventajas de este método son:

- La **facilidad de recuperación** de una página por un ítem en particular (p.ej: el título, un metatag).
- Se obtiene una **ganancia de tiempo en el momento de ranqueo**, por el hecho de que la página ya se encuentra dividida. El tiempo se consume en el momento del almacenamiento (esto no influirá en el tiempo del usuario y se produce sólo una vez por página).

Las desventajas derivan de oposiciones a las ventajas del método de almacenamiento plano; se pierde el concepto de unidad de página y se consume tiempo en la división por partes de la página.

2.5 Evitando la indexación

Podemos preferir que algunas o todas nuestras páginas no sean indexadas. Tal vez nuestro sitio está aún bajo construcción y queremos obtener comentarios de algunas personas, pero no estamos listos para contarle al mundo entero que nuestras páginas están "vivas".

Muchos crawlers obedecen al ***Estándar de Exclusión de Robot***[1], que especifica un protocolo para los administradores de sitios de forma tal que pueden dirigir o establecer acciones a los llamados "robots" que recorren la web e indexan sitios. Podemos hacer esto con un pequeño archivo de texto al que debemos llamar "robots.txt" -un archivo que contiene las instrucciones que el administrador establece para los robots visitantes. Podemos excluir un crawler en particular o todos los crawlers (que siguen el estándar) de nuestro sitio entero, de directorios en particular o de archivos en particular.

Este archivo necesita ser ubicado en el nivel más alto del espacio de documentos del servidor, por lo tanto, si nuestro sitio está situado en un ISP, necesitaremos contactar al webmaster del ISP por ayuda sobre este tema.

Para excluir un sitio de todos los crawlers, debemos crear un archivo llamado "robots.txt" que establezca:

```
User-agents: *  # Dirigido a todos los robots
Disallow: /
```

Para excluir sólo el crawler de Altavista (conocido como "Scooter") el archivo debería contener el siguiente texto:

```
User-agents: scooter  # Evita el indexador de páginas de Altavista
Disallow: /
```

Para excluir los archivos multimediales del indexador de imágenes, MP3/audio y clips de video de Altavista:

```
User-agents: vscooter  # Evita el indexador multimedial de Altavista
Disallow: /
```

Para limitar las exclusión a un directorio en particular o un archivo, debemos poner la dirección después de **Disallow:**. Por ejemplo:

```
Disallow: /imagenes/personal/
```

[1] El estándar reside en http://info.webcrawler.com/mak/projects/robots/exclusion.html

También podemos usar metatags, sin la cooperación de nuestro ISP, para excluir a páginas en particular de los crawlers. Hay que tener en cuenta que no todos los robots observan estos metatags. Por ejemplo, si agregamos la siguiente línea a una de nuestras páginas web, el crawler no agregará esta página al índice y no seguirá los enlaces que encuentre en ella.

```
<meta name="robots" content="noindex, nofollow">
```

Los crawlers que reconocen esta exclusión por metatag aceptan las siguientes opciones:

- NOINDEX: previene que cualquier elemento de la página sea indexado.
- NOFOLLOW: previene al crawler de seguir los enlaces en la página y de indexar las páginas enlazadas.
- NOIMAGEINDEX: evita que las imágenes en la página sean indexadas, pero el texto en la página puede aún ser indexado.

El hecho de excluir a los crawlers de archivos específicos puede darnos un modo de mantener cierto control sobre la experiencia de los visitantes en nuestro sitio. Por ejemplo, si quisiéramos realizar una competencia de preguntas y respuestas, podríamos hacer que el robot excluya las páginas con las respuestas, de tal forma que la gente sea incapaz de encontrar esas páginas al azar -ellos sólo encontrarían páginas con preguntas.

Un último punto: no todos los robots se adhieren al "Estándar de Exclusión de Robot", por lo tanto, si queremos mantener realmente a nuestras páginas fuera del alcance de las máquinas de búsqueda, deberíamos considerar el hecho de establecer algún tipo de protección mediante clave.

2.6 Ranqueo y cálculo de relevancia

Hay ciento de millones de páginas web, por lo que casi cualquier consulta posiblemente tenga un alto número de concordancias. Para que los resultados de búsqueda sean útiles, las máquinas de búsqueda deben ranquear más alto aquellas páginas que posiblemente tienen información más relevante. La fórmula para hacer esto es mantenida en secreto por cada buscador y es sujeto de continua puesta a punto. Pero un entendimiento de los ingredientes principales puede ayudarle a uno a construir páginas que serán valuadas por las máquinas de búsqueda y entonces encontradas por la gente que las usa.

El contenido cuenta; el contenido cercano a lo más alto de la página cuenta más que el contenido al final. En particular el título HTML y las primeras pocas líneas de texto son lo más importante de las páginas. Si las palabras y frases que concuerdan en una consulta aparecen en el título o las primeras líneas del texto de las páginas, las chances son muy grandes de que la página aparezca alto en la lista de resultados.

Supongamos que quisiéramos poner un curriculum vitae en la Web. Hay que mantener la siguiente regla en mente: "no hay que poner el nombre del titular del curriculum al principio de la página". No queremos que sea encontrado por personas que ya conocen al titular, sino que queremos que sea encontrado por aquella gente que nunca escuchó hablar de él. Por lo tanto, no hay que desperdiciar letras en el título con el nombre. Las primeras palabras deberían ser "curriculum vitae". Después de esto, deberían estar listadas todas las cualificaciones y los tipos de trabajo que el titular realizó y busca. Debemos poner el mismo tipo de elementos en las primeras líneas del texto, que serán la descripción por defecto en la lista de resultados y también será importante en el posicionamiento del ranqueo.

Por ejemplo Altavista no premia a las páginas web la repetición inútil, solamente cuenta cada palabra dos veces.

2.6.1 Metatags

Los metatags son breves instrucciones que se pueden incluir en el encabezamiento de las páginas web. Contienen información referida al documento HTML pero no se presentan en la pantalla del navegador. De acuerdo con el nombre que tenga asignado, el contenido de dicha información servirá a distintos propósitos.

Estos descriptores mantienen la siguiente sintaxis:

```
<META NAME="nombreTag" CONTENT="contenido">
```

Se debe tener en cuenta que **nombreTag** será alguna palabra clave entre keywords, robots, refresh, title, description, etc. y **contenido** deberá corresponderse con la palabra clave establecida en la parte **NAME**. Podemos colocar un metatag con cada una de las palabras claves nombradas anteriormente, pero todos ellos deben ubicarse en el encabezamiento de la página (entre <HEAD> y </HEAD>).

Muchos webmasters enfocan la atención en los ***metatags*** y pierden de vista la importancia del título HTML y el texto plano contenido en la página, que para propósitos de ranqueo son aún muy importantes. Los metatags no tienen precedencia. Ellos son texto también. Entonces si la página esta pobremente diseñada, con varias palabras asociadas con gráficos, y con un título insignificante, la descripción de los metatags no ayudará en el ranqueo. *Es más importante hacer hincapié en el título y el contenido visible.* La utilización de los metatags es recomendable y, aunque de por sí no aseguran posicionamiento alguno en los resultados de búsqueda, un trabajo que combine los metatags con un título adecuado y un texto descriptivo, nos permitirán mejorar el que obtendríamos con el mismo sitio si no los utilizáramos.

¿Por qué no se les da a los metatags precedencia?. Porque a través de ellos se tienen chances de abusar y de hacer spamming. Lo que le importa a los usuarios de los buscadores es el contenido actual que es visible en las páginas web, no las notas orientadas al marketing que han sido agregadas en los metatags. Y el texto que aparece en el título y en las primeras pocas líneas estará posiblemente más relacionado con el tópico de la página.

Básicamente, los metatags son parches que ayudan a aquellas páginas que no establecen claramente y de frente el tópico del cual tratan. Si uno lo hace bien desde el principio, no necesitará de ningún modo de los metatags; y obtendrá muchos mejores resultados, en base al tráfico de las máquinas de búsqueda, de este modo.

Cabe tener en cuenta que algunos buscadores los ignoran y otros los leen solamente como parte del contenido de una página sin darle una relevancia particular (ver Tabla 2.1 en la Sección 2.8).

A continuación detallaremos la utilización de las palabras clave de los metatags que pueden ser utilizados por distintas máquinas de búsqueda al momento de la indexación.

Keywords

El metatag que contenga esta palabra clave en su parte NAME nos permitirá incluir en su parte CONTENT las palabras claves mediante las cuales deseamos que nuestro sitio sea hallado. Ciertos motores de búsqueda, cuando lean nuestras páginas, indexarán este contenido como si fuera parte del cuerpo de la página.

"Keywords" no es un nombre apropiado. Muchos webmasters malinterpretan el propósito de tales metatags y presumen que los buscadores actúan como una base de datos y que esas palabras son las únicas palabras de una página que importan en una búsqueda. Por el contrario, los buscadores indexan cada palabra de cada página, y cada palabra (y el orden en el cual aparece) es importante. El propósito de este metatag es simplemente permitir agregar

sinónimos[2] que describen qué se puede encontrar en la página pero que realmente no aparece en ella. Uno de los mejores usos para el metatag *keywords* es para traducciones lingüísticas de las principales palabras de la página, por lo tanto, por ejemplo, alguien buscando en francés encontrará la página.

Muchos webmasters piensan que usando el metatag *keywords* van a obtener alguna ventaja en el ranqueo por el solo hecho de que sus páginas tienen muy poco contenido de texto, sólo a efectos de llamar la atención. Pero aquellas palabras son subvaluadas con respecto a cualquier otra palabra en el texto principal de la página. No hay ninguna clave acerca de esto. Simplemente se está agregando unas pocas palabras más en la página en un lugar que no es visible.

Por ejemplo, si nuestro sitio se dedica a la "venta de pizzas a domicilio", podríamos incluir las siguientes palabras clave: pizza, pizzas, venta domiciliaria, delivery, comidas, comidas rápidas. Todo aquello que pueda estar buscando un posible visitante interesado en nuestro producto o servicio. No se trata, sin embargo, de hacer una lista enorme, sólo debemos incluir las palabras relativas necesarias.

La sintaxis adecuada sería:

```
<META NAME="keywords" CONTENT="pizza, pizzas, delivery, ...">
```

Es importante remarcar que no debemos repetir varias veces la misma palabra clave (sumando las de la parte CONTENT y el cuerpo de la página) ya que los buscadores penalizan esta práctica (Ver Spamming en Subsección 2.6.2).

Description

Por ejemplo, el metatag *description* permite ingresar unas pocas líneas que describen la página. Con esta palabra clave podemos controlar la forma en que aparecerá nuestro enlace en los resultados de búsqueda, esto es, estas palabras aparecerán como la descripción de las páginas en la lista de resultados en vez de las primeras líneas del texto del cuerpo.

Siguiendo el ejemplo anterior, la sintaxis sería:

```
<META NAME="description" CONTENT="Ordene sus pizzas por teléfono">
```

Robots

Con esta palabra clave podemos controlar el acceso de los indexadores a páginas específicas dentro de nuestro sitio (ver información detallada en Sección 2.5).

2.6.2 Spamming

La creación de diversas barreras para evitar la indexación de ciertas páginas, se debe al mal comportamiento de un conjunto de webmasters que han tratado de engañar a las máquinas de búsqueda con el objetivo de situar "sus" páginas más alto en las listas de resultados e incluirlas como resultados en consultas sin coincidencias reales. Este comportamiento se conoce como ***spamming***. El spamming degrada el valor de un índice convirtiéndose en una molestia.

La lógica que lleva a la gente intentar tales trucos es bastante rara. "Me imagino que toda la gente busca la palabra 'sexo'. Yo no tengo algo de sexo en mi sitio, pero yo quiero que la gente inadvertidamente llegue a mi sitio. Por lo tanto voy a poner la palabra 'sexo', tres mil veces como comentarios. Entonces cada vez que alguien busque 'sexo' mis páginas se mostrarán primeras en la lista de resultados".

[2]Palabras que son apropiadas para lo que está en la página.

La gente realmente intenta eso. Han tratado de hacer el mismo tipo de cosas en los fondos de sus páginas. Incluso han creado página tras página de texto que está en el mismo color que el color de fondo, por lo tanto los visitantes no verán esas palabras pero el spider de la máquina de búsqueda lo hará. Han intentado todo lo inimaginable para engañar a las máquinas de búsqueda.

Si es importante ser encontrado por las máquinas de búsqueda, uno debe ser muy cuidadoso acerca de donde hospeda sus páginas. Si el servicio de hospedaje también hospeda spammers y pornógrafos, uno sería penalizado o excluido simplemente porque la dirección IP de ese servicio es la misma para todos los dominios virtuales que incluye.

2.6.3 Enlaces

Los buscadores generalmente basan su ranqueo en factores estáticos (cálculos del valor de una página independientes de cualquier consulta en particular) y factores consulta-dependientes. Los factores estáticos valoran:

- Largas páginas **ricas en texto significativo** (o sea aquellas letras y palabras que no son generadas al azar).
- Las páginas que sirven como buenos puntos de partida hacia otras páginas, con **muchos enlaces a páginas que tienen contenido relacionado** (similitud en tópicos, no aquellas generadas por el intercambio de enlaces con la intención de generar una falsa impresión de "popularidad").
- La conectividad de las páginas, incluyendo no sólo cuantos enlaces hay a una página sino de donde vienen los enlaces: el número de dominios distintos y la "calidad" de ranqueo de esos sitios en particular. Esto es calculado para el sitio y también para las páginas individuales. **Un sitio o una página son "buenos" si muchas páginas en muchos sitios diferentes apuntan a ellos y especialmente si muchos "buenos" sitios apuntan a ellos**.
- El nivel del directorio en el cual la página reside. **El más alto es considerado más importante**. Si una página es almacenada profundamente (demasiados niveles en el camino hacia ella), el crawler simplemente no irá tan lejos y nunca la encontrará.

Estos factores estáticos son recomputados cerca de una vez por semana, y las nuevas buenas páginas lentamente escalan en el ranking.

Los factores consulta-dependientes incluyen:

- El título HTML.
- Las primeras líneas del texto.
- Las palabras y frases de la búsqueda que aparecen al principio antes que las que aparecen al final.
- Metatags que son tratados como palabras ordinarias en el texto que aparece en el comienzo de la página (si es notorio que los metatags no están relacionados con el contenido de la página, la página será penalizada).
- Palabras mencionadas en el texto asociado a los enlaces (p.ej: si muchos sitios buenos apuntan a un sitio con enlaces que poseen el texto "Visual Age" y la consulta es "Visual Age", se tendrán buenas chances de aparecer en lo más alto de la lista de resultados).

Se debe tener en cuenta que en una consulta **las palabras raras cuentan más que las comunes**. Si alguien busca por "fruta y granada", las páginas con la palabra "granada" aparecerán en lo alto de la lista (una técnica conocida como "frecuencia de documentos inversa"). Se deberían utilizar términos específicos en las páginas, en los anchors y en los metatags, no generales que no darán ninguna ventaja. **Hay que ser específico siempre que se pueda.**

2.6.4 Frase de búsqueda

Las tres subsecciones anteriores se refieren a conceptos estáticos de la página al momento del cálculo de la relevancia ante la solicitud de una búsqueda. Los metatags son valores fijos que están determinados en el texto de la página y se almacenan en el catálogo del buscador al momento de la indexación. El spamming se calcula cuando se detecta algún tipo de spam en la página y se mantiene fijo mientras la misma no sea modificada. Finalmente los enlaces, de manera similar a los metatags, se obtienen de las páginas al momento de la indexación, recorriendo el árbol de referencias, generalmente con la intención de obtener la popularidad de la página. La frase de búsqueda se diferencia del resto de los elementos utilizados en el cálculo de la relevancia por el hecho de que aporta un factor dinámico al ranqueo.

Este elemento está compuesto por una serie de palabras, y quizá frases, que sigue una sintaxis que varía con cada buscador pero que guarda cierta similitud. Por lo general esta frase admite la utilización de signos `+` y `-` que permiten indicar si una palabra debe aparecer o no en cada página que fuera a ser incluida en el conjunto de resultados. Si uno no utilizase ninguno de estos signos estaría permitiendo que la palabra pueda o no aparecer en la página (concepto de opción).

En si la frase de búsqueda no está compuesta únicamente por palabras sueltas. Por ejemplo, si buscáramos información sobre **San Martín** e insertáramos esta frase en el buscador, no sería lo mismo para él buscar: **San** y **Martín** como palabras separadas, que buscar **San Martín** como si fuera una única palabra. En el primer caso el buscador generalmente traería todas aquellas páginas que contengan la palabra **San**, la palabra **Martín** o ambas palabras (en cualquier posición y orden). En el segundo caso, en cambio, traerá un subconjunto de dichas páginas en las que aparezca la frase **San Martín** (ambas palabras en el orden especificado y una junto a la otra). La forma de lograr el segundo caso es encerrando la frase entre comillas dobles; para este ejemplo: **"San Martín"**.

Si uniéramos lo descrito en los dos párrafos anteriores, para buscar todas aquellas páginas que traten sobre el tópico de la pelea de Chacabuco en la que participó San Martín, deberíamos introducir la siguiente frase:

```
+Chacabuco +"San Martín"
```

Es importante remarcar que el uso de comillas a veces no es necesario en algunos buscadores, ya que "deducen" cuales son palabras sueltas y cuales frases. Por ejemplo, en Altavista, si buscáramos **Charles Darwin**, el buscador automáticamente buscaría la frase **"Charles Darwin"**. Esto se logra manteniendo una lista con las frases más utilizadas en los distintos lenguajes. Obviamente, nunca será redundante la utilización de las comillas, ya que las bases de datos que almacenan estas frases son finitas.

Algunos buscadores brindan la funcionalidad de las llamadas "búsquedas avanzadas" mediante la cual se pueden realizar búsquedas expresadas en lenguaje lógico, usando los operadores *AND*, *OR* y *NOT*. Esta funcionalidad está provista a través de un asistente, donde se colocan las palabras a buscar (como operandos) interconectadas con los operadores mencionados anteriormente.

2.7 Ejemplo: Altavista

Esta sección ejemplificará el uso de todos los conceptos descritos en este capítulo aplicados a una máquina de búsqueda particular, en este caso volcados a Altavista.

2.7.1 Partes

Altavista contiene las 3 partes nombradas. Tiene un spider llamado "scooter" que indexa las páginas de la web. Cuenta además con otro spider más específico, llamado "vscooter", que es un indexador multimedial (almacena en el índice referencias a imágenes, videos, sonidos, etc.). Mantiene un catálogo actualizado por estos spiders y posee, como todo buscador, el software de búsqueda de frases.

2.7.2 Clasificación

En base a que Altavista posee un spider, y que permite la incorporación de URLs a través de una interfaz, podemos decir que está dentro del grupo de las máquinas de búsqueda híbridas (aunque mayormente es una máquina de búsqueda pura).

2.7.3 Agregado de URLs

Posee una página que facilita que cualquier usuario puede dar de alta una URL. ¿Por qué cualquier usuario? Porque al no ser un directorio un navegante puede solicitar la inserción de una página en el catálogo de Altavista. Esta página luego será visitada por el spider y lo que el usuario haya indicado no influirá en la indexación, sólo se considerará como un pedido de visita.

También se pueden realizar bajas a través de la misma interfaz. Si un usuario agrega una URL de una página ya existente en el catálogo de Altavista, y el spider al revisitarla detecta que no existe, se la retira del catálogo.

2.7.4 Indexación

La obtención de URLs puede producirse por el spider, al seguir enlaces de páginas indexadas hacia nuevas páginas, o por el agregado de URLs de parte del usuario.

Luego, las páginas referenciadas por las URLs, son recuperadas y almacenadas como texto plano, ya que según Altavista se mantiene la información intacta sin pérdida del concepto de "unidad de página".

2.7.5 Ranqueo y cálculo de relevancia

El método de ranqueo de Altavista es en general el que siguen todos los buscadores. Da preferencia al título y al cuerpo de la página, utiliza los metatags sin darle demasiada importancia ya que se componen de texto no visible para el navegante, sigue los enlaces definidos en las páginas y realiza cálculo de popularidad.

Penaliza el spamming (en realidad no cuenta a una palabra más de dos veces por página) y penaliza los dominios que almacenan spammers, con lo cual una página que no realice spamming pero que esté almacenada en el mismo ISP puede sufrir penalización. Detecta spamming por ocultamiento mediante colores y mediante texto pequeño.

Pueden realizarse búsquedas simples mediante el uso de los signos "+" y "-", y búsquedas avanzadas utilizando los operadores "AND", "OR" y "NOT". Para realizar búsquedas por sentencias exactas (en posición y orden) provee la funcionalidad de comillas, aunque detecta

frases comunes. Es más, da mayor prioridad a las palabras menos usuales en los distintos lenguajes.

Es posible realizar una limitación en el conjunto de búsqueda en base al lenguaje, al sitio o por referencia a un enlace específico (usando las palabras clave :language, :site y :link).

2.7.6 Evitando la indexación

Altavista se adhiere al *Estándar de Exclusión de Robot*, por lo que si el archivo "robots.txt" contiene exclusiones hacia su indexador de páginas, "scooter", o hacia su spider multimedial, "vscooter", no procesará las páginas o directorios excluidos por el mismo.

También soporta la exclusión de páginas en particular, respetando el metatag con palabra clave "robots" y su contenido "NOINDEX", "NOFOLLOW" o "NOIMAGEINDEX".

2.8 Comparación de máquinas de búsqueda

Aquí se describe, a través de una tabla, un resumen de los factores y características importantes que pueden afectar a como los sitios son indexados y ranqueados. Inmediatamente a continuación de la tabla encontraremos explicaciones completas de cada uno de los ítemes.

Nota: esta tabla abarca Altavista, Excite, Google, Inktomi, Infoseek, Lycos y Northern Light. Excite también cubre los servicios que están sustentados por él: AOL NetFind y WebCrawler, a pesar de que estos servicios se nombren individualmente. Lo mismo ocurre para Inktomi, que incluye HotBot y MSN Search. Esta tabla está actualizada al 6 de Julio de 1999.

Expansión (Crawling)

Aquí se cubren los factores relacionados a cuan bien las máquinas de búsqueda se expanden a través de los sitios web.

- **Expansión en profundidad:** las máquinas de búsqueda que realizan una expansión profunda listarán muchas páginas de un sitio web, a pesar de que las páginas no sean explícitamente propuestas a ellas. El resto usualmente listará muchas menos páginas de un sitio. En general, cuanto más grande es el índice de una máquina de búsqueda, será más probable que tenga muchas páginas por sitio.

- **Indexación instantánea:** en una máquina de búsqueda con indexación instantánea, usualmente una página que sea propuesta aparecerá dentro de uno o dos días del momento del agregado de la URL.

- **Soporte de frames:** esto muestra cuales máquinas de búsqueda pueden seguir enlaces de frames. Aquellas que no pueden probablemente se perderán mucho de un sitio.

- **Mapas de imágenes:** esto muestra cuantas máquinas de búsqueda pueden seguir mapas de imágenes. Como ocurre con los frames, aquellas máquinas de búsqueda que no puedan seguir mapas de imágenes probablemente se perderán de listar mucho de un sitio.

- **robots.txt:** el archivo robots.txt es el establecido por los webmasters para mantener a las máquinas de búsqueda fuera de los sitios.

- **Metatag "robots":** este es un descriptor especial que permite, a los dueños de las páginas, especificar si deben ser indexadas.

Expansión	Altavista	DIPI	Google	Inktomi	NLight	Excite	Infoseek	Lycos	Notas
Expansión en profundidad	S	S	S	S	S	N	N	N	WebCrawler sólo lista las Home Pages
Indexación instantánea	S	S	N	N	N	N	S	N	Las páginas aparecerán dentro de los 2 días
Frames	S	S	S	N	S	N	N	N	Lycos provee soporte limitado
Mapas de imágenes	S	S	N	N	S	N	S	N	
robots.txt	S	S	S	S	S	S	S	S	
Metatag "robots"	S	S	N/A	S	S	S	S	S	Google puede no soportarlo
Popularidad de enlaces	N	S	N/A	S	N	S	N	S	
Apredizaje de frecuencia	N	N	N/A	S	N	S	N	S	
Indexación	Altavista	DIPI	Google	Inktomi	NLight	Excite	Infoseek	Lycos	Notas
Texto del cuerpo completo	S	S	S	S	S	S	S	S	Excepto las palabras de detención
Palabras de detención	S	N	S	S	N	S	N	S	
Metatag "description"	S	S	N	S	N	S	S	N	
Metatag "keywords"	S	S	N	S	N	N	S	N	
Tag "alt" en imágenes	S	S	N	N	N	N	S	S	
Comentarios	N	N	N	S	N	N	N	N	
Ranqueo	Altavista	DIPI	Google	Inktomi	NLight	Excite	Infoseek	Lycos	Notas
Relevancia a metatags	N	S	N	S	N	N	S	N	
Relevancia por revisión	N	N	N	N	N	S	S	N	Muy importante en Infoseek
Popularidad de enlaces	S	S	S	S	N	S	S	N	Muy importante en Google
Direct Hit	N	N	N	N	N	N	N	N	Lo soporta HotBot motorizado por Inktomi
Spam	Altavista	DIPI	Google	Inktomi	NLight	Excite	Infoseek	Lycos	Notas
Metatag "refresh"	S	N	N	N	N	N	S	S	
Texto invisible	S	N	N	S	S	N	S	S	
Texto pequeño	S	N	N	S	N	N	N	S	
Repetición de palabras	S	S	S	S	S	S	S	S	Altavista limita a 2 palabras

Tabla 2.1: *Características de los buscadores más populares.*

- **Popularidad de enlaces:** todas la máquinas de búsquedas pueden determinar la popularidad de una página analizando cuantos enlaces hay hacia ella desde otras páginas. Algunas motores usan esto como una forma para determinar cuales páginas serán incluidas en el índice.

- **Aprendizaje de frecuencia:** un número de máquinas de búsqueda pueden aprender con que frecuencia las páginas cambian. Las páginas que cambian comúnmente pueden ser visitadas más frecuentemente.

Indexación

Esta subsección explica como los distintos spiders de los motores indexan una página.

- **Texto del cuerpo completo:** muchas máquinas de búsqueda dicen que indexan todo el texto visible del cuerpo de una página, otras no indexarán las "palabras de detención" o excluirán las palabras que estiman que forman parte del spamming (explicado en puntos subsiguientes).

- **Palabras de detención:** algunas máquinas de búsqueda dejan fuera algunas palabras cuando indexan o puede que no busquen por estas palabras en una consulta. Estas "palabras de detención" son excluidas de forma tal de ahorrar espacio de almacenamiento o ganar velocidad en la búsqueda.

- **Metatag "description" y "keywords":** muestra cuales máquinas de búsqueda soportan estos metatags. No significa que usando estos descriptores se les de a las páginas mayor relevancia, esto esta cubierto en la sección siguiente.

- **Texto alternativo de imágenes. Comentarios:** muestra cuales motores indexan el texto alternativo (parámetro ALT del descriptor IMG) o el texto de los comentarios.

Ranqueo
Entre los factores comunes de ranqueo de páginas en base a una consulta se encuentran el título y la frecuencia de palabras, pero otros factores importantes se listan a continuación:

- **Relevancia a metatags:** algunas máquinas que soportan los metatags "description" y "keywords" también darán una relevancia extra si el término de búsqueda aparece en estas áreas. No todos los motores que soportan esos descriptores también dan una relevancia extra.

- **Relevancia por revisión:** algunos motores también revisan los sitios o los listan en un directorio asociado, dándole una relevancia especial por encontrarse en estos directorios.

- **Relevancia por popularidad de enlaces:** aquellas páginas que tienen muchos enlaces, o enlaces de sitios web importantes, adquieren un incremento en su relevancia.

- **Direct Hit:** este sistema consiste en la medición de los usuarios que clickean directamente en alguno de los resultados. Algunos motores usan esto como factor en el ranqueo.

Spamming
Las máquinas de búsqueda principales penalizan a los sitios que utilizan el spamming para tratar de mejorar su posición. Si detectan la técnica de spamming, pueden degradar el ranqueo de una página o excluirla de los listados. Los ítemes siguientes detallan aquellos elementos que pueden causar penalización por spamming.

- **Metatag "refresh":** algunos dueños de sitios crean páginas que automáticamente llevan a los visitantes a diferentes páginas dentro del mismo sitio (las llamadas páginas puente). El metatag "refresh" es un modo típico de hacer esto. Algunas máquinas de búsqueda no indexarán una página con un refresco muy rápido o directamente no aceptarán la redirección.

- **Texto invisible:** esta es una técnica que consiste en colocar texto en una página en el mismo color que el fondo, haciéndolo invisible a los navegantes. Muchos motores no indexan este texto o no indexan la página.

- **Texto pequeño:** es una técnica que consiste en colocar texto en una página en una fuente de pequeño tamaño. Las páginas que contengan texto pequeño no serán indexadas o quizás sólo el texto será excluido.

Capítulo 3

Agentes

El estudio de los agentes es un área de investigación en pleno y rápido desarrollo. El abuso producido en el uso del término "agente" ha llevado a enmascarar lo que en realidad es un campo heterogéneo de investigación, aunque, eso sí con múltiples variantes.

A pesar de que todavía no existe consenso para definir el concepto de agente, se han hecho ya importantes aproximaciones a la definición, diseño y construcción de diversos tipos de agentes. En este capítulo pretendemos dar una visión detallada de la noción agente y de cuales son sus características básicas, haciendo una revisión general de los diferentes avances realizados en este campo, haciendo especial hincapié en las propiedades y habilidades de los agentes orientados a Internet.

Los esfuerzos en investigación sobre agentes inteligentes han sido dirigidos inicialmente por la comunidad de Inteligencia Artificial. La pregunta ¿Qué es un agente? ha suscitado una gran discusión en el seno de esta comunidad, como ocurrió en su tiempo con la pregunta ¿Qué es la inteligencia?.

3.1 Historia de los agentes

Los agentes de software se pueden considerar como el resultado de la evolución de los sistemas multiagente, una de las áreas de investigación que, junto con la Resolución de Problemas Distribuida (DPS) y con la Inteligencia Artificial Paralela (PAI), se engloban en la Inteligencia Artificial Distribuida (DAI).

El concepto de agente, en el sentido que pretende comprender este documento, puede ser observado en los primeros días de investigación en la DAI, en los años 70, donde fue propuesto el paradigma de los actores . En este modelo se proponía el concepto de un objeto autocontenido, interactivo y concurrente llamado actor, que tiene encapsulado un estado interno y que responde a mensajes de otros actores. En el desarrollo histórico de los agentes se pueden distinguir dos ramas principales [Nwa96]: la primera cubre desde 1977 hasta nuestros días, y la segunda desde 1990 también hasta nuestros días.

La primera rama se centra en los agentes como entidades vivas y deliberativas, con un estado interno de tipo simbólico. Se define aquí a los agentes como un artefacto que posee un modelo simbólico del mundo, representado de manera explícita, donde las decisiones -por ejemplo sobre qué acciones realizar- se toman mediante razonamiento simbólico.

La rama que surge a partir de 1990 es muy distinta, pues incluye los trabajos en investigación y desarrollo sobre software de cualquier tipo al que se le pueda asignar el término agente. Algunos no dudan en afirmar que esta rama aparece debido a que en nuestros días todo el mundo llama agente a cualquier tipo de artefacto.

3.2 Agentes

Como hemos dicho, por el momento no parece haber una definición universalmente aceptada para los agentes, aunque se pueden encontrar algunas que pueden ayudar a formarnos una idea de lo que es un agente.

Una de las primeras definiciones aparecidas es la comentada anteriormente, un agente es algo que puede ser visto percibiendo el entorno y actuando sobre él . Esta definición depende fuertemente de lo que se considere como "entorno" y también de lo que se considere como "percibir" y "actuar". Un agente es una entidad de software con un propósito específico. La persistencia distingue a los agentes de las ya clásicas subrutinas, ya que los agentes tienen sus propias ideas sobre cómo realizar tareas y las subrutinas no. El propósito específico los distingue de las aplicaciones generales, aunque ciertamente hay autores que no consideran el tener un propósito específico como una de las características fundamentales de los agentes.

Un agente autónomo es un sistema que habita en un entorno dinámico y complejo, en el que percibe y actúa de manera autónoma, alcanzando el conjunto de objetivos para el que fue diseñado . Esta definición añade a los agentes un aspecto importante como es la autonomía y la capacidad de poseer un conjunto de objetivos.

Hay definiciones que tienen en cuenta el razonamiento que realiza un agente en su proceso de selección de acciones a realizar. Los agentes inteligentes realizan tres funciones continuamente: perciben las condiciones dinámicas de su entorno, actúan con el objetivo de modificarlo y razonan en orden a interpretar las percepciones, resolver problemas y determinar las acciones a realizar . O también, programa autocontenido que es capaz de controlar sus acciones y decisiones para alcanzar unos objetivos, basándose en su percepción del entorno . Entidades de software que llevan a término un conjunto de operaciones bajo las órdenes de un usuario o de otro programa, con un cierto grado de independencia o autonomía, empleando conocimiento o representaciones de los objetivos del usuario .

Donde parece haber acuerdo es en considerar los agentes como entidades autónomas e inteligentes, juzgando estas características como fundamentales.

La autonomía referida a los agentes puede considerarse como que un agente ha de tener una actuación periódica, una ejecución espontánea e iniciativa, en la que el agente debe ser capaz de realizar acciones independientes o apropiativas que beneficiarán al usuario . Para comprobar esto sería interesante disponer de un test de autonomía para poder decir si una acción es realmente independiente o apropiativa.

La idea de autonomía parece más útil que la de inteligencia para distinguir a los agentes de otro tipo de software, por esto se puede considerar la autonomía como elemento fundamental en la definición de los agentes.

A modo de resumen, se puede decir que un agente es un sistema que está situado y que forma parte de un determinado entorno, que percibe este entorno y que actúa en él continuamente con su propia planificación, persiguiendo el objetivo de cambiar su propia percepción [Gra96]. De esta definición se deriva que los agentes están fuertemente ligados al entorno, y si cambiamos el entorno puede ocurrir que lo que era un agente deje de serlo. Esto es, un agente lo es en un entorno determinado.

Esta definición también permite distinguir a un agente de un simple programa. Se podría decir que un programa "normal" es un agente si se consideran sus entradas como las percepciones que tiene del entorno, y sus salidas como la actuación sobre él, pero en realidad esto no es un agente, ya que sus salidas no afectan a sus percepciones futuras y además no tiene continuidad temporal. Podemos concluir que todos los agentes son programas, pero que no todos los programas son agentes.

De la misma manera se puede decir que las subrutinas no son agentes, y que las subrutinas

que poseen los agentes no tienen por qué serlo, aunque los agentes pueden tener subcomponentes que a su vez son agentes.

Los agentes no se definen por las tareas que realizan. Así, por ejemplo, un simple corrector ortográfico no es un agente -por las razones expuestas anteriormente-, pero un corrector ortográfico que observa como escribe el usuario y corrige sobre la marcha los errores sí puede ser considerado un agente.

3.2.1 Características

Hasta aquí hemos tratado de aproximarnos someramente a la noción de agente. Viendo la incapacidad de encontrar una definición comúnmente aceptada, vamos a tratar de exponer las características en las que sí hay unanimidad en considerar fundamentales para los agentes.

Cuando se intenta enumerar las características de los agentes aparece de nuevo el problema de los diferentes usos del término agente. Se presenta aquí una visión amplia de agente, que incluye características aceptadas generalmente pero que hay quien considera que son insuficientes, lo que resulta en la elaboración de una lista de características que constituyen una visión estricta o fuerte de los agentes .

Visión amplia de los agentes

El uso más general del término agente se utiliza para definir sistemas hardware o software con las siguientes propiedades:

- Autonomía: Los agentes han de actuar sin la intervención directa de los humanos o de otros agentes y además han de tener algún tipo de control sobre sus acciones y su estado interno .
- Sociabilidad (habilidades sociales): Los agentes pueden interactuar con otros agentes (y posiblemente con humanos) mediante algún tipo de lenguaje de comunicación de agentes [Ket94].
- Reactividad: Los agentes perciben su entorno (que puede ser el mundo real, un usuario a través de una interfaz gráfica, un conjunto de agentes, Internet o quizás la combinación de algunos de los anteriores) y responde en un tiempo aceptable a los cambios que ocurren en él.
- Capacidad de iniciativa: Los agentes no sólo han de reaccionar a los cambios de su entorno, también han de ser capaces de tomar la iniciativa, exteriorizando algún tipo de comportamiento orientado a alcanzar unos objetivos. Esta visión de los agentes se ha propagado con gran facilidad entre un gran grupo de investigadores.

Visión estricta de los agentes

Algunos investigadores, especialmente los que trabajan en IA, consideran que las propiedades de autonomía, sociabilidad, reacción e iniciativa no son suficientes para caracterizar a los agentes. Su opinión se apoya en el pensamiento de que un agente es un sistema computacional que además de las cualidades anteriores, ha sido diseñado o implementado utilizando conceptos que normalmente se aplican a los humanos. Es corriente en la IA el caracterizar a los agentes usando nociones mentales como pueden ser conocimiento, creencias, intenciones y obligaciones . Algunos incluso se atreven a considerar los llamados agentes emocionales .

Para estos autores, un agente ha de tener además de las características generales:

- Propiedades intencionales
- Capacidad de razonamiento
- Movilidad
- Veracidad
- Benevolencia
- Racionalidad

3.2.2 Clasificación

Para hacer una clasificación de los agentes se pueden emplear criterios muy diversos. Puede ser útil clasificar los agentes de acuerdo con el conjunto de sus propiedades. Cada agente posee unas propiedades básicas. Si se añaden nuevas propiedades se pueden distinguir distintos tipos de agentes como pueden ser los móviles, los que aprenden, etc.

Hay otros modos de clasificar a los agentes. Podemos clasificarlos según las tareas que realizan , la eficiencia de sus acciones, su estado interno o el entorno en el que se encuentran .

Una taxonomía sobre agentes podría basarse también en el modelo biológico, siguiendo la jerarquía natural de los agentes. Se trata de una clasificación al mismo nivel de la que en el mundo real se distingue entre el reino animal, vegetal o mineral.

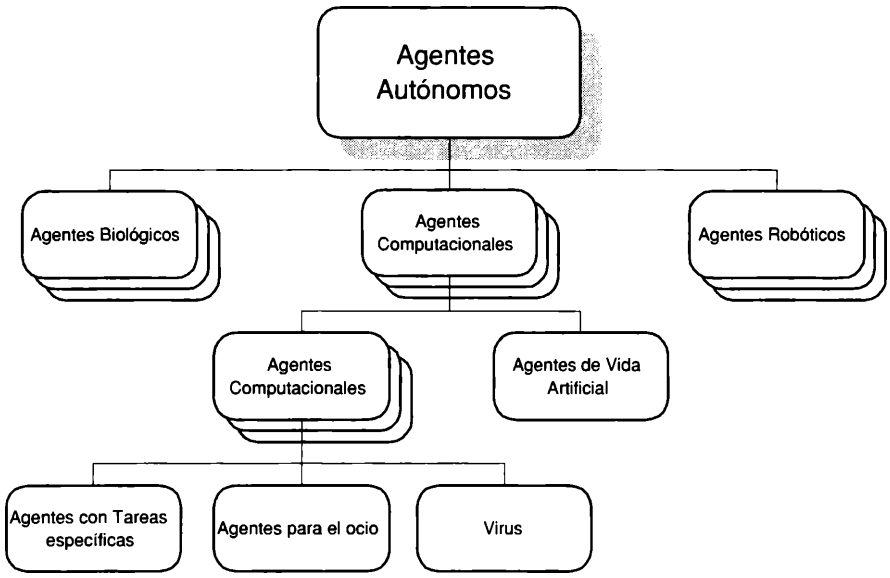

Figura 3.1: *Taxonomía de los Agentes [Gra96].*

A nivel de "reino" situamos a los agentes autónomos, pues ya hemos dicho que la autonomía es una propiedad esencial. Podemos clasificar a los agentes autónomos como agentes biológicos, robóticos o computacionales, como si se trataran de clases naturales. Hay que notar que cada cultura, o incluso cada niño, fácilmente distingue entre organismos vivos, cosas y conceptos abstractos. En el nivel de "phylum" podemos, de manera razonable, clasificar los agentes computacionales en agentes de software y en agentes de vida artificial . A nivel de "clase" podemos subclasificar los agentes software en agentes con tareas específicas, agentes

para el ocio y virus informáticos. Esta clase es la más grande, y es la que hemos tratado de definir en este capítulo.

3.3 Teoría de agentes

En la sección anterior hemos dado una visión informal de la noción de agente. Ahora pretendemos mostrar una teoría, más o menos general, de los agentes. Consideramos una teoría de agentes como la manera de especificar un agente , los teóricos de los agentes desarrollan formalismos que sirven para representar las propiedades de los agentes, de forma que, usando estos formalismos se puedan definir teorías que engloben las propiedades deseables de los agentes.

3.3.1 Actuación de los agentes

El objetivo que nos proponemos consiste en la construcción de agentes que puedan ser vistos como conocedores del mundo en que habitan, y capaces de razonar sobre los posibles cambios que ocasionan sus acciones y sobre las alternativas. Un agente no sólo ha de ser capaz de razonar sobre los cambios que hay en el entorno, sino también sobre los cambios que él mismo provoca en el entorno. Tal como veremos más adelante, razonar sobre las alternativas futuras significa planificar.

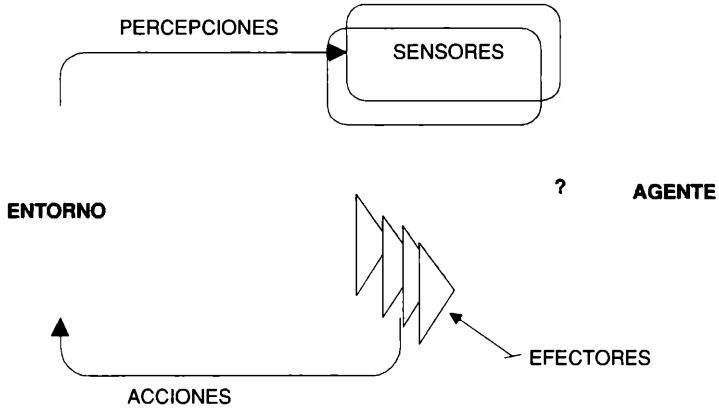

Figura 3.2: *Agente genérico que interactúa con el entorno.*

Como siempre, un buen punto de partida es estudiar algunas de las características deseables de los agentes. Un agente ha de ser capaz de aceptar nuevas tareas en forma de metas descritas explícitamente. Ha de poder mejorar su actuación mediante entrenamiento o por aprendizaje de nuevos conocimientos sobre el dominio. Ha de adaptarse a cambios en el entorno y actualizar el conocimiento relevante. Para esto, los agentes necesitan saber cosas, tener conocimiento del estado actual del mundo, saber usar métodos de inferencia, inferir propiedades desconocidas mediante la propia percepción del mundo, han de saber sus objetivos, etc. De estas propiedades se deriva que un agente es cualquier cosa que percibe su entorno mediante sensores y que actúa sobre él mediante efectores o actuadores. En la figura 3.2 podemos ver un diagrama que representa a este agente genérico.

Hace falta ahora encontrar una medida sobre la conducta de los agentes, es decir, sobre la inteligencia. Un agente racional es aquel que realiza la acción correcta . La acción correcta es aquella que permite que el agente tenga éxito. La idea de éxito depende de cada agente, tener éxito puede significar supervivencia, alcanzar el máximo beneficio, etc. En este punto podemos definir a un agente racional ideal como "algo" que, para cada secuencia de percepciones, trata de realizar una acción que maximiza su medida de actuación, en función de lo percibido y del contenido de su base de conocimiento. Se puede caracterizar a un agente según su entorno y sus percepciones, acciones y objetivos .

3.3.2 Estructura de los agentes

Hemos hablado, en esta sección, del comportamiento de los agentes. Vamos a ver ahora cómo trabajan interiormente los agentes. Es función de la IA el diseño del programa de un agente, este programa implementa la relación que existe entre las percepciones y las acciones de deben realizarse. El programa del agente se ejecuta en una determinada arquitectura hardware, y esta arquitectura hace que las percepciones de los sensores estén disponibles para el programa. La relación entre agente, programa y arquitectura puede representarse como sigue:

```
agente = arquitectura + programa
```

Para diseñar el programa de un agente hay que formarse una idea de las posibles percepciones y las acciones, los objetivos o qué medida de actuación puede alcanzar el agente, y además, en que tipo de entorno se situará. Estos cuatro aspectos, se utilizarán, a partir de ahora, para caracterizar a un agente, y los nombraremos como sigue:

```
PAGE (percepciones, acciones, objetivos, entorno)
```

PAGE deriva de las palabras en inglés: perceptions, actions, goals y environment. En la Tabla 3.1 se puede observar un ejemplo de distintos tipos de agentes con su PAGE.

AGENTE	PERCEPCIONES	ACCIONES	OBJETIVOS	ENTORNO
Sistema de diagnosis médica	Síntomas de los pacientes	Preguntas, tests, tratamientos	Salud del paciente, minimizar costos	Paciente, hospital
Analizador de imágenes vía satélite	Pixeles de distinta intensidad, color, etc.	Dibujar una escena, obtener categoría	Corregir categoría	Imágenes enviadas por un satélite en órbita
Profesor interactivo de inglés	Palabras escritas mediante un teclado	Mostrar ejercicios, sugerencias, correciones	Maximizar la puntuación de los tests del alumno	Conjunto de estudiantes
Controlador de una refinería	Temperatura, presión	Abrir, cerrar válvulas; ajustar temperatura	Maximizar seguridad y pureza del producto	Refinería

Tabla 3.1: *Ejemplos de tipos de agentes con su PAGE.*

3.3.3 Entorno

El entorno es donde habitan nuestros agentes. Como ya se ha dicho, el entorno determina que un agente lo sea, esto es, el entorno limita y condiciona al tipo de agente que podemos encontrar en él. Esto significa que antes de diseñar un agente hay que tener en cuenta las propiedades del entorno. El entorno puede clasificarse como:

- Real vs Digital: Un entorno real sería aquel en el que se encuentran los robots autónomos: almacenes, fábricas, etc. El clásico ejemplo de entorno digital es Internet, donde se construyen los "softbots", de los que hablaremos más adelante.

- Accesible vs Inaccesible: Cuando los sensores del agente son capaces de obtener toda la información que hay en el entorno decimos que éste es accesible. Nos referimos aquí a la información relevante para poder determinar qué acciones hay que realizar. Si parte de la información del mundo queda oculta al agente significa que es inaccesible. En un entorno totalmente accesible no es necesario que el agente mantenga un estado interno ya que a través de sus sensores conoce en todo momento el estado en el que se encuentra.

- Determinista vs No determinista: El entorno es determinista si existe un conjunto de reglas que permiten pronosticar cómo va a cambiar. En otras palabras, en estos casos el futuro del entorno viene totalmente determinado por su estado actual y por la acción que ejecuta el agente. Cuando tengamos que decir si un entorno es determinista o no determinista hay que hacerlo desde el punto del agente, ya que si el entorno es inaccesible, el agente considerará que es no determinista, aunque para el diseñador sea lo contrario.

- Episódico vs No episódico: En un entorno episódico la interacción del agente con el entorno está dividida en fases de percibir y actuar. La percepción actual no está determinada por fases o episodios anteriores y no es necesario tener en cuenta las consecuencias futuras de las acciones actuales, esto significa que desaparece la necesidad que el agente planifique su actuación.

- Estático vs Dinámico: Si el entorno puede cambiar mientras el agente esta tomando decisiones respecto a alguna percepción entonces es dinámico. Los agentes que están en entornos dinámicos son mucho más complejos que los que actúan en entornos estáticos.

- Discreto vs Continuo: Es discreto cuando hay un número finito de acciones y de percepciones posibles.

ENTORNO	Accesible	Determinista	Episódico	Estático	Discreto
Póker	No	No	No	Si	Si
Backgammon	Si	No	No	Si	Si
Diagnóstico médico	No	No	No	No	No
Ajedrez (con reloj)	Si	Si	No	Si/No	Si
Ajedrez (sin reloj)	Si	Si	No	No	Si

Tabla 3.2: *Ejemplos de entornos donde pueden actuar los agentes.*

Hemos visto hasta ahora qué podemos entender por "agente", aunque normalmente, cuando se construyen aplicaciones basadas en agentes, no es frecuente que estas tengan un sólo agente, sino que se construyen sistemas con varios agentes con la idea de aprovechar las habilidades de cada uno. En esta sección vamos a exponer cómo pueden cooperar entre sí varios agentes con la intención de alcanzar un objetivo común, cómo nace este método de trabajo y cuáles son sus características fundamentales.

3.4 Sistemas Multiagente

El paradigma de los Sistemas Multiagente (MAS) se fundamenta en la idea de superar las limitaciones inherentes a cualquier sistema inteligente, natural o artificial, formando agrupaciones de sistemas simples de tal forma que puedan compartir conocimientos y capacidades de la misma manera que la gente supera sus limitaciones individuales agrupándose en sociedades. Cada uno de estos sistemas simples que agrupamos constituiría un agente y la comunidad considerada globalmente es un sistema multiagente.

La inteligencia artificial distribuida estudia los problemas que aparecen cuando varios agentes interaccionan para tratar de resolver problemas. Pretende construir agentes inteligentes capaces de tomar decisiones que les permitan alcanzar sus objetivos en un mundo poblado por otros agentes inteligentes, con objetivos propios. Esto incluye el estudio de mecanismos para la coordinación, organización y negociación entre agentes. Es necesario en este punto diferenciar entre la resolución de problemas de manera distribuida, la búsqueda de soluciones distribuidas y los sistemas multiagente. Encontrar una solución distribuida a un problema está siempre relacionado con su naturaleza. Hasta este momento los trabajos realizados al respecto consisten en aplicaciones que necesitan una interpretación y una planificación distribuida. Desde el punto de vista de los agentes, la DAI (Inteligencia Artificial Distribuida) puede considerarse como una generalización de las estrategias anteriormente expuestas, puesto que no impone ninguna metodología individual específica. Cada agente puede aplicar técnicas propias para alcanzar su objetivo, aunque será necesario trabajar bajo una metodología global que organice todo el sistema.

La inteligencia artificial, hasta el momento en que aparecieron las ideas de agente y sistema multiagente, estaba básicamente concentrada en sistemas monolíticos. Estos sistemas, pueden considerarse como de un solo agente con un único objetivo y con conocimiento fijo, el cual, la mayor parte de las veces, acaba siendo implementado como un Sistema Basado en el Conocimiento. La única solución utilizada para aumentar la capacidad de estos sistemas era aumentar el tamaño de su base de conocimiento. Los sistemas multiagente ofrecen la alternativa de fragmentar estas enormes bases de conocimiento en sistemas más pequeños y especializados, que cooperan entre sí. Desde este punto de vista es evidente que los sistemas multiagente posibilitan una expansión de los horizontes de la IA, ofreciéndonos una nueva perspectiva de entender y afrontar los problemas. Aunque también esta claro que no todos los trabajos realizados en el campo de la IA pueden ser de nuevo planteados y aprovechados desde el punto de vista de los agentes .

La nueva forma de plantear los problemas que ofrecen los MAS es muy sugestiva, ya que es una solución que la naturaleza ha aplicado desde siempre: los organismos unicelulares se agrupan para formar organismos pluricelulares, que en muchos casos acaban formando comunidades. Casi todas las materias de la IA pueden ser reinterpretadas desde el punto de vista de los sistemas multiagente, rescatándolas del callejón sin salida en que parecían abocados los sistemas basados en el conocimiento. Además, los sistemas multiagente constituyen un modelo aplicable a otras áreas aparte de la IA, como puede ser la Ingeniería del Software, la robótica, etc.

Para que este nuevo paradigma pueda considerarse realmente útil, y convertirse así en una disciplina, ha de estar en condiciones de ofrecer por lo menos, la capacidad de resolver problemas que hasta este momento habían quedado fuera de nuestro alcance, y poder solucionar de manera más eficiente problemas ya resueltos.

La resolución de problemas no acometidos anteriormente puede darse en distintas situaciones. Puede tratarse de problemas relativos a entornos abiertos, respecto de los cuales se desconoce a priori sus componentes y que pueden cambiar en el tiempo y que son muy hete-

rogéneos. Un ejemplo muy claro de esto es Internet. En estos dominios abiertos es necesario que los sistemas construidos y sus interfaces sean flexibles y robustas, ya que en el momento de hacer el diseño es difícil prever todas las interacciones que habrá. Es necesario que tengan sofisticados conocimientos sociales, no sólo la habilidad de comunicarse, sino capacidad de negociación para que puedan llegar a acuerdos, y mecanismos de coordinación que permitan describir cómo se han de llevar a cabo las acciones interrelacionadas, de forma cooperativa, por parte de varios sistemas alejados en el espacio. Es evidente que todas estas actividades sociales han de estar soportadas por un mecanismo de comunicación que permita la transmisión e intercambio de objetivos y conocimiento entre todos los agentes de un mismo entorno.

Cuando se trata de resolver problemas en entornos complejos, posiblemente la única manera en que puede ser abordado es desarrollando componentes especializados, en términos de representación y resolución de problemas. En este tipo de dominios el acercamiento basado en MAS implica que éste se puede separar de modo natural en componentes menores y más simples, que pueden ser desarrollados y mantenidos de forma aislada. En estas situaciones hace falta un sistema de interacción entre componentes o módulos, para así asegurar que las dependencias entre subproblemas se resuelven de manera correcta. Cada uno de estos componentes sería un agente especializado en resolver un subproblema de forma eficiente. Todos los agentes, por el hecho de ser independientes, podrían ser desarrollados con el paradigma más apropiado al subproblema que han de resolver. Se evita de esta manera tener que elegir un paradigma único para todo el sistema, lo cual representaría un compromiso global, pero que no sería óptimo en las distintas partes.

Cuando se trata de resolver más eficientemente problemas que ya tienen solución computacional, los sistemas multiagente pueden hacerlo mediante [Bon88] la distribución de datos, control o recursos. Esto se puede llevar a cabo valiéndose de la metáfora que consiste en utilizar la tendencia natural de los humanos que lleva a personalizar cualquier objeto que encuentran en su entorno, tenga poca o mucha autonomía, para presentar como inteligentes cosas que presentadas de otra forma no lo parecerían. Los MAS pueden también reestructurar los grandes sistemas que operan sobre dominios complejos y cambiantes, es mas fácil modificar, mantener y actualizar componentes que cubran partes concretas del dominio. Esto se puede hacer transformando la aplicación en un conjunto de agentes y envolviendo a cada agente con una capa se software que les proporcione habilidades sociales [Ket94].

3.5 Organización de los sistemas multiagente

Cuando se construye un sistema multiagente se han de planificar las actividades de cada agente teniendo en cuenta que interacción realiza con otros agentes, que pueden ayudarle o perjudicarle. La DAI tiene el objetivo de estudiar estas interrelaciones . Cualquier sistema basado en la inteligencia artificial distribuida ha de considerar un conjunto de propiedades como son el compromiso, negociación, cooperación, resolución de problemas de forma distribuida, coherencia, conocimiento de los otros agentes. La consecuencia es ver los MAS como "sistemas sociales", ya que agrupan a varios agentes y se asume que las propiedades del sistema no pueden derivarse únicamente de las propiedades de sus agentes.

Se propone entonces una organización de los MAS donde pueden ser vistos según tres dimensiones: las habilidades sociales de un agente individual, la organización de los agentes, y su control a lo largo del tiempo.

3.5.1 Las habilidades sociales de un agente individual

- Razonamiento sobre otros

Los sistemas DAI han de incorporar la habilidad de razonar sobre las acciones y los planes de los otros agentes, en orden de predecir su comportamiento. Un agente no sólo ha de razonar sobre los otros, sino que ha de ser capaz de cambiar las creencias y las acciones de los otros agentes, así como su comportamiento. Estas habilidades son fundamentales en las negociaciones entre agentes. El diseñador de sistemas multiagente ha de saber qué conocimiento es necesario para que los agentes realicen ciertas acciones, y cómo puede ser adquirido a través de la comunicación con los otros.

La mayoría de los modelos de conocimiento y creencias están basados en el modelo de "mundos posibles", que permite el razonamiento explícito sobre el conocimiento y las acciones . Con los mundos posibles se puede generar automáticamente deducciones lógicas. En este modelo, un agente dice conocer un hecho si ese hecho es cierto en todos los mundos que él piensa que son posibles. Este planteamiento presenta algunos problemas como el de la omnisciencia lógica ya que han de ser capaces de derivar todas las fórmulas válidas y sus consecuencias lógicas y el sentido común, no sólo para saber que un hecho es cierto, sino para saber lo que todos saben.

La adquisición y modificación del conocimiento propio según los otros plantea cuestiones semejantes a las que se estudian en aprendizaje automático. ¿Debe un agente cambiar sus creencias sólo porque ha mantenido una conversación con otro? Hay modelos que describen estrategias de cómo un agente puede cambiar sus creencias durante su interacción con otros : la teoría de fundamentos que hace que las creencias se mantengan hasta que las justificaciones cambien, la teoría de la coherencia hace que un agente conserve sus creencias hasta que éstas entren en conflicto con sus percepciones del entorno.

- Valoración de una situación distribuida

  En un sistema multiagente, un agente puede tener en todo momento una gran variedad de elecciones sobre lo que debe creer, qué conocimiento puede ser considerado relevante y qué acciones perseguir. Por tanto, cada agente individual debe valorar la situación en la que se encuentra, teniendo en cuenta a los otros agentes. Para esta valoración se necesita recoger, organizar y evaluar la información sobre el entorno, la cual puede adaptarse perfectamente a las expectativas del agente o utilizarse para crear nuevos agentes. Esta cuestión puede resolverse parcialmente aplicando técnicas de planificación, teniendo en cuenta que los agentes operan en entornos dinámicos.

- Planificación

  Un sistema de planificación trata de encontrar a priori una secuencia de acciones que transformen un estado del mundo que satisface la representación del estado inicial de un problema, en uno que cumple la descripción del objetivo. Este método de trabajo se utiliza normalmente en problemas con un sólo agente situado en un entorno estático que únicamente cambia como consecuencia de las acciones del agente.

  La planificación aplicada a los sistemas multiagente que operan en entornos dinámicos requiere representaciones apropiadas a acciones simultáneas y a procesos continuos, y además, mecanismos de resolución de problemas que usen estas representaciones.

  Un plan para un sistema DAI se construye con la idea de evitar, en primer lugar la duplicación del esfuerzo, y en segundo lugar, las acciones inconsistentes o conflictivas. Generalmente, se aplica a un entorno multiagente para identificar los conflictos y planificar sobre ellos. Se han presentado dos maneras de abordar esta cuestión. Una es la *planificación centralizada* y otra la *planificación distribuida*. En la planificación centralizada de sistemas multiagente hay siempre un agente que conoce las actividades de

todos los otros agentes. Uno de los métodos desarrollados consiste en que inicialmente cada agente construye sus planes individuales, y entonces un agente central los analiza para tratar de encontrar posibles interacciones, como pueden ser conflictos entre agentes. El agente central realiza un "análisis de seguridad" para determinar qué interacciones pueden derivar en conflictos y lo comunica a los agentes para que así puedan evitarlos. Otro método de planificación centralizada para sistemas multiagente consiste en que cada agente comunica sus planes a un agente coordinador, el cual construye un plan general que especifica las acciones planeadas por todos los agentes. La ventaja de este método es que evita generar planes inconsistentes, aunque necesita realizar varias planificaciones para alcanzar una solución aceptable. En sistemas multiagente, donde la planificación se realiza de modo distribuido no hay un agente con una visión global de las actividades del conjunto. En este caso, detectar y resolver las interacciones entre agentes es mucho más difícil. La estrategia que se utiliza consiste en proporcionar a cada agente una capacidad de planificación que tenga en cuenta las actividades de los demás.

3.5.2 Organización del sistema

Históricamente, los sistemas de resolución de problemas basados en la DAI han sido poco flexibles cuando han tenido que adaptar su comportamiento global, por esto es básico desarrollar un método de trabajo adecuado para que los sistemas sean más adaptables. Cuando se desarrolla un sistema multiagente hay que tener en cuenta el conocimiento necesario para construir agentes que puedan reconfigurar sus acciones e interacciones con los otros, cuando cambie el entorno. Por este motivo necesitamos saber cómo están organizados estos sistemas. Pero antes de estudiarlo será útil tener una buena idea de lo que significa *organización*, distinguiéndolo de la noción de *estructura*.

En un entorno multiagente, la estructura es un patrón de las relaciones de información y control que existen entre los agentes, y de la distribución de las capacidades que tienen para resolver un problema. Bajo este punto de vista, hay que asegurarse que los agentes reúnen las condiciones que son esenciales para resolver problemas, que son:

- covertura: todas las partes del problema deben ser contempladas en la capacidad de resolver problemas de, al menos, un agente.
- conectividad: los agentes deben interactuar de tal manera que las actividades que han de ser cubiertas sean desarrolladas e integradas en la solución final.
- capacidad: la cobertura y la conectividad han de alcanzarse dentro de las limitaciones de los métodos de comunicación, y así garantizar la fiabilidad a las especificaciones del grupo.

En el contexto de la DAI, una organización no se pensaría como una relación estructural entre un conjunto de agentes. El concepto de organización se encuentra dentro de las creencias, intenciones y compromisos de los agentes . También se puede definir como un conjunto de agentes con compromisos mutuos, compromisos globales, creencias comunes y que, eventualmente, unen sus intenciones actuando juntos para alcanzar un objetivo. Nociones como creencia, intención y compromiso vienen dadas por el grado de cooperación que existe entre los agentes, y por el abanico de posibles formas de comunicación que se ofrece a un agente para intercambiar creencias, intenciones y compromisos. Las siguientes secciones tienen la intención de profundizar en estas dos dimensiones.

3.5.3 Cooperación entre agentes

El grado de cooperación que hay entre agentes puede variar desde una cooperación completa hasta una antagónica u hostil. En el primer caso se paga un alto costo por la total comunicación entre agentes, pero en el segundo puede ocurrir que unos agentes bloqueen los objetivos de otros.

La cooperación total entre los agentes únicamente se puede encontrar en sistemas de resolución de problemas distribuida y cooperativa , de los que ya hemos hablado anteriormente. En estos sistemas, los agentes pueden trabajar juntos para tratar de resolver un problema que va más lejos de sus capacidades individuales. La CDPS es particularmente útil cuando cada agente tiene conocimiento específico y no tiene una idea clara de lo que puede hacer o la información que puede intercambiar con los otros agentes.

Algunos afirman que la cooperación total en los MAS no se puede conseguir pues en el mundo real los agentes pueden tener objetivos que entren en conflicto. Esto significa que es necesario siempre un proceso de resolución de conflictos, que puede realizarse mediante la negociación entre los agentes del sistema, o bien, mediante un Mediador, esto es, un tercer agente que hace de árbitro entre el resto .

3.5.4 Comunicación entre agentes

Las soluciones al problema de la comunicación en los MAS puede ir desde los sistemas que no se comunican hasta los que mantienen conversaciones:

- No hay comunicación: los agentes han de inferir los planes de los demás sin establecer ningún tipo de comunicación. Este modelo de comunicación es el que consigue mejores resultados cuando los objetivos de los agentes no están interrelacionados y por lo tanto no pueden entrar en conflicto . De todos modos, la carencia de métodos de comunicación representa una dificultad mayor para el sistema, será necesario por lo tanto, trabajar con incertidumbre y probabilidades, dotar al agente de complejos sistemas de razonamiento acerca de los demás, y de unos mayores recursos computacionales para realizar estas inferencias.

- Primitivas de comunicación: en este caso la comunicación está restringida a un número de señales fijas con una interpretación establecida de antemano . El problema de esta estrategia es que el limitado número de señales limita la cooperación entre agentes.

- Intercambio de planes e información: dos agentes pueden intercambiarse sus respectivos planes, y de esta manera cada uno puede adaptar sus estrategias. Esto presenta algunos inconvenientes. El primero es el alto costo computacional del intercambio, que según el caso, podría no ser soportado por ningún medio de comunicación. El segundo es que no queda garantizado que el plan resultante pueda ser aceptado por la base de conocimiento del agente .

- Intercambio por medio de un tablero: el método del tablero o blackboard , es muy usado en la inteligencia artificial como una manera de compartir memoria y conocimiento. Los agentes pueden escribir mensajes, dejar resultados parciales o encontrar información en un tablero que todos saben donde está. Ahora bien, puede haber problemas dependiendo de si el flujo de información está dirigido por los datos o por los objetivos.

- Intercambio de mensajes: esta estrategia consiste en agentes que actúan en respuesta al procesamiento de una comunicación. Las acciones que pueden ejecutar estos agentes son: enviarse una comunicación a ellos mismos, crear otros agentes del mismo tipo, y

especificar el reemplazo del comportamiento, que describe el nuevo estado después de la comunicación.

- Comunicación en alto nivel: el diálogo entre agentes permite la generación e interpretación de palabras o declaraciones, con el objetivo de comunicar la información que el emisor tiene en mente -creencias, compromisos, intenciones- pretendiendo llevar al receptor al mismo estado mental que el emisor .

3.5.5 Dinámica de la organización

En una organización de agentes es importante el comportamiento coherente de cada agente y la coordinación en la negociación de los recursos distribuidos. En este apartado vamos a ver estos dos aspectos.

Comportamiento coherente

Suponemos que tenemos un grupo de agentes cooperando en alguna tarea. La cuestión es saber cómo asegurar que los agentes actúen coherentemente cuando toman decisiones o realizan alguna acción, adaptando los efectos globales de las decisiones locales en orden a evitar interacciones perjudiciales. Es decir, ¿cómo pueden los agentes alcanzar una coherencia global respecto al sistema?. La información comunicada entre los agentes afecta a la coherencia. Hay que ver, entonces, cómo ha de ser la información que los agentes intercambian :

- Relevante: la relevancia de un mensaje sirve para medir la cantidad de información que es consistente con la solución derivada por el grupo de agentes. Un mensaje irrelevante puede llevar a un agente a desperdiciar trabajo o a intentar integrar en su conocimiento información inconsistente.

- Oportuna: la conveniencia de enviar un mensaje puede servir para medir hasta que punto el mensaje puede influir en la actividad actual del agente que lo recibe.

- Completa: que un mensaje sea o no lógicamente completo mide la fracción de la solución completa que el mensaje representa.

En consecuencia, la coherencia puede entenderse como el grado de bondad en el cual el sistema se comporta como una unidad. Por tanto, en un sistema de DAI se buscará [Bon88]:

- una solución global satisfactoria

- eficiencia

- capacidad de resolver diferencias

- un comportamiento elegante ante los posibles fallos

Coordinación y cooperación

Los agentes de un MAS pueden coordinar sus capacidades individuales y para eso han de buscar una técnica apropiada para trabajar juntos en armonía. Esta técnica depende de la distribución de los recursos y de la autonomía local de cada agente. Hay que tener en cuenta que coordinación no implica una cooperación total, mientras que la coordinación y la coherencia están parcialmente relacionadas. En la inteligencia artificial distribuida se usan los procesos de negociación para coordinar a un grupo de agentes. Ahora bien, el concepto

de negociación es difícil de definir. Se han propuesto algunas definiciones, que se acercan bastante a lo que buscamos, aunque tienen el problema de que recogen demasiados aspectos del concepto humano de negociación.

Aplicando la metáfora de la negociación en los humanos , un agente que necesita ayuda debe descomponer su problema en subproblemas, anunciar los subproblemas a los agentes del sistema, recoger las ofertas de otros agentes y adjudicar el subproblema al que ha presentado la mejor oferta.

Hasta este momento hemos visto que los agentes han de comunicarse de alguna manera y que esta comunicación puede ser muy variada. Llegados a este punto, es interesante estudiar cómo pueden ser los lenguajes de comunicación y qué trabajos se han realizado al respecto.

3.5.6 Comunicar agentes

Queremos construir sistemas con programas que puedan interactuar e intercambiar información y servicios con otros programas, para así resolver problemas que un único programa no puede. Esta capacidad de interacción es necesaria en los sistemas multiagente. Algunos autores no han dudado en definir a los agentes como componentes software que se comunican mediante mensajes en un lenguaje de comunicación de agentes.

Cuando abordamos el tema de la comunicación entre agentes, aparecen algunas cuestiones importantes:

- ¿Cómo es un lenguaje de comunicación apropiado?
- ¿Cómo construir agentes capaces de comunicarse en este lenguaje?
- ¿Qué arquitecturas de comunicación contribuyen a la comunicación?

Hay varios niveles en los que los sistemas basados en agentes deben tener un cierto acuerdo, al menos en lo que se refiere a sus interfaces de comunicación:

- Transporte: cómo envían y reciben mensajes los agentes
- Lenguaje: qué significan los mensajes
- Política: cómo estructuran sus conversaciones los agentes
- Arquitectura: cómo conectar sistemas

3.5.7 Lenguajes de comunicación

La estandarización del lenguaje de comunicación facilita la construcción de software que puede interoperar mediante la separación de la implementación y de la interfaz. Los problemas surgen cuando es necesario que unos agentes con un lenguaje de comunicación interaccionen con agentes que usan otro lenguaje. Puede haber inconsistencias en el uso de la sintaxis o del vocabulario. Un agente puede usar una expresión para tratar de significar algo, mientras que otro puede usar la misma con la intención de expresar un significado distinto. También puede haber incompatibilidades, ya que agentes distintos pueden usar diferentes expresiones para significar la misma cosa.

La ingeniería de MAS ataca este problema proponiendo lenguajes de comunicación universales, donde las inconsistencias y las notaciones arbitrarias han sido eliminadas. Podemos encontrar dos maneras de abordar el diseño de un lenguaje: el enfoque procedural y el declarativo.

- Enfoque procedural: se basa en la idea de que la mejor forma de modelizar la comunicación es mediante el intercambio de directivas procedurales. Los lenguajes utilizados permiten a los agentes intercambiar no solo órdenes individuales sino procedimientos enteros, que implementan la manera de alcanzar objetivos. Estos lenguajes suelen ser sencillos y de ejecución eficiente. El enfoque procedural tiene también algunos inconvenientes. Por ejemplo, los procedimientos intercambiados pueden necesitar información sólo disponible para algunos agentes. Por esto, es mucho más difícil fusionar la información procedural que la declarativa.
- Enfoque declarativo: se fundamenta en la idea de que la comunicación puede ser modelada mejor como el intercambio de información declarativa. Para que este enfoque sea realmente útil, debe ser suficientemente expresivo para comunicar información de varios tipos, incluyendo procedimientos. El lenguaje debe ser razonablemente compacto, y debe asegurar que la comunicación es posible sin una excesiva carga respecto a lenguajes especializados.

Vamos a ver ahora el trabajo desarrollado en el *ARPA Knowledge Sharing Effort*, donde se ha construido un entorno de trabajo sobre lenguajes de comunicación entre agentes que sigue el enfoque declarativo. El lenguaje propuesto es el *ACL*.

3.5.8 ACL

El lenguaje de comunicación entre agentes ACL esta formado por tres componentes:

- Vocabulario
- Lenguaje "interno": KIF
- Lenguaje "externo": KQML

Un mensaje en ACL es una expresión KQML, en la que los argumentos son términos o sentencias KIF, formadas por palabras del vocabulario del ACL.

Vocabulario

El vocabulario de un ACL es un diccionario abierto de palabras apropiadas para áreas de aplicaciones comunes. Cada palabra del diccionario tiene una descripción en inglés para ser usada por los humanos, y cada palabra tiene anotaciones formales, escritas en KIF, para ser usadas por los programas. El diccionario está abierto para que sea posible añadir nuevas palabras referentes a áreas de aplicación existentes o a nuevas áreas. La existencia de un diccionario no implica que haya una sola manera de describir el área de aplicación. El diccionario puede tener varias ontologías por área. Más adelante veremos qué se entiende por ontología en el ámbito de la comunicación entre agentes.

KIF

El lenguaje KIF es una versión en notación prefija del cálculo de predicados de primer orden, con algunas extensiones que le proporcionan expresividad. Es posible diseñar toda una estructura de comunicación en la que los mensajes tienen el formato de sentencias KIF. Gracias a la independencia del contexto del KIF, cada mensaje puede incluir información implícita acerca del emisor, del receptor, del tiempo, de la historia del mensaje, etc. Se puede aumentar la eficiencia de la comunicación suministrando un estrato lingüístico en el que se tiene en cuenta el contexto. Esta es la función del KQML.

KQML

KQML es un lenguaje y un protocolo para el intercambio de información y conocimiento. Es al mismo tiempo un formato de mensaje y un protocolo de intercambio de mensajes, que permite compartir conocimiento entre agentes, en tiempo de ejecución. Se puede usar KQML como lenguaje para que un programa pueda interactuar con un sistema inteligente, como para que dos o más sistemas inteligentes compartan conocimiento en tareas de resolución de problemas de modo cooperativo.

KQML se basa en un conjunto extensible de performativas, que definen las operaciones permitidas que los agentes pueden ejecutar en la base de conocimiento de los otros agentes. Además, el lenguaje KQML proporciona una arquitectura básica para la compartición de conocimiento mediante un tipo de agentes mediadores llamados facilitadores de comunicación (communication facilitators), que coordinan las relaciones e interacciones entre los otros agentes.

3.5.9 Ontologías

La palabra ontología parece generar mucha controversia en las discusiones de IA. Tiene mucha historia en la filosofía, donde ontología hace referencia al sujeto de la existencia, es decir, al ser. Ontología se confunde muchas veces con epistemología, que hace referencia al conocimiento y al modo de conocer.

En el ámbito del intercambio de conocimiento, el término ontología se usa para significar una especificación de una conceptualización . Esto es, una ontología es la descripción de los conceptos y relaciones que pueden existir para un agente o para una comunidad de agentes. Esta definición de ontología es consistente con el uso del término como conjunto de definiciones de conceptos, pero más generalizada; y adopta un sentido distinto del término usado en filosofía.

Se han diseñado y construido ontologías con el propósito de compartir y reutilizar conocimiento. En este ámbito, las ontologías son especificaciones que sirven para establecer compromisos ontológicos. Más adelante veremos la definición formal de compromiso ontológico. Por razones prácticas, las ontologías se escriben como un conjunto de definiciones en un vocabulario formal. Aunque ésta no sea la única manera de especificar una conceptualización tiene algunas propiedades interesantes para compartir conocimiento entre programas inteligentes.

Un compromiso ontológico es un acuerdo para usar un vocabulario, que es consistente, aunque no completo, respecto a la teoría especificada en la ontología. Se trata entonces de definir un vocabulario común en donde se representa el conocimiento compartido.

En resumen, podemos construir agentes comprometidos con ontologías, y diseñamos ontologías con las que los agentes pueden compartir conocimiento.

3.6 Comunicación entre agentes mediante KQML

El lenguaje KQML es realmente interesante en la comunicación entre programas basados en agentes, puesto que son programas autónomos y asíncronos. En esta sección vamos a ver cómo trabajar con KQML, viendo la sintaxis y la semántica del lenguaje, y el tipo de comunicación que se asume para los agentes.

3.6.1 Requisitos de transporte

La especificación de KQML no trata de estandarizar cómo ha de ser la infraestructura de transporte de mensajes, ya que normalmente esto depende del lenguaje de programación y de la red. Lo que sí define es una abstracción del nivel de transporte:

- Los agentes están conectados mediante enlaces de comunicación unidireccionales
- Los enlaces de comunicación pueden tener un poco de retraso asociado
- Cuando un agente recibe un mensaje, sabe desde qué link de entrada ha llegado el mensaje
- Cuando un agente envía un mensaje puede dirigirlo a un enlace de salida concreto
- Los mensajes dirigidos a un destino único han de llegar en el orden en que fueron enviados
- La entrega de mensajes es segura y fiable

Esta abstracción sobre el nivel de transporte puede ser implementada de diversas formas. Por ejemplo, los enlaces podrían ser conexiones TCP/IP sobre Internet, que están activos solamente mientras dura la transmisión de un mensaje. Los enlaces podrían ser caminos fijos de correo electrónico. También pueden ser conexiones IPC entre procesos distintos ejecutados en máquinas UNIX. Dejando de lado cómo está implementada la comunicación, KQML considera que, a nivel de agentes, la comunicación es un paso de mensajes punto a punto, y necesita que el método de comunicación sea un paso de mensajes punto a punto.

3.6.2 Sintaxis KQML

El lenguaje KQML es independiente del formato de la información. A un mensaje KQML también se le llama performativa o actuación. Una performativa se representa como una cadena ASCII, con la sintaxis propia de KQML. La sintaxis de KQML se puede considerar como una subclase de la notación prefija del Common Lisp. No obstante, a diferencia de las del Lisp, los parámetros en las actuaciones están indexados por palabras clave, que pueden usarse en cualquier orden. Las palabras clave, llamadas nombres de parámetros (parameter names), deben empezar por dos puntos (:), y deben ir delante de sus correspondientes valores, llamados en la especificación valores de parámetros (parameter values).

En la Tabla 3.3 podemos ver la sintaxis de KQML en formato BNF (BACKUS-NAUR FORM).

3.6.3 Semántica KQML

El modelo semántico que subyace en KQML es un contexto simple y uniforme para los agentes, que permite ver las capacidades de otros agentes. Desde fuera, cada agente, aparece como si manejara una base de conocimiento (KB). Entonces, la comunicación entre agentes consiste en mirar el contenido de las KB. La implementación de un agente no siempre -y no necesariamente- es con una base de conocimiento. Un agente puede utilizar un simple sistema de base de datos o una estructura de datos propia. La capa de comunicación del agente puede traducir estas representaciones en una abstracción de una base de conocimiento, para ayudar a los demás. Entonces, se dice que cada agente organiza una base de conocimiento virtual (VKB).

Cuando se definen las performativas, es útil clasificar el contenido de la VKB en dos categorías:

```
<performativa> ::= (<word> {<whitespace> :<word> <whitespace>
<expression>}*)

<expression> ::= <word> | <quotation> | <string> | (<word>
{<whitespace> <expression>}*)

<word> ::= <character> <character> *

<character> ::= <alphabetic> | <numeric> | <special>

<special> ::= < | > | = | + | - | * | / | & | ^ | ~ | _ | @ | : |
. | ! | ? | $ | %

<quotation> ::= '<expression> | ,<comma-expression>

<comma-expression> ::= <word> | <quotation> | <string> |
,<comma-expression> (<word> {<whitespace> <comma-expression>}*)

<string> ::= "<stringchar>*" | #<digit><digit>*"<ascii>"

<stringchar> ::= \<ascii> | <ascii>-\-<double-quote>
```

Tabla 3.3: *Sintaxis KQML en BNF.*

- Creencias
- Objetivos

Las creencias de un agente codifican información sobre él mismo y sobre su entorno, incluyendo las VKB de otros agentes. Los objetivos codifican los estados del entorno que el agente quiere alcanzar cuando actúa.

Los agentes se comunican entre sí sobre el contenido de su VKB y del contenido de las de los otros, usando KQML. Pero la codificación del contenido de las distintas VKB puede usar gran variedad de lenguajes de representación. Esto es, la performativa `tell` sirve para indicar que un string concreto se encuentra en el almacén de creencias de un agente, donde el string puede estar representado con un lenguaje de representación distinto del KQML

3.6.4 Performativas

Tal como hemos visto, un mensaje KQML es una performativa. Una performativa, cuando es enviada, invita al receptor realizar alguna acción. Se ha definido un sustancial número de performativas en términos de lo que significan sobre el conocimiento del que las envía. Las performativas definidas pueden no ser suficientes para implementar un sistema multiagente. Del mismo modo, los agentes no necesitan soportar el conjunto entero de performativas definidas. En realidad, lo que se espera es que los agentes utilicen solo un pequeño conjunto. Los agentes pueden usar performativas que no aparecen en la especificación del lenguaje. Las nuevas performativas habrían de definirse de modo preciso y en el mismo estilo de las de la especificación.

Los nombres de las performativas, en la especificación del lenguaje, son palabras reservadas; cuando una aplicación usa las performativas de modo inconsistente de acuerdo con la definición en la especificación, entonces no es del tipo KQML.

3.6.5 Parámetros de las performativas

Tal como hemos dicho antes, las performativas tienen parámetros que se identifican por palabras clave. En este apartado, vamos a ver el significado de los parámetros más comunes.

Los parámetros aquí descritos son palabras clave, en el sentido que todos los mensajes que usen parámetros con estas palabras clave deben ser consistentes con las definiciones de la Tabla 3.4

Palabra clave	Significado
:content	Información sobre la que la performative expresa una actitud
:force	El emisor nunca negará el significado de una performative
:in-reply-to	La etiqueta esperada en una respuesta
:language	El nombre del lenguaje de representación del parámetro `:content`
:ontology	El nombre de la ontología -conjunto de definiciones de términos- usada en el parámetro `:content`
:receiver	El receptor del mensaje
:reply-with	El receptor espera una respuesta con etiqueta
:sender	El emisor del mensaje

Tabla 3.4: *Parámetros de un mensaje con sus significados.*

Podemos ver las palabras reservadas para las performativas, en la Tabla 3.5.

En este capítulo hemos visto qué se entiende actualmente por "agente" y que los sistemas multiagente proporcionan un nuevo método de resolver problemas que hasta antes de su desarrollo no existía. Hemos visto también las características que debe tener un lenguaje de comunicación entre agentes y una propuesta de lenguaje que es el KQML. En el siguiente capítulo veremos el área de aplicación donde los agentes han encontrado mayor aceptación: Internet. Se realizará una visión general sobre Internet y se estudiarán los agentes que habitan en Internet, sus características y algunos ejemplos.

Nombre	Significado
Achieve	S wants R to do made something true of their environment
advertise	S is particulary-suited to processing a performative
ask-about	S wants all relevant sentences in R's VKB
ask-all	S wants all of R's answers to a question
ask-if	S wants to know if the sentence is in R's VKB
ask-one	S wants one of R's answers to a question
break	S wants R to break an established pipe
broadcast	S wants R to send a performative over all connections
broker-all	S wants R to collect all responses to a performative
broker-one	S wants to get help in responding to a performative
deny	the embedded performative does not aplly to S (anymore)
delete	S wants R to remove a ground sentence from its VKB
delete-all	S wants R to remove all matching sentences from its VKB
delete-one	S wants R to remove on matching sentence from its VKB
discard	S will not want R's remaining responses to a previous performative
eso	end of a stream of responses to an earlier query
error	S considers R's ealier message to be mal-formed
evaluate	S wants R to simplify the sentence
forward	S wants R to route a performative
generator	same as a standby of a stream-all
insert	S asks R to add content to its VKB
monitor	S wants updates to R's response to a stream-all
next	S wants R's response to a previously-mentioned performative
pipe	S wants to route all further performatives to a another agent
ready	S is ready to respond to R's previously-mentioned performative
recommend-all	S wants all names of agents who can respond to a performative
recommend-one	S wants the name of a agent who can respond to a performative
recruit-all	S wants R to get all suitable agents to respond to a performative
recruit-one	S wants R to get another agent to respond to a performative
register	S can deliver performatives to some named agent
reply	communicates an expected reply
rest	S wants R's remaining responses to a previously-mentioned performative
sorry	S cannot provide a more informative reply
standby	S wants R to be ready to respond to a performative
stream-about	multiple-response version of ask-about
stream-all	multiple-response version of ask-all
subscribe	S wants updates to R's respnse to a performative
tell	the sentence in S's VKB
transport-adress	S associates symbolic name with transport adress
unregister	a deny of a register
untell	the sentence is not in S's VKB

Tabla 3.5: *Palabras reservadas para las performativas.*

Capítulo 4

Agentes en Internet

En el Capítulo 3 hemos presentado los conceptos de agente y de sistema multiagente. Vamos a ver ahora el campo donde actualmente los sistemas basados en agentes han encontrado mayor desarrollo. Se trata de Internet.

4.1 Internet y la World Wide Web

Internet nace como consecuencia de un proyecto desarrollado a finales de los años 60. Este proyecto consistía en la construcción de una red experimental para la DARPA (Defense Advanced Research Projects Agency) del departamento de defensa de los EE.UU. El objetivo del proyecto era la construcción de un sistema de comunicación entre ordenadores altamente flexible y dinámico, que permitiera utilizar cualquier tipo de medio y tecnología de transmisión y que siguiera funcionando incluso ante la eventualidad de la destrucción de partes de la red. Así, en 1969, nació la red ARPANET, auténtica precursora de la posterior Internet, interconectando cuatro grandes ordenadores ubicados en distintas localizaciones.

La red ARPANET creció lentamente durante los años 70 hasta llegar a conectar unos 100 ordenadores a principios de los 80. Sin embargo, durante todos esos años, sirvió como banco de investigación, desarrollo y prueba de los pilares sobre los que se apoya Internet: las normas y lenguajes comunes que permiten la comunicación entre los distintos ordenadores conectados, conocidos como familia de protocolos TCP/IP. La adopción de éstos dentro de ARPANET en 1982 supuso un hito decisivo.

A mediados de los 80 se producen factores clave para el posterior desarrollo de Internet. Numerosos fabricantes empiezan a sacar al mercado equipos que "hablan" TCP/IP, convirtiéndolo en el estándar de facto para la intercomunicación de ordenadores. Así mismo, la proliferación de redes de área local, que interconectan ordenadores en el ámbito de un edificio o campus, hace cambiar la tendencia de conectar únicamente al exterior los grandes ordenadores de las instituciones, por la necesidad de conectar redes locales enteras, facilitando así el acceso simultáneo al exterior a multitud de equipos informáticos por organización.

Un último factor decisivo fue el nacimiento en 1986 de la red NSFnet. Con el objeto de facilitar el acceso a toda la comunidad científica americana a cinco grandes centros de supercomputación, la National Science Foundation, ante los impedimentos burocráticos para usar la red ARPANET, decidió crear una red propia que acabaría convirtiéndose en la auténtica espina dorsal de Internet. Dado su carácter abierto a toda la comunidad científica investigadora, la NSFnet desencadenó una explosión de conexiones, sobre todo por parte de las universidades. Así pues, aunque el objetivo inicial de la NSFnet era la compartición de costosos recursos de supercomputación, pronto las organizaciones conectadas descubrieron que disponían de un medio inmejorable de comunicación y colaboración entre ellas. El éxito

de NSFnet fue tal que hizo necesarias sucesivas ampliaciones de la capacidad de sus líneas troncales, teniendo que ser multiplicada por 30 cada tres años: 56.000 bits por segundo (bps) en 1986, 1.5 millones de bps en 1989 y 45 millones de bps en 1992. La NSFnet ha sido y todavía es una de las piezas más importantes dentro del gran mecano que es Internet .

Desde finales de los años 80 Internet ha venido experimentando un crecimiento exponencial en casi todos sus parámetros:

- **El número de usuarios de Internet crece rápidamente**

  En los primeros años de existencia de Internet, la mayoría de usuarios eran investigadores y algunos pocos proveedores públicos. Esos usuarios tenían importantes conocimientos sobre ordenadores, redes de ordenadores, y sobre el modo de acceder y usar los diversos servicios de Internet.

  Hoy en día, la mayoría de los usuarios conectados a Internet son noveles en el campo de la informática. No tienen grandes conocimientos sobre las posibilidades y técnicas de las redes de ordenadores, ni sobre Internet y sus servicios.

- **El número de proveedores de servicios e información en Internet ha crecido rápidamente**

  Un creciente número de empresas y otras entidades, como los gobiernos, han empezado a ofrecer servicios sobre Internet, normalmente a través de la *World Wide Web*.

- **El crecimiento del número de gente conectada a Internet está dejando atrás el también creciente ancho de banda**

  Los usuarios de Internet han tenido que aprender a convivir con los inconvenientes propios de esta sobrepoblación de la red. Las consecuencias son, entre otras, la imposibilidad de conectarse con algunos servidores o la espera, casi infinita, necesaria para la transferencia de información.

- **Desde 1995 la World Wide Web es el servicio más popular de Internet**

  La facilidad de uso y la imagen atractiva de la WWW ha colocado a éste en el primer lugar de las estadísticas de los servicios más usados en Internet. La WWW se puede utilizar para acceder a casi todos los servicios que se ofrecen en la red. Sirve para acceder a servidores FTP o Gopher de un modo más sencillo del que proporcionan estos servicios. Desde la WWW se puede acceder a las noticias de Usenet o al buzón de correo electrónico del usuario.

  A través de la World Wide Web, los usuarios pueden acceder a una cantidad de información prácticamente inacabable. Esta es una de las principales razones que explican por qué las grandes empresas están empezando a ofrecer servicios e información a través de la WWW: cuando la información interesante es hábilmente combinada con información comercial, una empresa puede alcanzar un altísimo grado de difusión entre los usuarios de Internet, y por lo tanto pasan a ser todos ellos clientes potenciales de dicha empresa.

- **Desde 1995 han surgido nuevas tecnologías**

  Java y VRML son algunas de estas nuevas tecnologías. Java permite la creación de programas de propósito general, independientes de la plataforma de ejecución. Por su parte, VRML es un lenguaje de programación que puede usarse para ampliar los documentos HTML. VRML hace posible la creación de entornos virtuales de tres dimensiones que los usuarios pueden explorar.

Como vemos, el desarrollo más notable de Internet es la World Wide Web. La popularidad de la red parece sugerir la necesidad de construir interfaces sencillas y de fácil manejo para los usuarios. Interfaces que hagan una abstracción y oculten las diferentes técnicas necesarias para realizar ciertas tareas sobre Internet.

Los desarrolladores de software han entrado en esta tendencia, construyendo web browsers cada vez más completos y versátiles. Pueden usarse como browser de la WWW, como lectores de noticias de Usenet, como programas de gestión del correo electrónico y como editores de páginas HTML.

Los agentes pueden ofrecer también funcionalidades como las de los web browsers. Incluso pueden hacerlo mejor si se mejoran algunas características -como la independencia del hardware y software- y si se añaden nuevas funcionalidades. Los agentes pueden ofrecer una gran ayuda a los usuarios de Internet, pues no sólo pueden ser usados como buscadores de información, sino que pueden proporcionar un modo apropiado y preciso de construir interfaces para interaccionar con la red.

4.2 Areas de aplicación de los agentes

Hemos visto que no hay un acuerdo generalizado sobre el uso de la palabra "agente", pues se trata en realidad de un paraguas que abarca un campo heterogéneo de investigación y desarrollo. Algunos investigadores, en respuesta a esta carencia de definición, han inventado algunos sinónimos para nombrar a los agentes, añadiendo todavía más confusión a la definición de agente. Algunos de estos sinónimos son:

- *softbots* (robots de software)
- *knowbots* (robots basados en conocimiento)
- *taskbots* (robots basados en tareas)
- *searchbots* (robots de búsqueda)
- *userbots* (robots de usuario)
- *robots*
- *personal agents* (agentes personales)
- *personal assistants* (asistentes personales)

Se han definido ocho grandes áreas de aplicación donde ahora, o en un futuro cercano, se utilizará la tecnología basada en agentes . Estas áreas son:

- Organización y mantenimiento de sistemas y redes
- Movilidad y accesos remotos
- Mensajería y Correo Electrónico
- Organización del acceso a la información
- Colaboración Workflow (flujos de trabajo) y organización administrativa
- Comercio Electrónico

- Interfaces de usuario adaptativas

Como veremos a continuación, estas áreas están todas o casi todas relacionadas de algún modo con las redes de ordenadores y con el actual paradigma de red: *Internet*.

- **Organización y mantenimiento de sistemas y redes**

  La incorporación del protocolo cliente-servidor a los sistemas informáticos ha supuesto un incremento de la complejidad de mantenimiento, especialmente en el área de las redes de ordenadores. Los administradores de estos sistemas necesitan herramientas que de modo simple les ayuden a administrar el sistema ante la creciente complejidad de éstos.

  En la administración de redes y sistemas informáticos se utilizan desde hace bastante tiempo sistemas semejantes a los agentes que realizan funciones fijas. Estos sistemas se denominan *daemons*. Un daemon es parecido a un agente en el sentido que se activa cuando ha de realizar una función -reactividad-, pero carece de las características propias de los agentes como por ejemplo la autonomía.

  Los agentes pueden utilizarse para aumentar la automatización que ya existe en la administración de sistemas. Así mismo, pueden actuar en un alto nivel de abstracción filtrando y controlando acciones automáticas, también pueden detectar y reaccionar ante esquemas del comportamiento del sistema. Incluso pueden organizar configuraciones dinámicamente.

- **Movilidad y accesos remotos**

  Es cada vez más frecuente que los usuarios de ordenadores deseen aumentar su movilidad, no sólo quieren acceder a los recursos remotos de las redes de ordenadores, sino que quieren hacerlo olvidándose de las limitaciones propias de la red -el ancho de banda.

  Sin embargo, los agentes que habitan en la red pueden alcanzar estos deseos de movilidad mejor que los usuarios con sus ordenadores personales. Pues los agentes pueden satisfacer las peticiones de los usuarios, despreciando la sobrecarga de la red. Además, pueden procesar la información en el lugar donde se encuentra y enviar respuestas compactas al usuario.

- **Mensajería y Correo Electrónico**

  El software de gestión de correo electrónico es un área donde actualmente ya se usan agentes inteligentes. Los usuarios quieren gestores de correo que organicen y asignen prioridades a sus mensajes. También les gustaría que algunos mensajes pudieran ser respondidos de forma automática, etc.

  Los agentes pueden facilitar estas funciones proporcionando reglas para la gestión de los mensajes, que pueden ser especificadas y modificadas a lo largo del tiempo. En un futuro los agentes de mensajería podrán inferir reglas sobre el comportamiento del usuario e incorporarlas a su base de conocimiento.

- **Organización del acceso a la información**

  El estudio del acceso a la información constituye en la actualidad, una importante área de actividad, debido entre otras cosas al incremento de la popularidad de Internet y la incontable cantidad de información disponible para sus usuarios. Los agentes pueden ayudar a los usuarios no sólo en la realización de búsquedas y filtrado de la información, sino también pueden seleccionar, priorizar o categorizar la información.

- **Colaboración**

  La colaboración es un área en rápido crecimiento, donde los usuarios trabajan juntos en documentos compartidos, utilizando video-conferencias, o compartiendo recursos a través de la red. Un denominador común de la colaboración es la compartición de recursos. Otro es el trabajo en equipo.

  Los usuarios no sólo necesitan un sistema que proporcione una infraestructura que permita compartir información de modo robusto y fiable, sino que también necesitan herramientas y utilidades que les ayuden a construir y administrar los equipos de trabajo.

- **Workflow y organización administrativa**

  Workflow Management es la administración de procesos de empresa mediante un sistema informático. Los usuarios del sistema necesitan realizar los procesos lo más eficientemente posible y reducir de esta manera el coste de los agentes humanos. En este contexto, los agentes software pueden ser utilizados para automatizar procesos empresariales o la toma de decisiones.

- **Comercio Electrónico**

  El comercio en Internet es también un área en pleno desarrollo. Los clientes necesitan encontrar vendedores de productos y servicios, y junto a eso, información sobre estos productos y servicios, asistencia técnica y consejo. Por su parte, los vendedores necesitan encontrar clientes para sus productos, informar sobre los mismos y ofrecer asistencia técnica.

  Los agentes pueden ayudar al comercio electrónico de muchas maneras. Pueden ir de compras en lugar del usuario, estudiar precios, solicitar información sobre los productos y demás, y después volver a casa e informar al usuario. También pueden actuar como vendedores automatizando el proceso de venta o el de asistencia técnica.

- **Interfaces de usuario adaptativas**

  Aunque en estos últimos años las interfaces de usuario se han transformado en interfaces gráficas más intuitivas y fáciles de usar, para muchos continúan ofreciendo dificultades en lo que se refiere a su aprendizaje o utilización. Del mismo modo que las aplicaciones informáticas han mejorado, éstas necesitan interfaces de usuario más complejas. Las interfaces de usuario deberían aprender los hábitos del usuario y adaptarse de forma individualizada.

  Los agentes llamados *interface agents* pueden solucionar estos problemas. La actual tecnología de agentes permite a los sistemas monitorizar las acciones del usuario y desarrollar modelos o perfiles de usuario. De este modo se pueden construir agentes que adapten la interface del sistema al perfil de usuario o que cambien el modo de mostrar la información.

4.3 Tipos de agentes en Internet

En el Capítulo 3 vimos una clasificación de los agentes según su naturaleza. En esta sección vamos a ver qué tipo de agentes podemos encontrar actualmente en la red. Se intentará situar a los agentes existentes en diferentes clases. En este sentido hay diversas maneras de clasificar los agentes de Internet:

- Los agentes pueden ser clasificados según su movilidad, esto es, según su habilidad para moverse y cambiar de sitio en una red. Este criterio proporciona las clases de agentes estáticos y agentes móviles.

- Pueden clasificarse como deliberativos o como reactivos. Los agentes deliberativos son agentes que poseen un modelo interno de razonamiento de tipo simbólico. Su planificación está orientada a alcanzar cierta coordinación con otros agentes. Los agentes reactivos no tienen modelo interno del entorno, sino que actúan siguiendo un comportamiento del tipo estímulo/respuesta.

- Los agentes pueden clasificarse también según un conjunto de características ideales que los agentes podrían exteriorizar. Como mínimo, estas características son tres: autonomía, aprendizaje y cooperación. Está claro que estas no son todas las características deseables para los agentes, pero se trata aquí, de considerar un conjunto de propiedades esenciales para clasificar a los agentes de Internet. La autonomía hace referencia a que los agentes pueden actuar por sí mismos, sin necesidad de ninguna guía humana. Los agentes tienen estados internos y objetivos individualizados. Un elemento clave de la autonomía de los agentes es la capacidad de tomar la iniciativa en lugar de actuar simplemente como respuesta al entorno. La cooperación con otros agentes es de una importancia capital, es la razón de ser de tener sistemas con varios agentes en lugar de uno. Como hemos visto en el capítulo anterior, para que los agentes puedan cooperar entre si necesitan ser dotados de habilidades sociales, es decir, la habilidad de poder interactuar con otros agentes o con humanos, mediante algún lenguaje de comunicación. Finalmente, los agentes deben ser capaces de aprender acerca de sus acciones y reacciones ante el entorno. El aprendizaje puede hacer aumentar el rendimiento del sistema a lo largo del tiempo. Usamos las tres características mínimas en la Figura 4.1 para derivar cuatro tipos de agentes que incluimos en nuestra tipología: *agentes colaborativos*, *agentes de aprendizaje colaborativos*, *agentes de interfaz* y *agentes inteligentes*.

Hay que tener en cuenta que las distinciones utilizadas para construir la tipología, no son definitivas. Por ejemplo, cuando se habla de agentes que colaboran se pone mayor énfasis en la colaboración y autonomía que en el aprendizaje, esto no implica que los agentes que colaboran no puedan aprender. Del mismo modo, cuando hablamos de agentes asistentes personales -agentes que adaptan interfaces de usuario- se presta más atención a la autonomía y el aprendizaje que a la colaboración.

Como hemos dicho, el objetivo de esta sección es presentar una tipología y de este modo poder situar en esta clasificación algunos ejemplos de agentes que trabajan en Internet. Hay otras características de los agentes de Internet que se han considerado secundarias, pero que no deben ser olvidadas. Por ejemplo, la versatilidad, ¿tiene un agente un conjunto fijo de objetivos o bien puede enfrentarse a diversidad de tareas? También podemos preguntarnos ¿es este agente benévolo?, ¿es útil?, ¿es altruista?, ¿dice siempre la verdad cuando se comunica con otros agentes?.

En la Figura 4.2 se resumen y ordenan los distintos tipos de agentes para Internet dejando abierta la clasificación.

4.3.1 Agentes que colaboran

Los agentes que colaboran son agentes con características de autonomía y cooperación. Aunque pueden aprender, no es ésta su principal característica. Para tener un grupo coordinado de agentes es necesario que negocien para alcanzar acuerdos mutuos aceptables en algunas materias.

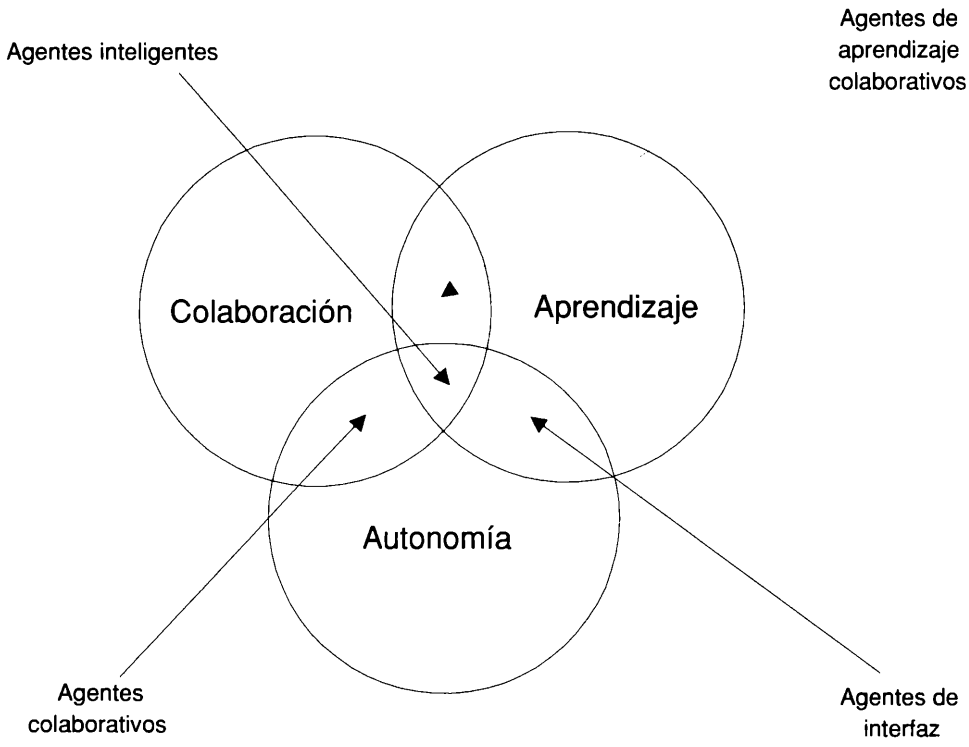

Figura 4.1: *Tipología de agentes.*

Las características fundamentales de estos agentes son la autonomía, habilidades sociales, un cierto grado de reacción y capacidad de tomar la iniciativa. Además, pueden exhibir racionalidad en su actuación, ser benévolos, decir siempre la verdad, etc.

Las razones que han llevado a construir Sistemas Basados en Agentes que colaboran son una especialización de las motivaciones de la Inteligencia Artificial Distribuida:

- Resolver problemas que exceden las capacidades de un solo agente individual.
- Ofrecer a los sistemas desarrollados hasta nuestros días la capacidad de interconectarse e interactuar con otros.
- Proporcionar soluciones a problemas de naturaleza distribuida
- Proporcionar soluciones a problemas que requieren datos distribuidos
- Aumentar la modularidad, velocidad, fiabilidad, flexibilidad

En el Capítulo 3 ya hemos hablado de sistemas multiagente con agentes que colaboran para alcanzar un objetivo común. En este capítulo veremos dos ejemplos de sistemas con agentes que colaboran realizando tareas sobre Internet.

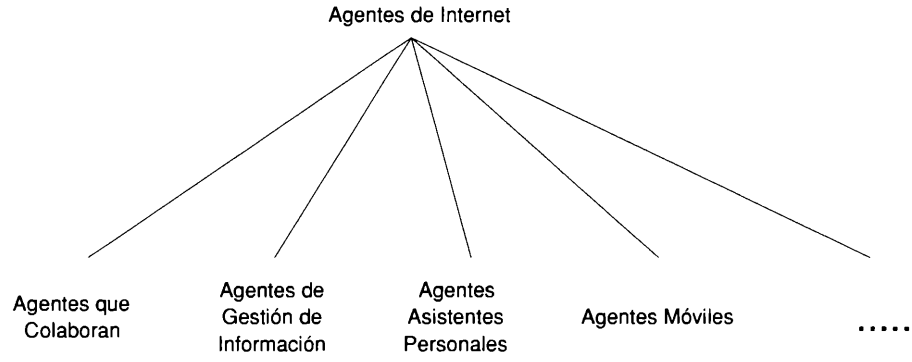

Figura 4.2: *Tipos de agentes en Internet.*

4.3.2 Agentes que gestionan información

Los agentes de gestión de información aparecieron como consecuencia de una creciente demanda de herramientas que ayudaran a manejarse con el crecimiento explosivo de la información disponible en Internet. Los agentes que gestionan información realizan tareas tales como manipular, manejar o recoger información de múltiples fuentes distribuidas en Internet.

La hipótesis que subyace en los agentes que gestionan información consiste en que un usuario se ahoga en la enorme cantidad de datos y, al mismo tiempo, se muere de hambre en lo que se refiere a la *información*. Los agentes pueden ayudar a mejorar, pero no eliminar este problema de exceso de información. Algunos autores no dudan en afirmar, que el mayor desafío de la era de la información es crear un interface de usuario simple donde la búsqueda y recuperación de información mediante agentes llegue a ser tan natural para la gente, como lo es llamar por teléfono o leer un periódico. Los agentes gestores de información han de estar dotados con las capacidades necesarias para conocer ***dónde*** buscar y ***cómo*** encontrar información y acumularla.

Aunque en un futuro se relacionen estrechamente los agentes asistentes personales, los agentes que adaptan interfaces, y los agentes gestores de información, es interesante que nosotros los veamos separados, para realizar nuestro estudio y poder ver su evolución. Los agentes gestores de información no tienen un modelo estándar de actuación, pueden ser estáticos o móviles, pueden ser no cooperativos o sociales, y pueden aprender o no.

En la Figura 4.3 podemos observar la representación de la actuación de un agente estático. Este agente muestra sus resultados desde un browser como Netscape. El agente utiliza los servicios de búsqueda disponibles en Internet, como son los rastreadores o *spiders*. Un rastreador es un programa indexador capaz de hacer una búsqueda en profundidad en la World Wide Web y almacenar información en un Sistema de Gestión de Base de Datos (SGBD). Además, también puede acceder al índice de URLs del WAIS. El agente, además de los rastreadores, puede usar otros buscadores o indexadores como Lycos o WebCrawler, actualmente podemos encontrar gran cantidad de rastreadores o buscadores en la WWW.

El agente de información del usuario, una vez lanzado a buscar alguna palabra clave, envía consultas a uno o varios buscadores de URLs, reúne los resultados y los envía al usuario. Este agente gestor de información es uno de los más sencillos que podemos encontrar. En este mismo capítulo podemos ver ejemplos de sistemas con agentes que buscan y gestionan

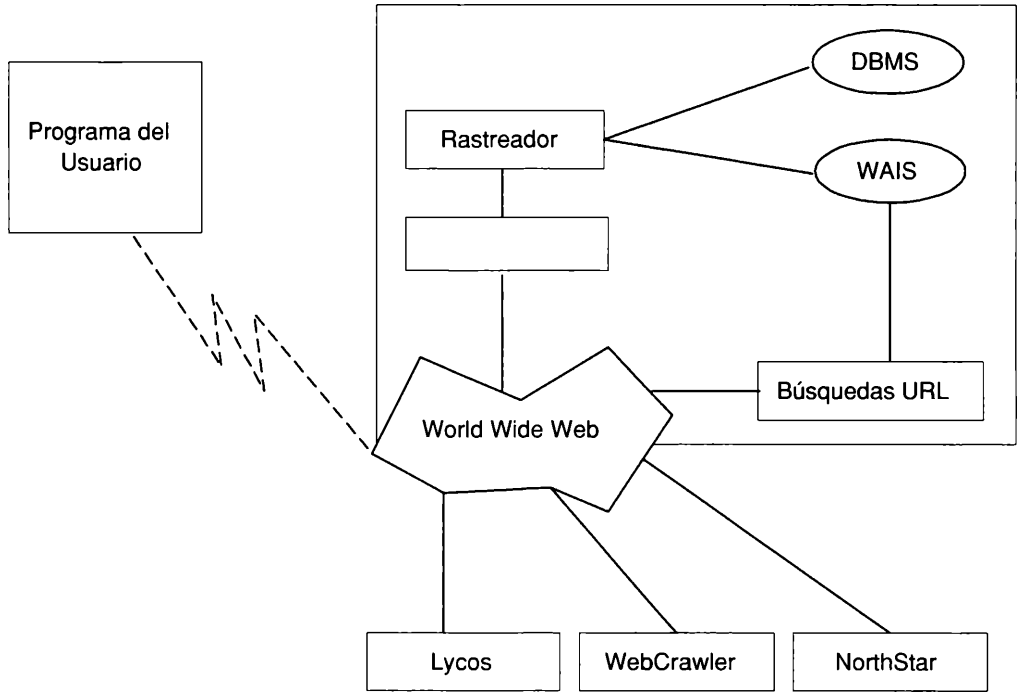

Figura 4.3: *Arquitectura de un agente que busca información.*

información cooperando unos con otros.

Un ejemplo de agente que busca y gestiona información en Internet, que ya puede considerarse un clásico, es el llamado *Internet Softbot* [Wel94]. El Internet Softbot es un agente que permite al usuario formular consultas en alto nivel y el agente es capaz de usar su base de conocimiento para realizar búsquedas e inferencias para determinar cómo resolver la consulta en Internet. El agente tiene en cuenta y tolera ambigüedades, omisiones y algunos tipos de error en la consulta del usuario. Los autores del Internet Softbot describen el proyecto haciendo un paralelismo entre el agente y un robot físico.

El principal problema de los agentes gestores de información, cuando estos son estáticos y construyen un índice que almacenan en una base de datos, es tener sus índices actualizados en un entorno enormemente cambiante como Internet. Por otra parte, los agentes estáticos pueden incorporar una interfaz que se adapte al usuario, que veremos en la siguiente sección. Los agentes que buscan y gestionan información pueden ser móviles. Esta movilidad les impide poder ofrecer al usuario una interfaz adaptativa pero en cambio son capaces de navegar y moverse por la WWW recogiendo información y construyendo índices en el mismo sitio en donde se encuentra la información. A los agentes gestores de información se les acusa de sobrecargar la red y colapsar las comunicaciones, aunque esto es cierto, algunos autores han propuesto unas normas de conducta para los agentes que trabajan en Internet, para de este modo reducir su influencia negativa sobre la red. Al final de este capítulo hablaremos de la ética de los agentes y del modo de reducir su influencia negativa en el funcionamiento de la red.

A continuación vamos a ver un ejemplo de un sistema formado por agentes que colaboran y por agentes que buscan y gestionan información.

Ejemplo: CIG Searchbots

El proyecto CIG Searchbots es un ejemplo sencillo de recuperación de información de modo cooperativo. Se trata de una aproximación a un sistema multiagente que recupera información. Los CIG Searchbots no son una herramienta de búsqueda en bases de datos. Para conseguir responder una consulta, diferentes agentes realizan una búsqueda en diversos lugares remotos, a través de la World Wide Web, mediante algunos métodos de búsqueda que pueden incluir el uso de *search engines*. Agentes expertos en el dominio determinan en qué lugares buscar y la manera de hacerlo para alcanzar la mejor solución de calidad con el coste mas bajo posible. Varios agentes trabajan juntos, cooperando, para localizar información relevante a la consulta.

El interés de los CIG Searchbots radica en objetivos de alto nivel, orientados a la recolección de información. Por tanto, no interesa la búsqueda dirigida por palabras clave en URL, tal como lo hacen Lycos o Infoseek. El objetivo en el que actualmente se está trabajando consiste en construir un sistema que permita a los usuarios especificar búsquedas del tipo: "*busca información en hojas de cálculo para Windows y OS/2*", o bien "*busca información sobre actualizaciones para Netscape*". Como respuesta a estos objetivos, los agentes han de planificar la actividad de recuperar información, combinar y unir los resultados, y enviarlos al usuario. En este modelo, los usuarios pueden especificar restricciones al uso de los recursos del sistema, como por ejemplo, la cantidad de concurrencia a utilizar, el tiempo de búsqueda, el número de lugares a visitar, etc.

Este modelo se diferencia de las búsquedas basadas en índices en que dirige y refina la búsqueda mediante la recuperación de información y las inferencias sobre ésta información. A diferencia de las búsquedas basadas en índices, las búsquedas CIG pueden ayudar al usuario en actividades de alto nivel como por ejemplo, usar la información para depurar la búsqueda. El sistema CIG searchbots no pretende reemplazar o ser una alternativa a los ya clásicos buscadores de Internet, sino que quiere ofrecer un complemento que permita realizar búsquedas en alto nivel, haciendo uso de Lycos u otros buscadores de la red. Las características más importantes del sistema CIG Searchbots son:

- Orientado a los objetivos
- Inferencia basada en información almacenada
- Multiagente o búsqueda paralela
- Estrategias de búsqueda adaptativas y dinámicas
- Búsquedas activas, no estáticas como en Lycos
- Asimilación de la información recogida

Vamos a ver ahora un ejemplo de búsqueda, con la intención de mostrar cómo trabaja el sistema y cómo su actuación se diferencia de la de un buscador basado en índices y palabras clave.

- El usuario especifica un objetivo, por ejemplo "busca información en hojas de cálculo para Windows o para OS/2", mediante un formulario HTML. Los agentes dispondrán de 10 minutos para completar la búsqueda y presentar los resultados al usuario. El usuario especifica que está dispuesto a ayudar en la búsqueda mediante razonamientos en alto nivel.
- El planificador selecciona una tarea que pueda satisfacer el objetivo y el programador paralelo construye un programa para las subtareas y procedimientos. La tarea a realizar está formada por tres subtareas básicas:

  - Buscar en las páginas WWW de IBM.
  - Buscar en las páginas WWW de Microsoft.
  - Localizar otros lugares de búsqueda mediante Infoseek.

- El programador paralelo contacta con los correspondientes agentes para que realicen simultáneamente la búsqueda de bajo nivel en la World Wide Web.

- Fase 1. Los agentes contratados empiezan sus respectivas tareas

  - Un agente recupera información de la página de prensa de IBM, con la intención de analizarla.
  - Un agente busca en el servidor WWW de Microsoft con la palabra clave "Excel", que ha sido identificada como el nombre de una hoja de cálculo en la base de conocimiento del planificador.
  - Un agente lanza la consulta "*spreadsheets windows os/2*" en Infoseek, recuperando una lista de URLs.

- Fase 2. Realización de inferencias sobre los resultados

  - El análisis de la página de prensa de IBM da como resultado: WordPerfect, Lotus y Novell. Estos son considerados como candidatos de ser productores de las hojas de cálculo que buscamos, y la información se envía al planificador.
  - El agente que está en las páginas de Microsoft pide ayuda a otro agente para buscar en el servidor de Microsoft.
  - La búsqueda en Infoseek da como resultado: Lotus, Computer Associates y Lighthouse Design. Esta información se envía al planificador/programador.

- Fase 3. El planificador/programador genera y refina los planes de recuperación de información, teniendo en cuenta el descubrimiento de Lotus, Computer Associates, Novell y Lighthouse Design como posibles productores de hojas de cálculo para Windows y OS/2.

- Fase X. El agente encargado de buscar en Lighthouse Design determina con un grado de probabilidad aceptable que los productos de Lighthouse no son para las plataformas OS/2 ni Windows. Aunque, como el usuario ha especificado que acepta participar en la búsqueda, el agente le pregunta si la información encontrada le interesa.

- Fase N. Se supera el tiempo límite y se envían al usuario varios artículos y páginas HTML sobre Lotus123, Novell Quattro Pro, Computer Associates CA-Compete! y Microsoft Excel.

Habiendo visto el desarrollo de una actuación del sistema, se pueden distinguir los componentes necesarios para su funcionamiento. En la Figura 4.4 podemos ver una representación de la arquitectura de los CIG Searchbots.

4.3.3 Agentes asistentes personales (que adaptan interfaces)

Los agentes asistentes personales ponen énfasis en la autonomía y el aprendizaje, con la intención de realizar tareas por sí mismos. Algunos autores han definido a los agentes de interface como asistentes personales que colaboran con el usuario en el mismo entorno de trabajo . Hay

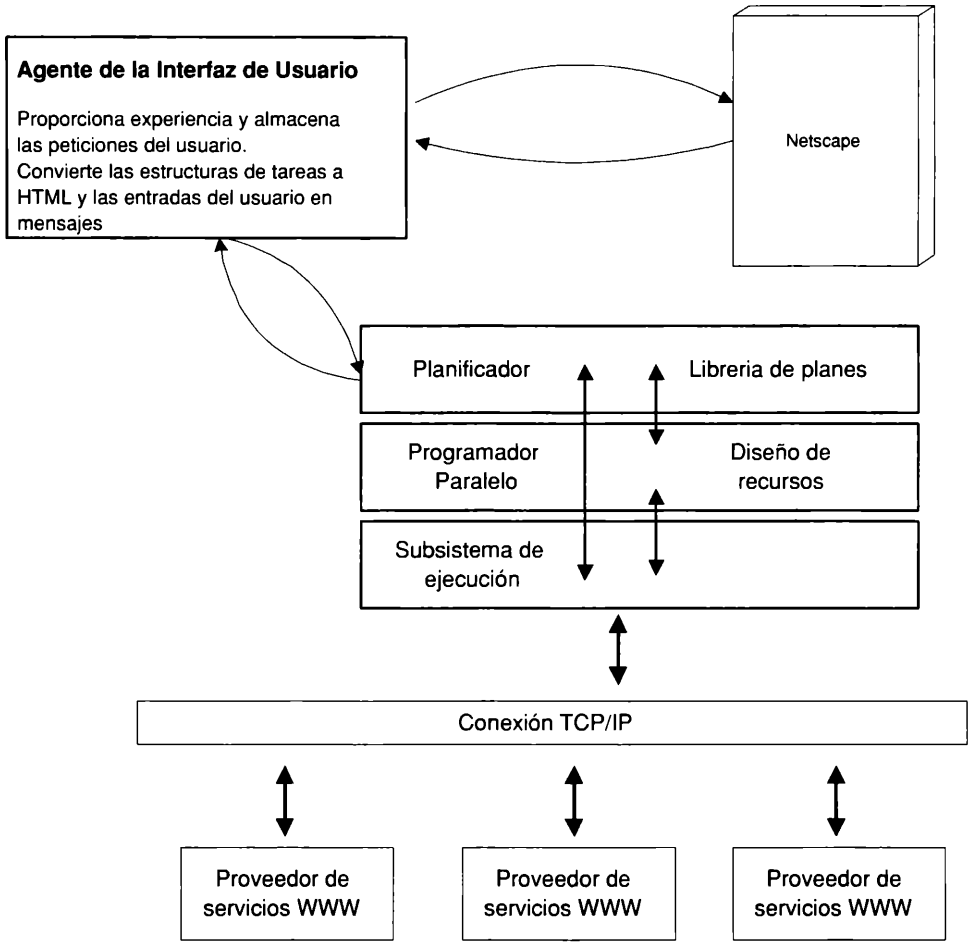

Figura 4.4: *Arquitectura de CIG Searchbots.*

que diferenciar el tipo de colaboración que realizan estos agentes: fundamentalmente colaboran con el usuario aunque también pueden colaborar con otros agentes. La colaboración con el usuario no requiere un lenguaje de comunicación explícito, como se necesita para la colaboración con otros agentes.

En la Figura 4.5 podemos ver como actúa un agente asistente personal genérico tratando de adaptar una interfaz de usuario de una aplicación.

Esencialmente, un agente de interfaz proporciona asistencia a un usuario inexperto que está aprendiendo a utilizar una aplicación particular, como una hoja de cálculo, un sistema operativo o un navegador de Internet. El agente observa y monitorea las acciones realizadas por el usuario en la interfaz, aprende nuevas acciones del usuario y hace sugerencias sobre cómo realizar acciones. El agente de interfaz actúa como un asistente personal que coopera con el usuario en la realización de alguna tarea con la aplicación.

Normalmente, estos agentes aprenden para ayudar mejor al usuario. El aprendizaje puede basarse en:

- Observación e imitación del usuario
- Recibir las experiencias positivas y negativas del usuario

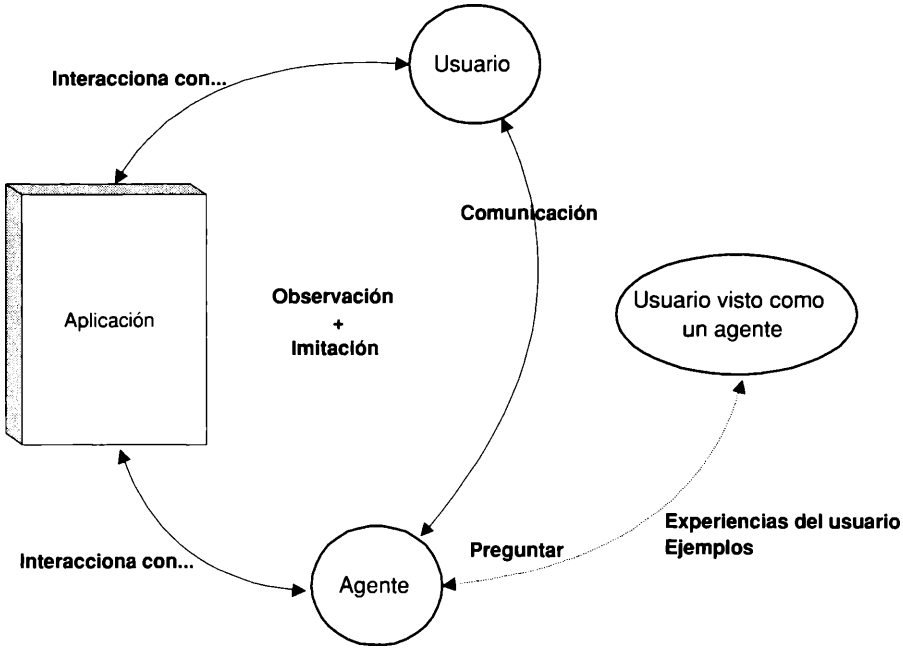

Figura 4.5: *Actuación de un agente adaptador de interfaces.*

- Recibir instrucciones explícitas del usuario
- Pedir consejo a otros agentes

La cooperación con otros agentes, si existe, se limita prácticamente a pedirles consejo, nunca existe una negociación como en el caso de los agentes que colaboran.

La ventaja de los agentes de interfaz es triple. En primer lugar, facilita el trabajo al usuario y al desarrollador de aplicaciones, haciendo que estos tengan que trabajar menos. En segundo lugar, el agente puede adaptarse, con el tiempo, a las preferencias y hábitos del usuario. Y finalmente, el agente puede adaptarse a un conjunto de usuarios, por ejemplo los trabajadores de una empresa.

A continuación vamos a ver un ejemplo de un sistema formado por un único agente asistente personal, que ayuda a su usuario a navegar y a buscar información en Internet.

Ejemplo: LETIZIA: un agente que asiste en la navegación por Internet

Letizia se basa en la idea de combinar la navegación por la World Wide Web con las búsquedas automatizadas. El modelo utilizado para Letizia se basa en que la búsqueda de información es una tarea cooperativa entre un usuario humano y un agente software inteligente. El usuario y Letizia navegan por el mismo espacio de búsqueda -documentos enlazados en la WWW-, buscando los más interesantes. No hay objetivos a *priori*. La diferencia entre la búsqueda que puede hacer un usuario y la que puede hacer Letizia, es que mientras la del usuario tiene una función de evaluación de confianza implícita, Letizia aporta la alternativa de realizar una búsqueda mucho más rápido que éste. Para eso utiliza el comportamiento pasado del usuario para anticipar una aproximación de sus intereses.

La parte más importante del diseño de Letizia es su estructura de control, que permite que el usuario pueda navegar manualmente por documentos y dirigir búsquedas sin que Letizia le

interrumpa. El cometido de Letizia durante la interacción del usuario es simplemente observar las acciones del usuario y realizar inferencias que puedan ser útiles en el futuro. Al mismo tiempo que el usuario está navegando, Letizia lanza una búsqueda que pueda anticipar las posibles necesidades futuras del usuario. En un momento dado, el usuario puede solicitar al agente un conjunto de recomendaciones basadas en el estado actual de su navegación y de la búsqueda de Letizia. Estas recomendaciones cambian dinámicamente, dependiendo de los cambios de interés del usuario o de los cambios del entorno.

Podemos clasificar a Letizia como un agente asistente personal que adapta interfaces de usuario. Inicialmente, la base de conocimiento del agente está vacía, y el conocimiento sobre el dominio se obtiene de modo incremental en el tiempo, como resultado de realizar inferencias sobre las acciones concretas del usuario. La estrategia para realizar inferencias se encuentra a medio camino entre:

- Recuperación de información

  El usuario que consulta una base de datos, que contiene información básicamente irrelevante, con la idea de encontrar una pequeña cantidad de información útil.

- Filtrado de información

  El usuario es víctima de una gran cantidad de información relevante, donde la tarea más difícil es eliminar la información menos relevante.

Modelar el proceso de navegación del usuario

El proceso de navegación del usuario consiste en examinar un documento HTML de la World Wide Web, decidir qué apuntadores seguir, retornar al documento anterior, ir a un documento explícitamente almacenado en una lista o bien añadir el documento actual en la lista. El objetivo de Letizia es construir un modelo del modo de navegación del usuario. Es decir, se trata de explorar de forma automática el documento HTML que el usuario esta visualizando o el que podría haber visualizado y evaluar los resultados desde la perspectiva del usuario. El resultado de la actuación de Letizia consiste en presentar unas recomendaciones en forma de apuntadores.

La ventaja aportada por Letizia es la capacidad de solapar la búsqueda y evaluación del resultado con la navegación del usuario. Mientras éste está leyendo un documento el agente trabaja. El usuario es quien mejor puede valorar la relevancia de un documento, y por tanto sería absurdo hacerle esperar para preparar una recomendación de algo que le interese, ya que si está leyendo un documento es porque le interesa. El mejor modo de usar las recomendaciones del agente es cuando el usuario no sabe qué hacer. Letizia sólo ha de proporcionar sugerencias, nunca ha de tomar el control de la interfaz de usuario.

Inferencias acerca del comportamiento del usuario

La observación del modo de navegación del usuario puede servir para que el agente conozca sus áreas de interés, pero se trata de una información pobre. Una información relevante sobre el comportamiento del usuario es cuando éste guarda un registro -*bookmark*- de un documento que le interesa. La decisión de seguir un apuntador puede indicar un cierto interés en el tópico del apuntador, aunque hay que tener en cuenta que el usuario muchas veces sigue un apuntador sin saber lo que hay detrás. Si el usuario vuelve atrás rápidamente, sin guardar referencia del documento o sin seguir más apuntadores, podemos considerar que el documento no interesa.

De todas maneras, seguir un apuntador es siempre un buen indicador sobre el interés del documento que contiene el apuntador. Volver muchas veces a un documento es señal clara acerca del interés como también lo es el tiempo -relativo a la longitud- de permanencia en un documento. Si se tiene en cuenta que la tendencia es explorar las páginas de arriba a abajo y de izquierda a derecha, si un apuntador es pasado por alto el agente puede estimar que no interesa. Un apuntador es pasado por alto si no es seleccionado mientras el usuario selecciona otros apuntadores situados más adelante en el documento. Una selección posterior del apuntador puede indicar lo contrario.

Letizia no soporta procesamiento de lenguaje natural, su modelo de un documento es una lista de palabras clave. Utiliza una arquitectura orientada a objetos para facilitar la incorporación de nuevas heurísticas sobre el comportamiento de navegación del usuario. No determina una medida de interés de un documento, simplemente ordena los distintos apuntadores según el interés del usuario. Puede dar explicaciones de sus inferencias y deducciones.

Una de las razones principales que impulsan a utilizar agentes como Letizia es la *persistencia del interés*. Cuando un usuario muestra interés por un tema seleccionando un apuntador o realizando una búsqueda con una palabra clave, sería extraño que el interés terminara al recibir los resultados de la búsqueda. Letizia se ocupa de recordar y buscar los items de interés expresados por acciones pasadas. La persistencia del interés también es importante para obtener las estrategias de búsqueda de información que prefieren los usuarios. Letizia tiene también heurísticas para saber cuando el interés decae. Las acciones más recientes hacen que las más antiguas pierdan importancia. Otra manera de olvidar es rebajar la importancia de los tópicos más lejanos, medidos en apuntadores accedidos desde el inicio de la navegación.

Estrategias de búsqueda

La estructura natural de las páginas de la World Wide Web inducen al usuario a realizar una navegación en profundidad. Esta manera de moverse por el hiperespacio provoca una pérdida de tiempo y que el usuario se acabe perdiendo. Los apuntadores que se recorren en profundidad no son siempre representativos. Letizia, para compensar esto de alguna manera, realiza una búsqueda a lo ancho. Esto resulta útil ya que el usuario puede haber pasado por alto apuntadores que son de su interés.

La búsqueda en profundidad de Letizia está limitada por los efectos de la interacción del usuario. Cada movimiento del usuario reenfoca la búsqueda, impidiendo que el agente se pierda en las profundidades del hiperespacio. Letizia se detiene cuando llega a una página que contiene un formulario de búsqueda. Si observa que el usuario esta muy *perdido* puede recomendarle ir a la página que contiene el formulario.

4.3.4 Agentes móviles

Vamos a ver ahora agentes software, capaces de deambular por las redes de ordenadores, interactuando con los hosts o recogiendo información para su propietario. Los agentes móviles realizan las tareas fijadas por su usuario recorriendo redes de ordenadores, y vuelven a casa para presentar los resultados. Estas tareas pueden ir desde la reserva de un billete de avión a la administración de una red de telecomunicaciones.

La movilidad no es una condición necesaria ni suficiente para los agentes. Los agentes móviles son agentes porque están situados en un entorno, porque son autónomos y pueden tener habilidades sociales que les permitan cooperar con otros agentes. Es importante aclarar esto, ya que la percepción usual de los agentes, en parte debido a los medios de comunicación, es la de entidades que se mueven.

La hipótesis que subyace en la idea de los agentes móviles es que los agentes no han de ser necesariamente estacionarios, esto es, podemos encontrar ventajas significativas, en ciertas áreas de aplicación, reemplazando los agentes estáticos por su contrapartida de tipo móvil.

Para poder entender mejor las ventajas de los agentes móviles vamos a suponer la situación en la que un programador ha de escribir un programa de reservas de vuelos . El programa ha de permitir hacer reservas desde un ordenador personal, accediendo a las bases de datos de reservas de las diversas compañías aéreas. Se han de poder especificar todas las preferencias del usuario: no fumador, salida entre las 7 y las 9:30 de Buenos Aires, llegada a Santiago antes del mediodía, hacer una sola escala como mucho, etc. Un Sistema Basado en Agentes de tipo estático necesitará lanzar consultas preguntando por todos los vuelos que salgan entre las dos horas especificadas, en todas las bases de datos, que pueden ser del orden de doscientas. También necesitará una lista de todas las conexiones de vuelos. Cada una de estas acciones implica trabajar con una gran cantidad de información heterogénea, que puede saturar la red.

La alternativa al sistema de agentes estáticos podría consistir en encapsular todo el programa, al estilo como lo hace la orientación a objetos, en un agente que sea capaz de recorrer la red de ordenadores de las compañías aéreas. Este agente será capaz de viajar de modo fiable hasta las bases de datos de reservas, una vez allí realizar consultas locales a las bases de datos y volver *a casa* con un plan que será evaluado para confirmarlo o anularlo. Esta alternativa permite desechar los altos costes que supone la transferencia de grandes volúmenes de información al ordenador que controla el sistema. Podemos concluir el ejemplo viendo las ventajas de la movilidad de los agentes:

- Reducidos costes de comunicación: puede haber mucha información en bruto que necesita ser examinada para determinar su relevancia. La transferencia de esta información para ser analizada de modo local podría sobrecargar la red.

- Limitación de los recursos locales: la capacidad de proceso y de almacenamiento en el ordenador local puede ser muy limitada. Podría darse el caso extremo de solo poder procesar y almacenar únicamente los resultados de una búsqueda.

- Fácil coordinación: lo más sencillo es coordinar cierta cantidad de peticiones remotas y procesar y enlazar los resultados de modo local.

- Arquitectura distribuida flexible: los agentes móviles proporcionan una arquitectura única y distribuida con nuevas funcionalidades respecto a las arquitecturas de tipo local. Aporta un nuevo punto de vista para construir aplicaciones distribuidas.

- Alternativa de diseño: los agentes móviles representan una oportunidad para replantear el proceso de diseño de aplicaciones, en general.

4.4 Comportamiento de los agentes en Internet

Como se ha visto, uno de los problemas a los que actualmente se enfrenta Internet es la sobrecarga del ancho de banda disponible. Los agentes que operan sobre la red son, en parte, responsables de esta saturación. Un agente actúa de modo continuo durante periodos de tiempo muy prolongados. Para aumentar la rapidez y el rendimiento, a menudo los sistemas multiagente lanzan peticiones en paralelo que provocan un uso muy alto del ancho de banda. Incluso partes remotas de la red pueden notar la saturación de recursos si un agente individual o un sistema multiagente está realizando un gran número de peticiones en un pequeño espacio de tiempo.

Además del uso masivo de la red, los agentes también hacen un uso por encima de lo normal de los propios servidores. Un servidor puede resultar considerablemente sobrecargado dependiendo de la frecuencia de las peticiones de documentos que hacen los agentes. Esta sobrecarga implica un menor servicio para los usuarios "normales" de estos servidores.

Se ha de estudiar el modo de evitar las consecuencias de este modo de actuar de los agentes, aunque no son sólo los agentes quienes sobrecargan la red. Por ejemplo, los web browsers contribuyen al problema, ya que incorporan mecanismos que permiten acceder a un documento y leer en paralelo todas las imágenes que contiene.

Los agentes y sus diseñadores pueden disminuir el efecto negativo de éstos en el ancho de banda. Algunas soluciones al respecto son:

- Ejecutar las tareas de modo local al servicio remoto. Se disminuye el tráfico en la red si un agente viaja hasta el lugar en el que debe realizar una tarea, por ejemplo una consulta en una base de datos, y envía el resultado *a casa*. Ejecutar tareas de modo local significa disponer de un conjunto de agentes móviles.

- Utilizar resultados y experiencia del pasado. Utilizando los resultados obtenidos en tareas pasadas podemos realizar las tareas futuras más eficientemente o incluso puede no ser necesario volver a realizar la tarea. De esta manera, los agentes buscadores pueden evitar hacer accesos remotos a buscadores y servidores cada vez que el usuario solicita una búsqueda, aunque también hay que tener en cuenta que los resultados pasados pueden caducar.

- Ejecutar tareas fuera de las horas de máximo trafico en la red. Los agentes podrían, por ejemplo posponer para la noche la actualización de sus bases de datos.

- Que los agentes sean educados y pidan permiso cada vez que quieran acceder a un servidor. Algunos servidores proporcionan información altamente volátil. Por ejemplo, periódicos electrónicos que actualizan sus páginas cada 20 minutos. Si un agente quiere acceder al servidor del periódico para indexarlo y estudiar su topología puede suceder que cuando termine su tarea el servidor haya cambiado totalmente su configuración inicial. Además, es de suponer que un periódico en línea es un servidor bastante solicitado por lo que la presencia de agentes puede representar un perjuicio para los visitantes humanos. Sería bueno que el administrador del periódico pudiera especificar un método para evitar que los agentes accedan al servidor.

Se ha definido un método para que los servidores de la World Wide Web puedan especificar si los agentes pueden acceder o no al servidor, y en caso afirmativo, qué tipo de consultas pueden hacer en el servidor. Este método se llama *Standard for Robot Exclusion*, y propone la creación de un archivo robots.txt en cada servidor, que especifica la política de accesos permitida (ver Sección 2.5).

Por último, se presenta en este capítulo un ejemplo de un sistema multiagente. Se ha elegido este ejemplo en primer lugar por la simplicidad de su arquitectura, y en segundo lugar porque presenta un diseño donde están incluidos muchos de los agentes vistos en este capítulo.

Ejemplo: MACRON

El sistema MACRON es una propuesta de sistema multiagente para la recuperación de información de modo cooperativo.

La consulta del usuario dirige la creación de planes parciales de IG (Information Gather). Esto deriva en la utilización de varios agentes que cooperan para la resolución de los planes,

utilizando la información disponible en Internet. Este proceso se basa en técnicas de resolución de problemas de modo distribuido. Algunas de las características de los agentes de MACRON son:

- Autonomía
- Reaccionan cambiando el entorno
- Pueden compartir información
- Están situados en una sociedad, la cooperación entre ellos implica gestionar las interdependencias entre sus actividades
- Un agente es libre para organizarse como quiera teniendo en cuenta su tiempo de respuesta.

Cuando se trabaja con múltiples agentes aparece la concurrencia, que es una gran ventaja cuando el espacio de búsqueda es muy grande, como Internet. Los planes de cada agente se pueden descomponer en subplanes con pocas dependencias, aunque se necesitará coordinación. Si un sistema maneja una gran cantidad de datos, manejarlos en los lugares donde se almacenan los datos es más viable.

Organización de MACRON

MACRON organiza el sistema con tres tipos de agentes:

- Agentes de razonamiento DECAF
- Agentes de red de recuperación de bajo nivel de red
- Agentes de interfaz de usuario

Todo sistema multiagente tiene una arquitectura que organiza y lleva adelante a los agentes involucrados en el sistema. Esta organización responde a las cuestiones:

- ¿Para qué tipos de tareas un agente debe estar preparado?
- ¿Qué recursos debe utilizar?
- ¿Cuáles son sus responsabilidades?

Desde un punto de vista organizacional, los agentes de MACRON forman una matriz que consta de unidades funcionales y de respuesta a consultas. Es la misma idea que se utiliza en las empresas al agrupar personal en una matriz para realizar un proyecto. Esta organización permite un alto grado de flexibilidad en entornos dinámicos e inciertos, mientras que sigue permitiendo una asignación y rastreo inteligente de los recursos.

Una unidad funcional da acceso a algún tipo particular de fuente de información. Cada unidad funcional tiene un agente organizador (FM: Functional Manager), con la función de asignar tareas a los agentes de la unidad. La unidad funcional puede contener agentes que tengan conocimiento y capacidades similares o bien agentes con habilidades semejantes. El FM tiene una visión abstracta de todo el estado de su unidad funcional desde la que puede asignar tareas.

Una unidad de respuesta de consultas comprende un agente organizador de consultas y un conjunto de agentes de distintas unidades funcionales para responder a una petición del

usuario. El organizador de consultas es el responsable de generar un primer plan de IG. Recluta los agentes necesarios, de las distintas unidades funcionales. Y además monitorea la ejecución del plan. El usuario interactúa con el organizador de consultas vía un web browser y formularios HTML.

Los agentes se comunican entre sí mediante KQML. La estructura de sus tareas se basa en TAEMS (Task Analysis, Environment Modeling, and Simulation). Utilizan una memoria OCM (Organization Chart Manager) que permite a un agente obtener información acerca de los agentes con quien se puede comunicar. Se pueden incorporar nuevos agentes y nuevas unidades funcionales añadiéndolos al OCM.

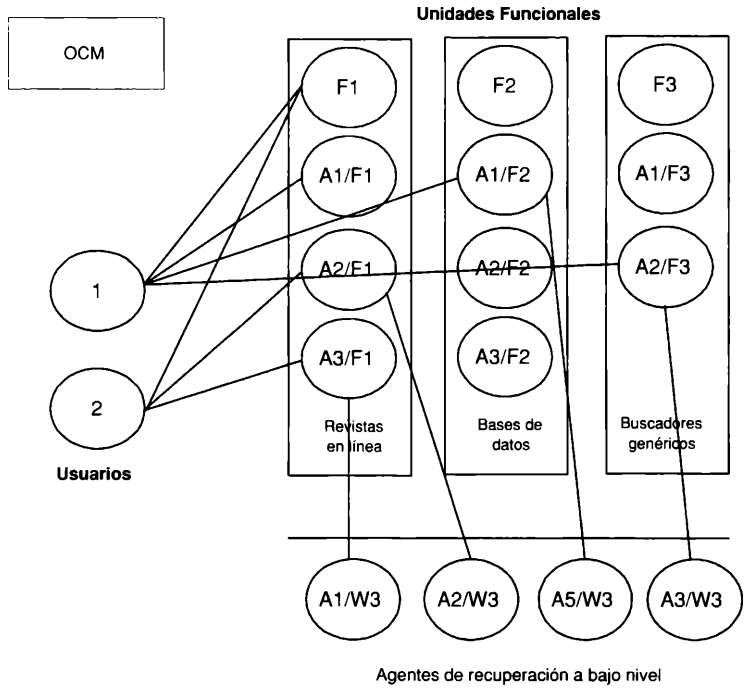

Figura 4.6: *Modelo de MACRON.*

Con una organización puramente funcional se pueden establecer equipos de agentes, donde cada equipo tiene al menos un agente de cada unidad funcional. Al formar estos equipos podemos tener algunos inconvenientes:

- Algunos agentes de los equipos podrían estar desocupados desde el momento en que no todas las consultas necesitan todos los recursos funcionales.
- Si un agente de un equipo no está disponible en un momento dado, la consulta puede fracasar
- Que los agentes aprendan implica que cada equipo se especialice en un área, pero sería bueno utilizar agentes expertos de distintos equipos, para resolver las nuevas consultas.

Si nos situamos en el otro tipo de organización, la no funcional, con un gran conjunto de agentes sin diferenciar, el gestor de consultas podría formar nuevos equipos prescindiendo

de los agentes no disponibles. Esto puede fracasar, ya que es difícil reunir a los agentes más apropiados para resolver una consulta.

MACRON utiliza lo mejor de estos dos tipos de organización, la flexibilidad de formar un equipo de agentes sobre la marcha y el hacer un seguimiento de los agentes.

Arquitectura de DECAF

DECAF trabaja del siguiente modo, cuando se recibe una petición, el planificador construye un plan compuesto de acciones primitivas. Se prepara una programación de los agentes, mediante el programador local y el módulo de coordinación. El monitor de ejecución es el encargado de realizar el seguimiento del plan y, si es necesario, puede encargarse de aplicar las medidas necesarias para reformular el plan o reprogramar la coordinación de los agentes.

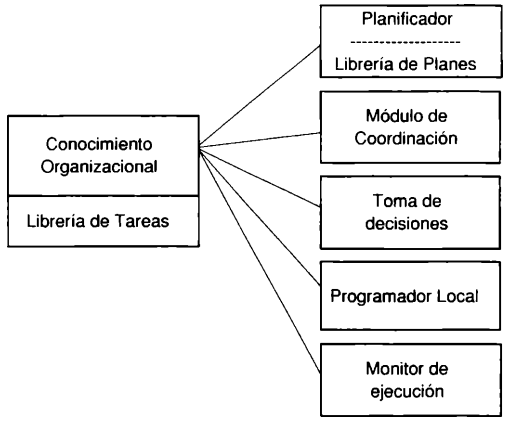

Figura 4.7: *Arquitectura de un agente individual DECAF.*

En la Figura 4.7 vemos la arquitectura de un agente individual con todos sus componentes:

- Librería de tareas: contiene las tareas del agente en un momento preciso y su relación con las tareas de otros agentes. Este componente es una estructura compartida por todos los otros componentes activos del agente. Aquí es donde el planificador crea nuevas *estructuras de tareas.*

- Planificador: crea una representación de las tareas. El planificador instancia una estructura de tarea, basada en una consulta o en una subtarea. Rellena los atributos de una tarea dependiendo de la consulta. La estructura de tarea instanciada con valores se almacena en una base de datos que es compartida por los agentes. Las unidades funcionales pueden tener librerías de tareas especializadas en sus funciones propias.

- Módulo de coordinación: su tarea consiste en reconocer patrones en la actual estructura de tarea y formular sugerencias acerca de las tareas internas de coordinación.

- Programador local: construye programaciones posibles, dependiendo de la estructura de tarea actual y de los compromisos entre agentes, tanto entre los agentes del sistema como con agentes externos.

- Toma de decisiones: elige el programa del programador local que sea más adecuado al actual estado de los agentes del sistema.

- Monitor de ejecución: supervisa las ejecuciones de acciones del programa en curso. El monitor de ejecución es útil para anticipar los posibles problemas y excepciones del sistema.

Agentes de recuperación de bajo nivel

Estos agentes pertenecen a la clase de los agentes gestores de información que hemos visto en este capítulo. El sistema MACRON dispone de un conjunto de estos agentes, que son responsables de buscar y procesar URLs para que puedan ser usados por la planificación de alto nivel. También se ocupan de la interacción con bases de datos remotas y buscadores.

Parte II

BIPI: metabuscador

Capítulo 5

Arquitectura

5.1 Arquitectura

En esta sección se presenta una definición general de la arquitectura. Se ha optado por un modelo multi-tier y distribuido. Se ha buscado mediante este modelo lograr independencia entre capas, escalabilidad y performance, basados principalmente en la naturaleza de la aplicación.

En forma general, se distribuyeron los distintos componentes del modelo en cuatro capas bien definidas. La primera es la capa del Cliente, constituida por el Web Browser y utilizada por el/los usuario/s de la aplicación. La segunda, el Servidor de Web, contiene aquellos componentes que responden a las solicitudes de los clientes interactuando con la tercera capa. Esta tercera capa está conformada por la aplicación propiamente dicha, distribuyéndose en el Servidor de Aplicación (Aplicación Local) y en Servidores Remotos (Aplicación Remota); quedando definida de esta forma la distribución física de los componentes. La capa inferior o cuarta capa es en la que interviene el Sistema de Gestión de Base de Datos, elemento externo a la aplicación que puede ser intercambiable entre los distintos productos similares que existen actualmente en el mercado.

A continuación se enumeran cada una de las capas junto con un breve detalle de su funcionalidad y su constitución, cuya información será ampliada en secciones subsiguientes.

- **Cliente:** está conformada por el Web Browser que provee la visualización de la interfaz con el usuario, interactuando con el Servidor de Web para obtener las páginas que proveerán dicha interfaz.
- **Servidor de Web:** es la capa encargada de la interacción con los distintos perfiles de clientes. Internamente tiene dos componentes:
  - **Componente Común de Web:** es el componente al cual acceden los clientes Usuarios Comunes para resolver búsquedas, mediante el *Servicio de Búsqueda*, y para incorporar nuevas URLs, que posteriormente serán indexadas por BIPI, a través del *Servicio de Incorporación de URLs*.
  - **Componente Administrador de Web:** permite que el Usuario Administrador pueda establecer los parámetros del buscador, a través de la aplicación de Administración (Web Admin).
- **Aplicación:** conforma la capa lógica del sistema, ubicándose en ella todos los agentes encargados de la búsqueda, indexación y administración. Está compuesta por dos elementos diferenciados por el entorno en el que se ejecutan: *entorno local* y *entorno remoto*.

- **Aplicación Local:** este elemento de la Capa de Aplicación se ejecuta en el servidor de la aplicación. Sus seis componentes se dividen en cuatro grupos:
  1. *Sub-entorno de Búsqueda*: genera resultados de búsqueda a partir del envío de una solicitud proveniente de la Interfaz de UsuarioÈste sub-entorno abarca tres componentes: Agente Coordinador, Agente Compactador y Agente de Búsqueda Continua.
  2. *Agente Indexador*: encargado de indexar las páginas que le son indicadas por el Agente de Búsqueda Continua y el Agente Autónomo de Mantenimiento.
  3. *Agente Autónomo de Mantenimiento*: procura revisar la caducidad de las páginas indexadas.
  4. *Agente de Incorporación de URL*: recepciona las solicitudes de incorporación de URLs provenientes de la capa superior, almacenándolas como indexaciones pendientes en la capa inferior.
- **Aplicación Remota:** este elemento de la Capa de Aplicación está conformado por distintos agentes que realizan la búsqueda en buscadores ubicados en sitios remotos (excepto el motor de BIPI), filtran la información recolectada y envían los resultados a la Aplicación Local. Está conformado por n Grupos de Búsqueda, que son tuplas de los siguientes componentes específicos:
  1. *Agente Específico de Búsqueda*: se comunica con un buscador determinado para resolver una búsqueda.
  2. *Agente Filtrador*: filtra la información generada por el Agente Específico de Búsqueda y envía los resultados al Agente Compactador (que forma parte del Sub-entorno de Búsqueda que procesa en la Aplicación Local).

- **Sistema de Gestión de Base de Datos (SGBD):** es la capa encargada de proveer los servicios necesarios para el acceso y mantenimiento de la información generada por la aplicación, respondiendo a las peticiones de las capas superiores.

En la Figura 5.1 se puede visualizar un detalle de los distintos componentes que conforman la arquitectura del buscador y de los canales de comunicación entre los mismos, observándose la división en capas explicada anteriormente.

5.2 Cliente

El cliente se encuentra compuesto por el Web Browser que se ejecuta en la máquina del usuario. Al referenciar alguna de las páginas Web de la aplicación, que se ubican en el Servidor de Web, se ejecuta un pedido a dicho servidor. Este sirve la solicitud enviando la página correspondiente al cliente para que sea presentada al usuario.

Aquí se presentarán al usuario las páginas para poder interactuar con la aplicación, tales como las páginas de búsqueda y de administración.

5.2.1 Perfiles de Clientes

El sistema permite la interacción con dos perfiles de clientes diferenciados por el tipo de requerimientos que realizan al Servidor de Web y el conjunto de permisos que tienen sobre la aplicación.

- **Usuario Común**: interactúa con el Servidor de Web realizando principalmente requerimientos de búsqueda y, en menor grado, requerimientos de incorporación de páginas.

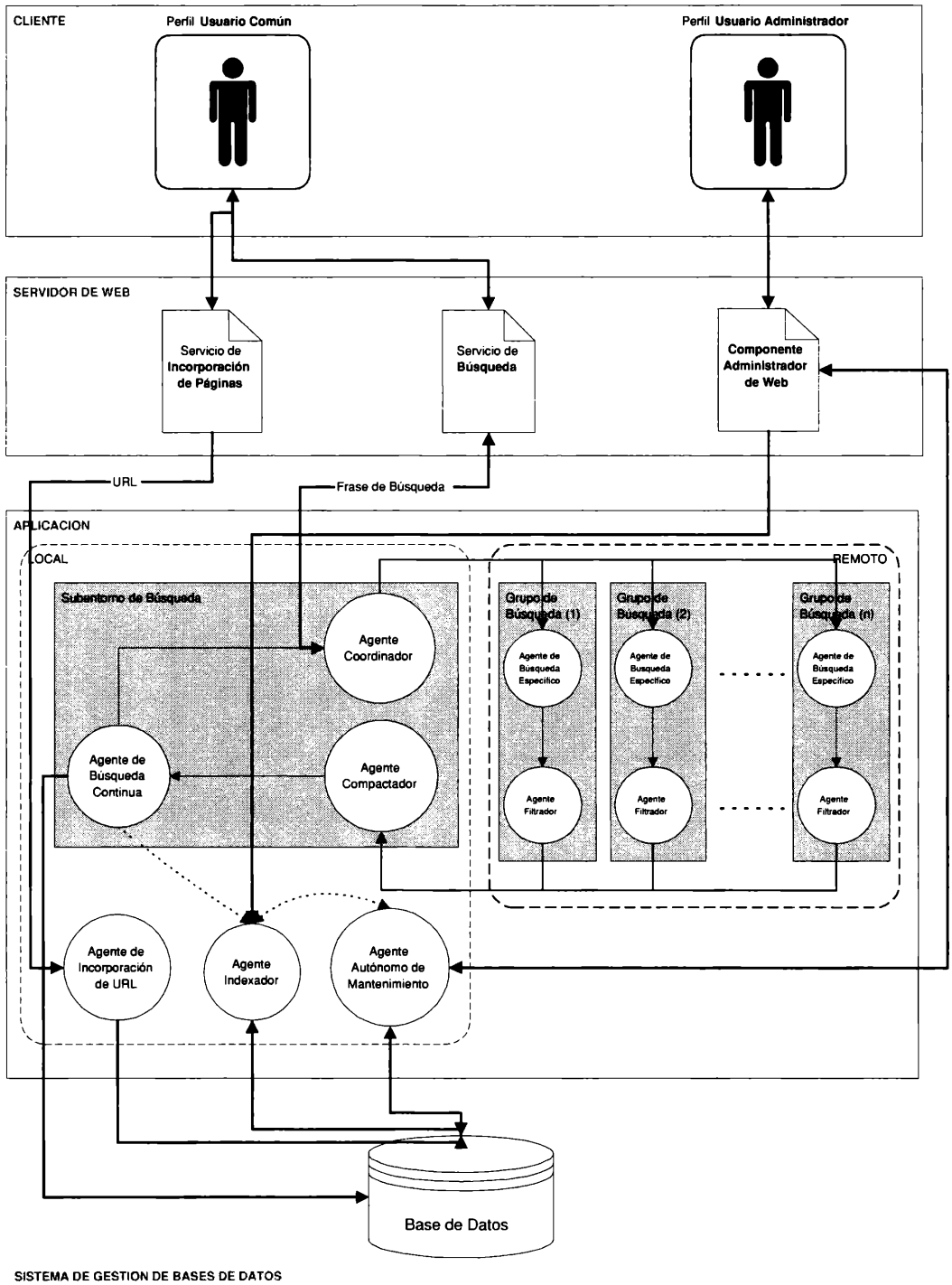

Figura 5.1: *Arquitectura de BIPI.*

- **Usuario Administrador**: interactúa con el Servidor de Web realizando requerimientos de configuración de la aplicación, estableciendo parámetros que influirán en el comportamiento de la aplicación para el perfil Usuario Común. Un usuario con perfil Usuario Administrador puede ejecutar requerimientos que son inherentes al perfil Usuario Común, como pueden ser los requerimientos de búsqueda.

5.3 Servidor de Web

El Servidor de Web conforma en sí mismo una capa dentro del modelo multi-tier del sistema, siendo una pieza clave en él. En este servidor se almacenan las páginas que permiten al usuario interactuar con la aplicación, ya que es quien atiende las solicitudes que realiza la capa Cliente. Todos aquellas pedidos que no hacen a la Interfaz de Usuario son derivados a la capa inferior siguiente (Aplicación). Esta capa posee dos componentes internos diferentes basados en los perfiles de clientes que realizan las peticiones: el *Componente Común de Web* y el *Componente Administrador de Web.*

5.3.1 Componente Común de Web

Facilita un conjunto de páginas que proveen el acceso a la aplicación permitiendo al perfil Usuario Común realizar las operaciones que se le posibilitan. Las funciones que puede realizar se dividen en dos servicios detallados a continuación:

- **Servicio de Búsqueda**: este servicio es el que se prioriza facilitar en la aplicación, permitiendo a los Usuarios Comunes realizar consultas buscando páginas que cumplan ciertos criterios. El sistema, en base a la información de sí mismo y aquella obtenida de otros servicios de búsqueda, provee los resultados más adecuados y los envía a los clientes en una página de resultados (para su posterior presentación a los usuarios); los resultados provenientes de otros buscadores servirán de "aprendizaje" del sistema siendo almacenados en la base de información interna, para su posterior utilización en búsquedas subsiguientes.

  La realización de una búsqueda está dada por la frase a buscar (en FBE[1]) y los parámetros de búsqueda necesarios (p.ej: se permite especificar los buscadores que se utilizarán en la realización de la búsqueda en la capa de aplicación). El establecimiento de estos datos y su posterior envío a la capa inferior genera una consulta que será atendida por la aplicación. Los resultados que la capa de aplicación retorne se enviarán al cliente como se explicó en el párrafo precedente.

- **Servicio de Incorporación de URLs**: mediante este componente interno del Servidor de Web, el Usuario Común (o incluso el Usuario Administrador en su rol de Usuario Común) puede "brindar conocimiento" al sistema. El usuario da de alta solicitudes de indexación de páginas (a través de la especificación de una URL, título de la página, palabras clave de la misma, etc.) que son incorporadas como peticiones pendientes (conocimiento potencial) que posteriormente el Agente Indexador procesará, indexando y desglosando la página (la definición de la misma pasa a formar parte de la información contenida en la Base de Datos del sistema).

[1] FBE (Formato de Búsqueda Estandarizado): formato utilizado para el pasaje de solicitudes de búsqueda entre diversos agentes del sistema.

5.3.2 Componente Administrador de Web

Este componente provee una interfaz accesible a través de un Web Browser, mediante la cual se pueden establecer los parámetros generales del sistema (p.ej: aquellos que influirán en las búsquedas realizadas por los usuarios), consultas directas a la Base de Datos, etc.

5.4 Aplicación

Esta es la capa que engloba la mayor funcionalidad del sistema conteniendo en sí la lógica necesaria para la interacción usuario/sistema y sistema/usuario.

Los componentes de esta capa se basan en el concepto de agente, por ello en esta sección serán llamados de tal forma. Los agentes se relacionan con la capa superior, con la capa inferior o con otros agentes locales y/o remotos de la capa de aplicación; y se diferencian en clases según el entorno en el que procesan.

5.4.1 Entornos

Los entornos en los que procesan los agentes son el ***entorno local*** y el ***entorno remoto***, siendo estos utilizados para la división de la funcionalidad en la capa de aplicación.

Esta división es necesaria dado que la aplicación realiza su "aprendizaje" comunicándose con otros buscadores que funcionan en entornos remotos. Cuando a la aplicación le llega un requerimiento de búsqueda, ésta obtiene información local almacenada en la capa inferior (ver Sección 5.5) y también obtiene información de buscadores externos. Para estos últimos existen agentes que trabajan remotamente enviando la información al agente receptor que procesa en la aplicación local.

5.4.2 Aplicación Local

La aplicación local contiene agentes para realizar tareas de búsqueda, indexación, mantenimiento e incorporación de páginas. La tarea de búsqueda es realizada en un sub-entorno llamado *Sub-entorno de Búsqueda* que se compone de tres agentes: Agente Coordinador, Agente de Búsqueda Continua y Agente Compactador, y las tareas de indexación, mantenimiento e incorporación de páginas son resueltas por el Agente Indexador, el Agente Autónomo de Mantenimiento y el Agente de Incorporación de URL respectivamente.

Sub-entorno de Búsqueda

El Sub-entorno de Búsqueda no es estático ni su existencia es eterna, sino que se genera dinámicamente para resolver un requerimiento de búsqueda determinado proveniente de la capa superior, por ello en un momento dado pueden coexistir ***n*** sub-entornos en el entorno local.

Ante la necesidad de realizar una búsqueda se crea un Sub-entorno de Búsqueda para su resolución que provoca la creación de una instancia de cada uno de los tres agentes que componen el sub-entorno. Estos agentes cooperan junto con los agentes remotos y finalmente producen resultados que el sub-entorno entrega a la capa superior.

El sub-entorno posee diversos canales de comunicación con su medio ambiente que pueden clasificarse, en primera instancia, de acuerdo al momento del proceso de la resolución de la búsqueda en el que se utilizan. Para una mayor comprensión de su utilización a su vez se pueden subclasificar en base al sentido del flujo de los datos que transportan; por un lado se encuentran los canales de entrada, que obtienen datos de su medio ambiente, y por otro lado los canales de salida, que envían datos hacia el medio ambiente.

A continuación se describe la clasificación de canales de entrada y salida de cada etapa del proceso de resolución de una búsqueda existentes en las inter-comunicaciones del Sub-*entorno* de Búsqueda con la Aplicación Remota y las capas superior (Servidor de Web) e inferior (Sistema de Gestión de Base de Datos).

- **Requerimiento de búsqueda:** esta etapa se inicia cuando el Servidor de Web transmite una solicitud proveniente del Cliente hacia la Aplicación Local (componente interno de la capa de Aplicación). La solicitud se genera en un navegador de Web comandado por un cliente con perfil Usuario Común y se envía hacia el Componente Común de Web (en el Servidor de Web) que es el origen de la solicitud de ingreso al Sub-entorno de Búsqueda. El receptor dentro del sub-entorno es el Agente Coordinador.

  Una vez resuelta la petición existirá una comunicación inversa (a través de la cual se retornarán los resultados) entre el Agente Coordinador y el Componente Común de Web.

  - Entrada: solicitud de búsqueda (sentencia en Formato de Búsqueda Estandarizado y parámetros adicionales) proveniente del Componente Común de Web.
  - Salida: resultados de la búsqueda (conjunto de elementos conformados por la URL, el título, la descripción y la relevancia de las páginas) que cumplen las condiciones establecidas a través del canal de entrada.

- **Requerimiento de búsqueda remota:** esta etapa ocurre cuando es necesario satisfacer la resolución de la búsqueda que le fuera requerida al Agente Coordinador. Este agente dispersa la solicitud hacia los Agentes Específicos de Búsqueda remotos y/o locales.

  - Entrada: ninguna
  - Salida: solicitudes hacia los Agentes Específicos de Búsqueda (sólo se envía una sentencia en Formato de Búsqueda Estandarizado a cada uno de ellos) provenientes del Agente Coordinador.

- **Recolección de resultados de búsqueda remota:** cuando los buscadores en la Aplicación Remota resuelven la petición de búsqueda retornan los resultados hacia el Agente Compactador. Este último unifica los resultados provenientes de los distintos Agentes Filtradores que funcionan conjuntamente con los Agentes Específicos de Búsqueda.

  - Entrada: resultados en FRBE[2] provenientes de cada uno de los Agentes Filtradores.
  - Salida: ninguna

- **Actualización mediante resultados:** cuando el Agente de Búsqueda Continua recibe los resultados compactados desde el Agente Compactador, a través de una comunicación interna al Sub-entorno de Búsqueda, evalúa la existencia de las páginas que componen el conjunto de resultados consultando al Sistema de Gestión de Base de Datos (SGBD). En base a estas evaluaciones determina cuales páginas se almacenarán en la Base de Datos del sistema y envía las mismas al SGBD para aportar nuevo conocimiento al sistema.

  - Entrada:

[2]FRBE (Formato de Resultado de Búsqueda Estandar): formato definido especialmente para el envío de resultados de búsqueda.

i. Resultado del chequeo de la existencia de una página específica (resultado positivo o negativo de respuesta del SGBD hacia el Agente de Búsqueda Continua).

- Salida:
  i. Solicitud de chequeo de la existencia de una página específica (el Agente de Búsqueda Continua envía al SGBD la URL que identifica la página para la cual se desea verificar su existencia en la Base de Datos).
  ii. Incremento de conocimiento: el Agente de Búsqueda Continua envía al SGBD una solicitud de alta de una URL específica para la cual dio negativo el chequeo de existencia.

Agente Coordinador

Tiene como objetivo interactuar y proveer un enlace entre la Aplicación Local y la capa superior (Servidor de Web), así como la coordinación de los agentes intervinientes en el procesamiento de una búsqueda.

Este agente es uno de los componentes del Sub-entorno de Búsqueda y se genera automáticamente ante la existencia de una solicitud de búsqueda, junto con los otros agentes que coexisten en el sub-entorno.

Sus funciones son recepcionar las búsquedas remitidas por la capa superior y coordinar su envío a los distintos Agentes Específicos de Búsqueda en el entorno remoto. El envío es realizado conservando el Formato de Búsqueda Estandarizado de manera que todos los agentes buscadores reciban la misma información para su posterior procesamiento en el buscador correspondiente. Por otro lado es el responsable de recibir del Agente de Búsqueda Continua los resultados de las búsquedas realizadas y formatearlos para su posterior envío a la capa superior.

Agente Compactador

El simple y único objetivo de este agente es generar un conjunto mínimo de resultados que finalmente se proveerá al usuario.

Es el encargado de producir un conjunto homologado de páginas a partir de los distintos resultados en Formato de Resultado de Búsqueda Estandar que le son entregados por los Agentes Filtradores a través de los diferentes canales de entrada que lo comunican con cada uno de ellos. A medida que obtiene los resultados de los Agentes Filtradores elimina los repetidos, obteniendo así un conjunto de enlaces únicos a páginas, ordenados por relevancia y que continúan utilizando el Formato de Resultado de Búsqueda Estandar. Posteriormente, al finalizar el ingreso de resultados, envía al Agente de Búsqueda Continua el conjunto mínimo de páginas resultante.

Este agente realiza su procesamiento en el Sub-entorno de Búsqueda de la Aplicación Local.

Agente de Búsqueda Continua

Es la pieza clave en el aprendizaje de la aplicación. Siendo uno de los componentes dinámicos del Sub-entorno de Búsqueda, este agente desarrolla como función principal el examen del conjunto mínimo de resultados entregado por el Agente Compactador y determina cuales de ellos proveen nueva información para la aplicación. También se encarga de redireccionar los resultados hacia el Agente Coordinador.

El Agente de Búsqueda Continua tiene tres canales de comunicación: uno de entrada con el Agente Compactador, uno de salida con el Agente Coordinador y uno bidireccional con el Sistema de Gestión de Base de Datos. A través del canal de entrada obtiene el conjunto mínimo de resultados a examinar y redireccionar, por el canal de salida envía los mismos resultados que obtiene del canal de entrada y por el canal bidireccional chequea la existencia de las páginas del conjunto de resultados y envía solicitudes de alta de URL específicas al SGBD (las cuales serán indexadas posteriormente por el Agente Indexador).

Agente Indexador

Su objetivo es procesar las solicitudes de alta de URL almacenadas en la Base de Datos, accediendo a las páginas (que las URLs indican) e indexando las mismas. Esto último consiste en el desglose de la información contenida en ella para su incorporación como conocimiento de BIPI.

Las solicitudes recuperadas de la Base de Datos pueden provenir de tres fuentes diferentes:

- El Agente de Incorporación de URL inserta nuevas solicitudes en la Base de Datos ante un requerimiento proveniente de la capa superior, más precisamente del Servicio de Incorporación de URLs.
- El Agente de Búsqueda Continua adiciona solicitudes cuando descubre que en el conjunto de resultados que analiza existen páginas aún no indexadas.
- El Agente Autónomo de Mantenimiento incorpora solicitudes de URLs cuando determina que ciertas páginas ya indexadas por el sistema cumplen con el CEP[3], por lo tanto están caducas, y necesita ser actualizada la información que se mantiene almacenada para las mismas.

Cuando el Agente Indexador recupera una URL del conjunto de solicitudes almacenadas en la Base de Datos, visita la página correspondiente a tal URL, investiga su contenido y la desglosa en las distintas partes que la componen (título, palabras clave, cuerpo, descriptores, etc.). Este procesamiento se denomina *Indexación de páginas*.

Su única vía de comunicación con otro componente del sistema se produce por el canal bidireccional que establece con el Sistema de Gestión de Base de Datos. A través de este canal recupera las URLs a indexar y envía la información desglosada de la página investigada. Adicionalmente existe la comunicación que establece al visitar la página correspondiente a la URL para recuperar su contenido.

La información que aporta el Agente Indexador, al almacenar los datos obtenidos por el desglose de las páginas visitadas (o revisitadas), es lo que provee parte del aprendizaje del sistema, ya que las nuevas indexaciones serán tenidas en cuenta en futuras búsquedas por el Agente Específico de Búsqueda que interactúa con el motor de búsqueda de BIPI.

Agente Autónomo de Mantenimiento

Interactúa con el Sistema de Gestión de Base de Datos obteniendo la fecha de incorporación y/o la fecha de actualización de cada una de las páginas ya indexadas por la aplicación. Sobre tales fechas aplica el Criterio de Expiración de Páginas para determinar si la información almacenada de una página debe ser actualizada. Para indicar que dicha información debe ser

[3]CEP (Criterio de Expiración de Páginas): criterio utilizado para identificar cuándo una página debe ser reindexada.

"desempolvada" se genera una solicitud interna de incorporación de URL que se almacena en la Base de Datos y que posteriormente será resuelta por el Agente Indexador.

Con el objetivo de realizar su función este agente posee un canal bidireccional de comunicación con el Sistema de Gestión de Base de Datos. Primero es utilizado como canal de entrada para obtener las fechas requeridas de cada página que se analizarán aplicando el Criterio de Expiración de Páginas. Una vez que son analizadas tales fechas el canal se reutiliza como canal de salida para insertar en la Base de Datos la solicitud de incorporación de página, si es que el Criterio de Expiración de Páginas indica que la página ha caducado.

5.4.3 Aplicación Remota

La aplicación remota contiene agentes para resolver búsquedas con determinados tipos de buscadores y para enviar al agente de la Aplicación Local (el Agente Compactador del Sub-entorno de Búsqueda) los resultados obtenidos. La tarea de búsqueda es realizada por distintos Grupos de Búsqueda que se componen de dos agentes: el Agente Específico de Búsqueda y el Agente Filtrador.

Grupo de Búsqueda

El Grupo de Búsqueda podría ser visto como el equivalente al Sub-entorno de Búsqueda de la Aplicación Local. Al igual que éste no es estático ni su existencia es eterna, sino que se genera dinámicamente para resolver un requerimiento de búsqueda determinado, proveniente del Agente Coordinador.

Por cada búsqueda que debe resolver un Sub-entorno de Búsqueda se generan n Grupos de Búsqueda en el entorno remoto. La cantidad de Grupos de Búsqueda que coexisten para resolver una búsqueda determinada está dada por los parámetros de búsqueda establecidos por un perfil de Usuario Común.

Además de coexistir n Grupos de Búsqueda en la resolución de una búsqueda pueden existir n conjuntos de Grupos de Búsqueda diferentes, cuya cantidad es directamente proporcional a la cantidad de Sub-entornos de Búsqueda.

Cuando a un Sub-entorno de Búsqueda se le indica por los parámetros adicionales de búsqueda que debe utilizar determinado tipo de buscador, se crea un Grupo de Búsqueda que interactúa con ese buscador en particular. Esto provoca la creación de una instancia de cada uno de los agentes que componen el grupo. Estos agentes cooperan entre ellos para la resolución de la búsqueda con un buscador específico.

El grupo posee dos canales de comunicación con su medio ambiente que pueden clasificarse directamente en base al sentido del flujo de los datos que transportan; por un lado se encuentran los canales de entrada, que obtienen datos de su medio ambiente, y por otro lado los canales de salida, que envían datos hacia el medio ambiente. Además posee un canal interno que permite la comunicación entre los dos agentes que conforman el grupo.

A continuación se describen los dos canales que proveen las inter-comunicaciones del Grupo de Búsqueda con la Aplicación Local y el canal que brinda la intra-comunicación.

- Canal de entrada: permite que se le informe al grupo la sentencia de búsqueda a utilizar en el buscador con el cual interactúa.
- Canal interno: mediante este canal el Agente Específico de Búsqueda envía al Agente Filtrador las páginas de resultados que recupera del buscador específico.
- Canal de salida: es utilizado para que el Agente Filtrador retorne al Sub-entorno de Búsqueda, que se corresponde con el grupo (más específicamente al Agente Compactador), los resultados filtrados.

Agente Específico de Búsqueda

El objetivo de este agente es encapsular las características de un buscador específico (Máquina de Búsqueda, Directorio o híbrido) interactuando con él para obtener resultados ante la necesidad de resolver una búsqueda.

En el momento de realizar un búsqueda existen *n* de estos agentes en el entorno remoto interactuando cada uno de ellos con un tipo de buscador.

El ingreso de información se realiza a través de un canal de entrada que lo relaciona con el Agente Coordinador del Sub-entorno de Búsqueda y a través del cual obtiene la sentencia de búsqueda. Una vez que obtiene la/s página/s de resultados del buscador con el que interactúa, las envía por un canal de salida (interno al Grupo de Búsqueda al cual pertenece) al Agente Filtrador correspondiente. La cantidad de veces que provee información de salida es directamente proporcional a la cantidad de páginas de resultados que genera el buscador con el cual se conecta.

Agente Filtrador

Este agente al igual que el Agente Específico de Búsqueda procesa en el entorno remoto y trabaja conjuntamente con él para obtener los resultados de un tipo de buscador determinado.

Recibe las distintas páginas de resultados de búsqueda, provenientes del Agente Específico de Búsqueda asociado, a través del único canal de entrada de información que posee (llamado canal de intra-comunicación del Grupo de Búsqueda). Luego filtra el contenido de las mismas eliminando información irrelevante, siendo esta su función principal y por la cual recibe el nombre. Finalmente envía los resultados de la búsqueda en Formato de Resultado de Búsqueda Estandar por un canal de salida hacia el Agente Compactador en la Aplicación Local (precisamente en el Sub-entorno de Búsqueda).

Es importante remarcar que este agente junto con el Agente Específico de Búsqueda trabajan en el entorno remoto buscando disminuir la carga del entorno local, ya que procesan remotamente y envían a la Aplicación Local sólo los resultados relevantes a la búsqueda.

5.4.4 Middleware

Para satisfacer la comunicación entre agentes que se ubican en la Aplicación Local y la Aplicación Remota se ha definido la capa denominada ***middleware***.

Esta capa utiliza la comunicación mediante sockets sobre TCP/IP. Para lograr establecerla es necesario que el emisor conozca el socket de destino (dirección IP y puerto) y que el receptor, ubicado en tal dirección IP, "escuche" en tal puerto.

Debido que tanto los emisores como los receptores son agentes que pueden variar el sitio de ejecución, se ha creado un agente servidor denominado Agente Servidor de Direcciones que reside en esta capa y conforma el elemento que provee el "sentido de la vista" a los agentes.

Agente Servidor de Direcciones

Este agente servidor concentra las direcciones de todos los agentes locales que necesitan comunicarse con agentes remotos y viceversa. Esta concentración se logra porque este agente corre en un servidor que posee una dirección determinada; escuchando en un puerto fijo, único y conocido por el resto de los agentes que necesitan intercambiar información. De esta forma, cualquier agente puede perder la "ceguera" y liberarse del aislamiento, recurriendo a este agente.

Internamente gestiona una lista de todas las direcciones y puertos de los servidores que necesitan ser conectados. Cualquier servidor que desee publicarse, debe informar al Agente

Servidor de Direcciones la dirección en que se ubica y el puerto en que escucha. Por ejemplo, para que el agente que se encarga de recolectar datos de Altavista pueda ser alcanzado por la aplicación local en el desarrollo de una búsqueda, deberá previamente haber enviado su solicitud de publicación al Agente Servidor de Direcciones para que su dirección se encuentre disponible para ser recuperada por el Agente Coordinador. ¿Cómo averigua el Agente Coordinador dicha dirección? A través de una solicitud de petición al Agente Servidor de Direcciones.

De lo anteriormente expresado, se desprenden los dos servicios que son provistos por este Agente Servidor de Direcciones: la *solicitud de publicación de dirección* y la *solicitud de petición de dirección*. El primer servicio consiste en que los agentes, al iniciar su ejecución, informan al agente servidor, a través de un mensaje KQML, donde esperan mensajes. El segundo servicio se basa en que los agentes que requieren comunicarse con otro agente que no se encuentra en el mismo espacio de direcciones, envíen un mensaje KQML solicitando la dirección y puerto remotos; ante esta solicitud el Agente Servidor de Direcciones responde con un mensaje KQML conteniendo los datos solicitados (siempre y cuando la dirección solicitada ya esté almacenada en el agente servidor).

Diagrama de flujo de direcciones

En el diagrama de flujo de la Figura 5.2 está reflejado como todos los agentes intervinientes en una búsqueda hacen uso de los dos servicios prestados por el Agente Servidor de Direcciones, publicación y petición de direcciones. Posteriormente se describe cada uno de los flujos.

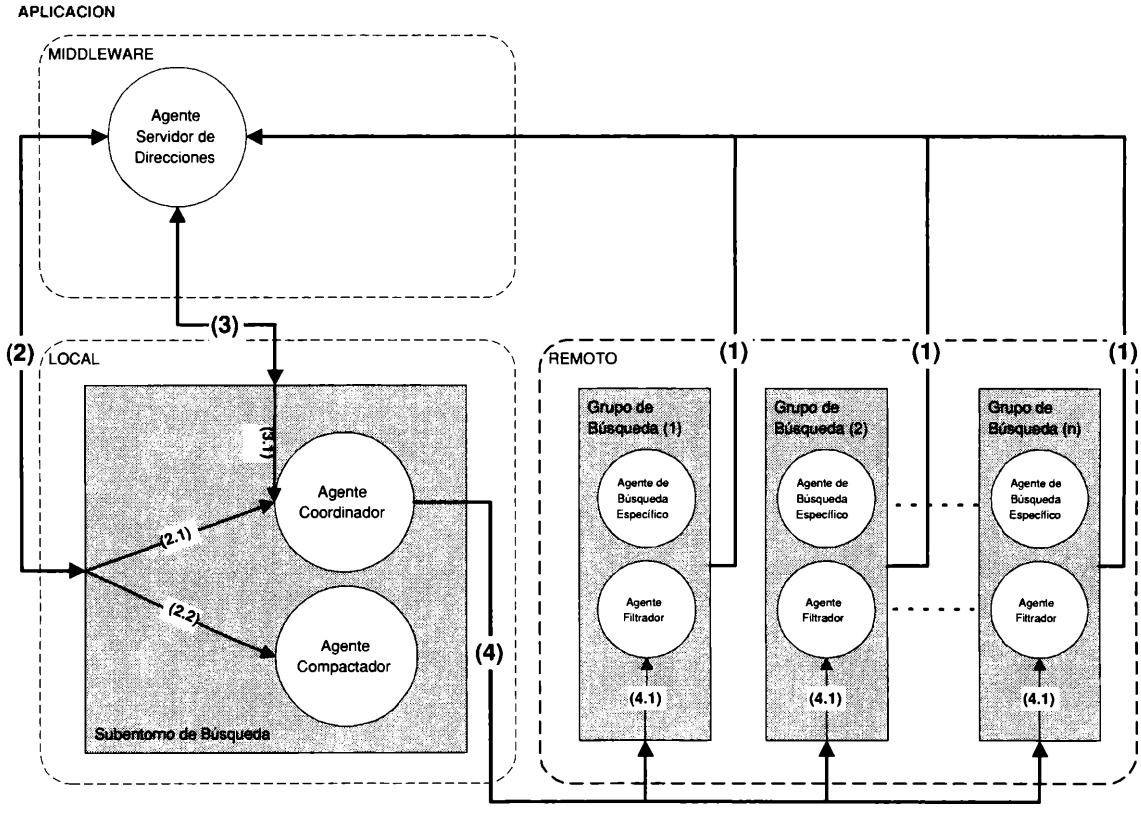

Figura 5.2: *Diagrama de flujo de direcciones.*

- **(1)** Cuando cada Grupo de Búsqueda inicia su ejecución publica la dirección y puerto por donde esperará mensajes de la Aplicación Local. Esto se realiza a través de un mensaje KQML que contiene estos datos, enviado a la dirección y puerto del Agente Servidor de Direcciones conocidos por todos los agentes. El Agente Servidor de Direcciones almacena la dirección y puerto informados en su catálogo interno.

- **(2)** Cuando un Sub-entorno de Búsqueda se crea, a partir de una búsqueda realizada por el usuario, solicita al Agente Servidor de Direcciones una dirección y puerto en donde escuchará el Agente Compactador.

- **(2.1)**[4] El Sub-entorno de Búsqueda informa al Agente Coordinador la dirección y puerto obtenidos.

- **(2.2)**[4] El Sub-entorno de Búsqueda informa al Agente Compactador donde debe esperar por mensajes de resultados provenientes de los Agentes Filtradores de la Aplicación Remota.

- **(3)** El Sub-entorno de Búsqueda realiza una petición al Agente Servidor de Direcciones de las direcciones y puertos de cada uno de los Grupos de Búsqueda que deberán intervenir en la búsqueda (si el usuario solicita una búsqueda en BIPI y WebCrawler, se solicitarán sólo los datos de los Grupos de Búsqueda de ambos). En el mensaje de petición se envía la dirección y puerto donde debe enviarse la respuesta.

- **(3.1)**[4] El Sub-entorno de Búsqueda envía al Agente Coordinador las direcciones y puertos obtenidos en el ítem anterior.

- **(4)** En dos mensajes consecutivos, el Agente Coordinador, envía a todas las direcciones obtenidas en el item 3.1 (correspondientes a los buscadores intervinientes en la búsqueda), la frase de búsqueda (no tenida en cuenta en este diagrama) y la dirección de espera de resultados del Agente Compactador.

- **(4.1)**[4] La dirección y puerto obtenidos por cada Grupo de Búsqueda es informada a su Agente Filtrador, para que éste pueda enviar los resultados obtenidos al Agente Compactador de la Aplicación Local.

5.5 Sistema de Gestión de Base de Datos

Este componente conforma la capa inferior de la arquitectura y es el que posee la base de conocimiento del buscador. El SGBD es el encargado de resolver las peticiones provenientes de la capa de Aplicación, más precisamente del Agente de Búsqueda Continua, el Agente de Incorporación de URL, el Agente Indexador, el Agente Autónomo de Mantenimiento y aquel Agente Específico de Búsqueda encargado de la interacción con el buscador interno de BIPI.

Para poder resolver la peticiones de la capa superior, en la Base de Datos se almacenan diferentes tipos de información, así como también procesos internos que permiten la resolución de las peticiones y aquellos concernientes a la manutención de la información almacenada. Esta información puede clasificarse de la siguiente forma:

[4]Esta comunicación no utiliza sockets ya que se realiza entre agentes que ejecutan en el mismo espacio de direcciones.

[4]Esta comunicación no utiliza sockets ya que se realiza entre agentes que ejecutan en el mismo espacio de direcciones.

- URLs pendientes de indexación: estas son almacenadas mediante peticiones generadas por el Agente de Incorporación de URL, el Agente de Búsqueda Continua o el Agente Autónomo de Mantenimiento.

- Desglose de páginas: conjunto de información incorporada por el Agente Indexador cuando desglosa las páginas en el proceso de indexación. Entre estos datos se encuentran los datos propios de las páginas (título, descripción, enlaces, cuerpo, etc.) y aquellos calculados por el sistema (penalizaciones, inter-referencias de páginas almacenadas, etc.).

- Parametrización del sistema: conjunto de parámetros establecidos por el Usuario Administrador, que permiten que el sistema sea configurable en forma dinámica, siendo esta una ventaja muy importante ya que impide que los usuarios que incorporan URLs infieran el comportamiento interno del buscador.

Hasta el momento se ha explicado la lógica de como BIPI, en su función de meta-buscador, puede satisfacer una solicitud de búsqueda en base a derivaciones hacia diferentes buscadores. Todos los buscadores intervinientes colaboran y retornan sus resultados que, como ya vimos, son consolidados y retornados hacia el usuario solicitante. Entre estos buscadores desempeña su función el buscador interno que provee BIPI, comportándose como uno más del conjunto de buscadores colaborativos.

El buscador interno de BIPI exporta la funcionalidad típica de un buscador: *permite realizar la búsqueda de una frase (que en particular este buscador utiliza el Formato de Búsqueda Estandarizado) y retorna los resultados pertinentes ordenados en base a la relevancia.* Para realizar esta función nos encontramos con una fuerte interacción con el Sistema de Gestión de Base de Datos dado que gran parte de la lógica correspondiente a la realización de una búsqueda, sobre este buscador, se basa en los procesos que provee. En el SGBD se almacenan los procesos intervinientes en la realización de una búsqueda a partir de los parámetros provenientes del Agente Específico de Búsqueda que interactúa con el buscador interno de BIPI.

Capítulo 6

Interfaz y funcionalidad

El capítulo anterior a dado base a un metabuscador que debe ser soportado por una interfaz capaz de interactuar con los usuarios comunes y con los usuarios administradores. Para la definición de esta interfaz fue necesario crear cuatro páginas básicas.

En las secciones siguientes describiremos tres páginas que dan cabida a la interfaz del Usuario Común y una página de administración de la parametrización de los factores que influyen en el ranqueo de páginas.

Se ha buscado definir una interfaz fácil de usar, intuitiva, con una distribución de componentes similar al resto de los buscadores del mercado, respetando las necesidades básicas de todo buscador, agradable a la vista y, a la vez, funcionalmente potente.

La totalidad de las páginas utiliza la tecnología JSP (ver Apéndice B), ya que deben generarse dinámicamente en base a peticiones de los usuarios que interactúan con ellas.

6.1 Home Page

Esta página (ver Figura 6.1) es el puntapié inicial de acceso al buscador. Tiene ciertos sectores comunes a todas las páginas a las que tiene acceso el Usuario Común. Estos sectores incluyen:

- **Sector superior: Especificación de búsqueda**

  En este sector se especifican los parámetros de búsqueda, o sea los buscadores y la frase de búsqueda a utilizar.

  Por un lado se encuentran los checks correspondientes a cada uno de los buscadores que pueden elegirse para realizar la búsqueda que, en la definición actual del prototipo, son cuatro: BIPI (ver Capítulo III), Yahoo!, WebCrawler y Altavista.

  También existe un espacio para la escritura de la frase de búsqueda, que tendrá que establecerse siguiendo el Formato de Búsqueda Estandarizado (ver Sección 9.1).

  Por último se encuentra el botón que dispara la búsqueda. En realidad este botón pasa el control de la búsqueda y la presentación de resultados a la página de búsqueda (ver Sección 6.2), siendo el único medio de acceso a esa página.

- **Sector izquierdo: Menú de opciones**

  Se brinda acceso a la "Home Page" y a la página de "Agregado de URLs" (ver Sección 6.3). Este es el lugar adecuado para agregar enlaces a funcionalidades futuras ya que brinda fácil alcanzabilidad a todas las páginas del sitio, convirtiéndose en una especie de mapa.

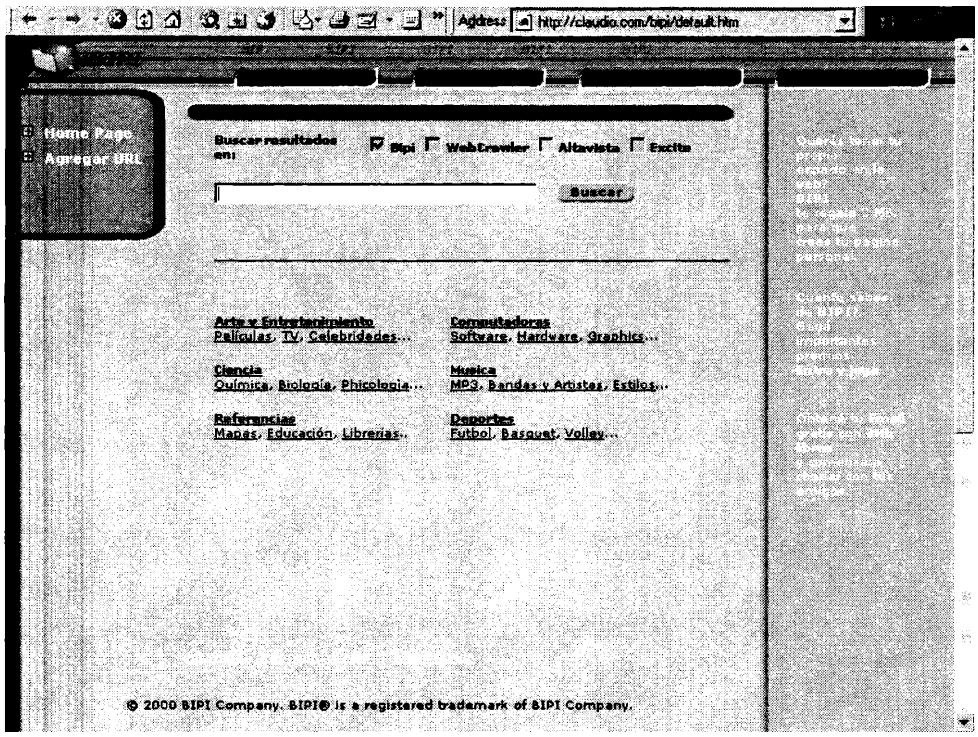

Figura 6.1: *Home Page de BIPI.*

- **Sector inferior: Otros motores**

  Existen diferentes enlaces a otros buscadores, entre los cuales se encuentran los que interactúan con BIPI.

6.2 Búsqueda

Como explicamos anteriormente, a esta página (ver Figura 6.2) se accede a través del botón de inicio de la búsqueda. Es dinámica ya que su contenido varía de acuerdo a la búsqueda realizada por el usuario, requiriendo de parámetros para su formación al instante. Los parámetros de entrada son siete y se agrupan en la siguiente clasificación:

- **Frase de búsqueda:** es el dato que incluye el usuario en la caja de texto de búsqueda.
- **Buscadores:** son cuatro parámetros que indican para cada uno de los buscadores si debe o no ser utilizado en la búsqueda.
- **Resultados a mostrar:** existen dos parámetros no obligatorios adicionales que establecen el rango del conjunto de resultados a mostrar. Uno indica el *primer resultado* (parámetro `first`) a mostrar en la lista y el otro la *cantidad* (parámetro `slice`) a mostrar a partir de `first` (p.ej: si `first` fuera 20 y `slice` fuera 10, se mostrarían los resultados 20-30 del conjunto de resultados total). Al ser opcionales tienen valores por defecto: 1 para `first` y 10 para `slice`.

Mediante la utilización de los parámetros indicados, pasándoselos al software, la página dinámica logra generarse con una lista de resultados que se muestran en la parte central

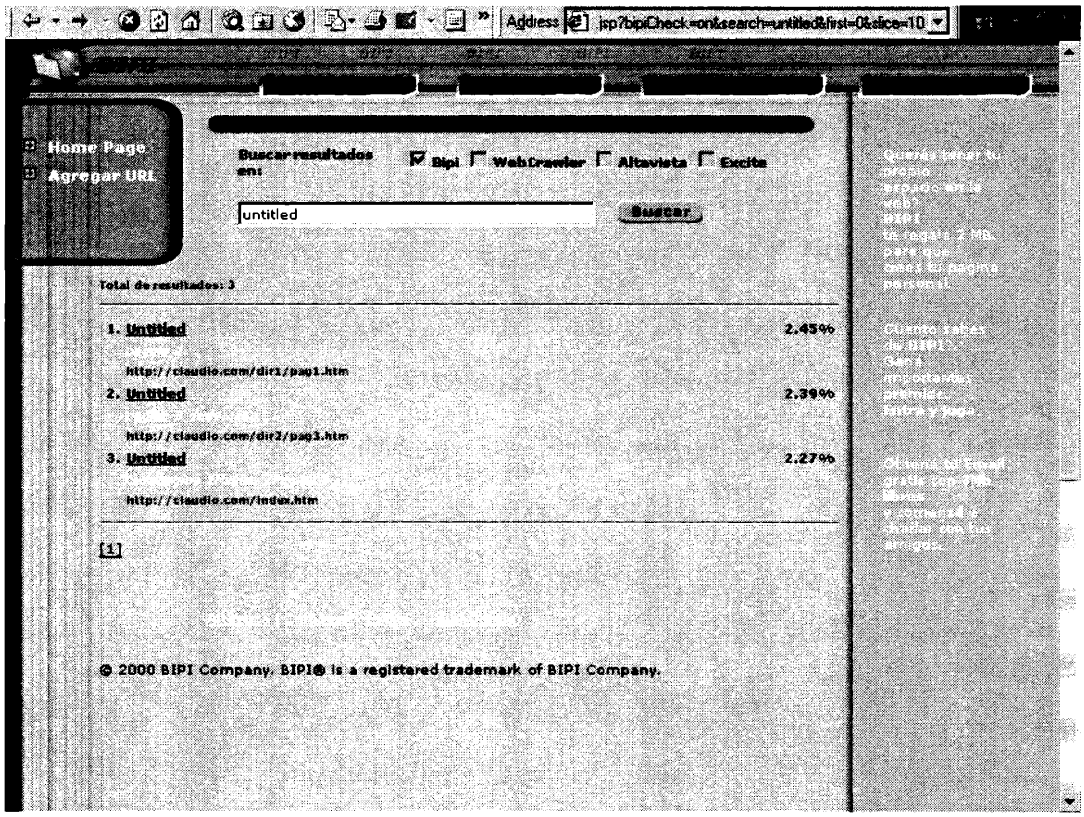

Figura 6.2: *Página de búsqueda y presentación de resultados.*

de la misma. Esta lista de resultados está ordenada por relevancia, y contiene el título, la descripción y un enlace a la página, para cada una de las páginas resultantes.

Si la cantidad de resultados superase a la cantidad especificada por **`slice`**, aparecerá debajo de la lista de resultados una serie de accesos directos al resto de las páginas de resultados (actualmente se muestran a lo sumo 10 accesos a 10 páginas de resultados, cada una con la cantidad de resultados establecida por **`slice`**. Este valor es fácilmente modificable).

6.3 Agregar URL

Con esta tercer página (ver Figura 6.3) se completa el conjunto de páginas a las cuales tiene acceso el Usuario Común. Provee una funcionalidad adicional a una máquina de búsqueda pura. Permite que el usuario "sugiera" al spider que visite una página indicada por la URL que ingresa en la caja de texto correspondiente.

Como dato adicional se solicita obligatoriamente el ingreso del correo electrónico del usuario para posibles necesidades de comunicación (actualmente no se utiliza ningún tipo de gestión de este valor). La obligatoriedad del dato se indica mediante **(*)** al final del nombre del campo, como establece la leyenda al pie de la página.

Existen tres datos opcionales en esta pantalla: la descripción de la página, las palabras clave y un valor lógico que indica si se deben seguir los enlaces a partir de la URL ingresada. Si el usuario no ingresara valores para estos campos se tomará la descripción del metatag "description", las palabras clave del metatag "keywords" y se indexará únicamente la página indicada por la URL.

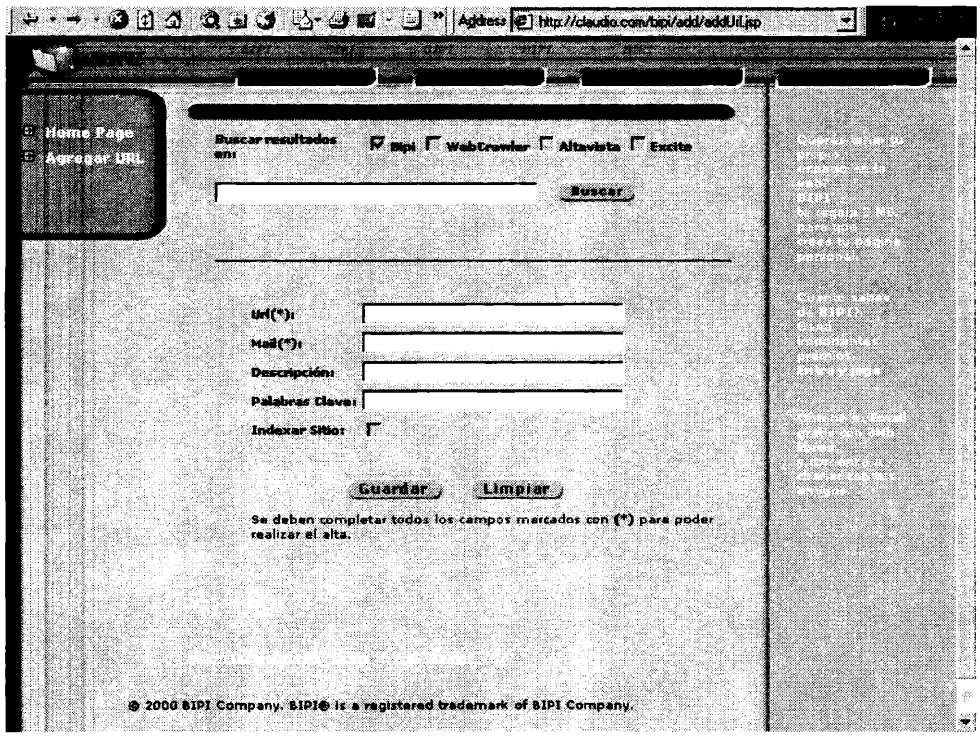

Figura 6.3: *Página de incorporación de URLs.*

Para que estos datos sean procesados es necesario confirmarlos a través del botón "Guardar". Si el usuario se equivocase en alguno de los datos, o deseara sugerir una nueva URL a visitar, dispone del botón "Limpiar" para borrar el valor de los campos.

6.4 Administración de la parametrización

Esta página (ver Figura 6.4) es únicamente accedida por el Usuario Administrador y está diseñada con el objetivo de gestionar los factores que influyen en los cálculos de ranqueo y relevancia que realiza el motor de búsqueda. De aquí la importancia de la restricción de acceso.

En el sector central superior se muestran los factores y parámetros generales. A continuación se da una breve descripción de cada uno de ellos.

- **Factor penalizaciones:** este valor es un porcentaje. Si es 100 se producirá un descuento del 100% de la penalización sobre la relevancia acumulada, si es 0 las penalizaciones no se tendrán en cuenta (no afectando la relevancia).

- **Factor enlaces:** este porcentaje afectará (igual que el porcentaje de las penalizaciones, pero en forma positiva) el peso de la popularidad de enlaces en la relevancia final.

- **Factor relevancia tags:** es un porcentaje que dará más o menos peso a la relevancia obtenida por los descriptores.

- **Factor cantidad de palabras:** porcentaje del peso obtenido por la cantidad de palabras encontradas en la página.

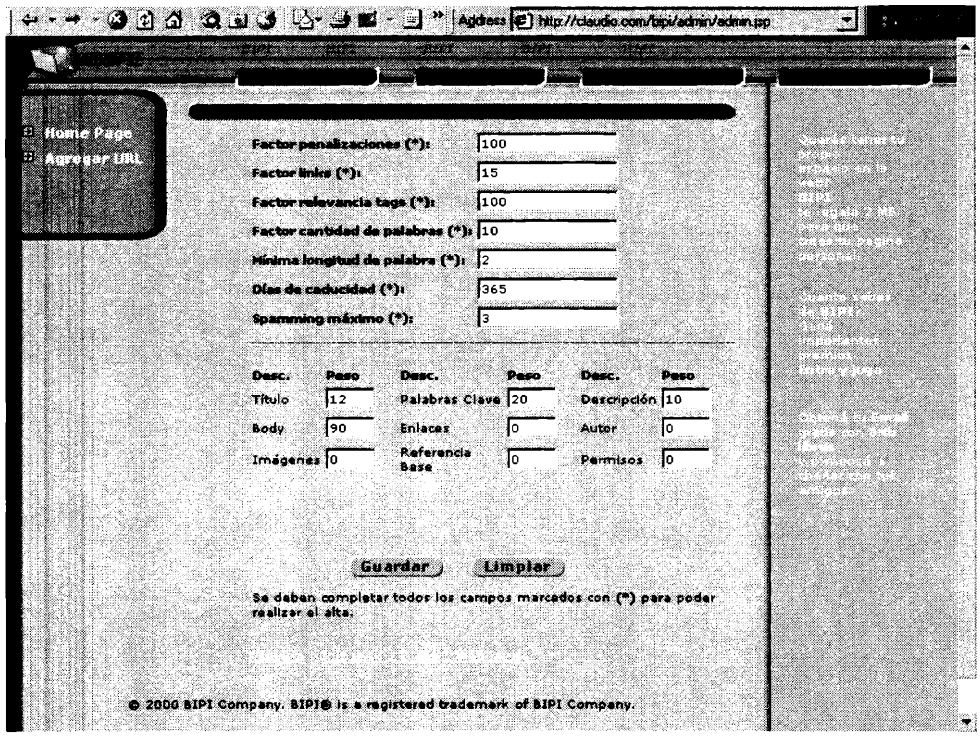

Figura 6.4: *Página de administración.*

- **Mínima longitud de palabra:** este número indica la cantidad mínima de letras que debe tener una palabra para ser indexada.
- **Días de caducidad:** este valor lo utiliza la regla del Criterio de Expiración de Páginas cuando procesa el Agente Autónomo de Mantenimiento. Determina la cantidad de días que debe transcurrir entre las revisiones de cada página.
- **Spamming máximo:** al momento de determinar si una palabra está siendo utilizada para realizar spamming, se cuenta la cantidad de veces que aparece en la página y, si supera el valor de este factor, se aplica una penalización por spamming.

En el sector central inferior se muestran todos los grupos de descriptores (p.ej: título, descripción, cuerpo, etc.) que tiene en cuenta el motor de búsqueda. Para cada uno de estos grupos se establece el peso que tiene sobre el resto. El valor definitivo será calculado internamente y resultará en un porcentaje. Por ejemplo, si al grupo título se le da un peso de 100, al grupo cuerpo un peso de 50 y el resto de los grupos no tuviera peso, se estaría indicando que el 66,66% de la relevancia para los descriptores estará dada por el título y el 33,33% restante la dará el cuerpo.

Para que los factores y pesos de los distintos campos sean procesados es necesario confirmarlos a través del botón "Guardar". La actualización no se realizará si los campos marcados con **(*)** no se completan.

Parte III

BIPI: motor de búsqueda

Capítulo 7

Modelo de almacenamiento

BIPI es un metabuscador y al mismo tiempo es una máquina de búsqueda pura, con la posibilidad de agregado de URLs por parte del usuario. El metabuscador fue descrito en la parte anterior (ver Parte II) y en ésta comenzaremos a describir el motor de búsqueda.

Al momento de definir el modelo de búsqueda de BIPI se examinaron las características de las máquinas de búsqueda más populares del mercado. Realizando una mezcla de las que nos parecieron las características más adecuadas llegamos a una definición que se soporta con el siguiente modelo.

7.1 Descripción de modelo

Todos los buscadores realizan un almacenamiento físico de ciertas características de las páginas que indexan. Algunos sólo mantienen referencias a ellas y detalles dados por los usuarios (directorios) y otros almacenan la definición completa de la página (quizás en texto plano o desglosada).

El spider navega a través de la web buscando páginas y siguiendo enlaces de ellas hacia otras. ¿De dónde obtiene las páginas a indexar? ¿Dónde almacena la información recuperada de cada página? ¿Qué información se debe almacenar de cada página? Cuando un usuario realiza una búsqueda se retorna una serie de resultados ordenados por relevancia. ¿Qué resultados recupera? ¿Cómo determina el orden de los mismos?. La respuesta a esta serie de preguntas se encuentra en la lógica de la *Base de Datos* y los datos almacenados en ella.

La Base de Datos es pieza clave dentro de la arquitectura del motor de búsqueda. Está compuesta por una serie de tablas que pueden clasificarse en distintos grupos de acuerdo a su funcionalidad y utilización. Esta agrupación está dada por las posibilidades inferidas de la serie de preguntas del párrafo anterior: páginas a indexar e indexadas, parámetros de búsqueda, datos a tener en cuenta en la indexación de cada página y datos usados en el cálculo de relevancia.

Se ha implementado una Base de Datos Relacional, mayormente con normalización BCNF, en la cual se almacenan ciertos datos redundantes (p.ej: el texto plano de la página es guardado y al mismo tiempo su desglose) y otros campos con datos calculados (que podrían obtenerse mediante cálculos sobre ciertas tablas), que se almacenan en busca de lograr una mayor performance en la recuperación de resultados (p.ej: cantidad de palabras de cada página).

En las secciones siguientes se mostrará cada uno de los grupos, la función que cumplen y las tablas que le sirven de sustento.

7.2 Parametrización

Este grupo permite establecer una serie de valores que son tenidos en cuenta al momento de la indexación, búsqueda y mantenimiento de las páginas. Todos estos parámetros son mantenidos a través de la página de administración.

Se sustenta en una única tabla denominada *Parametros*. En esta tabla se encuentran los campos que permiten almacenar los factores de penalizaciones, enlaces, relevancia de descriptores y cantidad de palabras; la palabra mínima a indexar, los días de caducidad entre revisiones y la cantidad de no accesos (este valor indica la cantidad máxima de intentos de acceso fallidos que se realizarán a una página antes de eliminarla). Para una descripción más detallada de estos ítemes referirse a la Sección 6.4.

Existe una segunda tabla que hace a la parametrización pero que no es accesible a través de una interfaz. En esta tabla se almacenan aquellos descriptores que contienen texto despreciable o descartable en el momento de la indexación. Estos descriptores son, entre otros, los de comentarios y scripts. De cada uno de los descriptores se almacenan el texto de inicio y el texto de fin (p.ej: para los comentarios se almacena <!- y ->).

7.3 Desglose de páginas

La información de cada página indexada por el spider, es dividida y almacenada en diferentes tablas, para su posterior recuperación.

Los datos principales de la página: URL, título, descripción, sitio, fecha de alta, fecha de último acceso, fecha de último refresco, estado de la página y cantidad de palabras son almacenados en la tabla correspondiente a la cabecera de la página.

En tablas satélites de la cabecera se almacenan, al momento de la indexación, datos relativos a cada página. A continuación se da una descripción de ellos.

- **Cuerpo:** se guarda completamente el texto de la página comprendido entre los descriptores <BODY> y </BODY>.
- **Enlaces a otras páginas:** se calculan las referencias cruzadas entre páginas ya almacenadas en la Base de Datos.
- **Datos de spamming:** se determinan datos de spamming en base al valor "Spamming máximo".
- **Cálculo de penalizaciones:** basándose en los datos de spamming se calculan, por página, las penalizaciones que influirán en la relevancia de la página en cada consulta de búsqueda (en base al "Factor penalizaciones").
- **Textos de los distintos descriptores:** se almacena el texto de cada uno de los descriptores indicados por la tabla de *Tags* (ver Sección 7.4).
- **Texto plano de la página:** es guardado todo el texto de la página.

7.4 Gestión de descriptores

Este grupo está comprendido por dos tablas muy importantes dentro del modelo de almacenamiento y colaboran en la parametrización del motor. En conjunto, almacenan los distintos grupos de descriptores y los descriptores relevantes al sistema que componen cada uno de esos grupos.

El almacenamiento de los grupos se realiza en una tabla denominada *TagGrupos*. Esta tabla almacena la relevancia de cada uno de los grupos de descriptores, modificable a través de la página de administración (ver Sección 6.4). Además de estos valores se almacenan los pesos que influyen en el ranqueo (datos calculados en base a las relevancias de cada grupo).

La segunda tabla es la que almacena los descriptores propiamente dichos. No se mantienen todos los descriptores que utiliza el lenguaje HTML, sino aquellos que se necesitan utilizar en algún cálculo dentro del motor de búsqueda o se utilizan en el chequeo de sintaxis del Parseador. Por cada descriptor se tiene:

- **Id grupo:** referencia al grupo de descriptores al que pertenece.
- **Id descriptor:** identificador único de descriptor.
- **Descripción:** palabra clave por la que se identifica el descriptor (puede ser el nombre de un descriptor HTML o un valor del parámetro NAME de los metatags).
- **Es repetible:** valor lógico que indica si el descriptor puede estar más de una vez dentro de la página.
- **Es doble:** valor lógico que determina si el descriptor contiene el valor relevante en un segundo parámetro.
- **Descripción doble:** si el valor anterior es verdadero, aquí se indica el nombre del parámetro que contiene el valor relevante del descriptor.
- **Es spamming:** valor lógico que establece si el texto de este descriptor debe ser tenido en cuenta para el cálculo de spamming.
- **Es metatag:** valor lógico que indica si el descriptor es un metatag.
- **Es enlace:** valor lógico que identifica aquellos descriptores que contienen referencias a otras páginas y que pueden ser seguidos para la indexación en profundidad.

Es importante notar que al momento de definir la estructura de esta última tabla se ha buscado una forma que sirva de fácil parametrización al motor de búsqueda. De tal forma que si decidiéramos incorporar un nuevo descriptor a tener en cuenta, sólo deberíamos insertar un nuevo registro que contenga los valores necesarios y correctos en cada uno de los campos.

7.5 Páginas pendientes

Todas aquellas URLs que los usuarios ingresan a través de la página de incorporación de URLs (ver Sección 6.3) y todas aquellas provenientes del Agente de Búsqueda Continua (ver Agente de Búsqueda Continua dentro de la Subsección 5.4.2) se almacenan con el objetivo de que el spider las recupere y las indexe con posterioridad.

De cada página pendiente se almacena:

- **URL**
- **Correo electrónico:** del usuario que propone su indexación.
- **Descripción:** descripción introducida por el usuario a través de la interfaz.
- **Palabras clave:** palabras insertadas por el usuario.

- **Fecha de inscripción:** fecha de inserción del registro en la tabla.
- **Fecha de orden:** fecha de actualización del registro. Se la denomina fecha de orden debido a que las páginas pendientes a indexar se van recuperando ordenadas por esta fecha en forma ascendente. Este dato sirve para que sucesivas incorporaciones de la misma URL desplacen el pedido hacia el final de la cola.
- **Fecha de cierre:** fecha de indexación exitosa de la página.
- **Cantidad de no accesos:** contador de accesos fallidos a la página. Este valor hará que una página sea eliminada de la Base de Datos cuando supere el valor establecido en la tabla de parámetros (ver Sección 7.2).
- **Marca de indexación en profundidad:** este valor lógico indica si se deben seguir los enlaces de la página.
- **Estado:** este campo acepta los siguientes valores:
  - Pendiente: aún no indexada.
  - Indexada: indexación exitosa.
  - Rechazada: no fue posible su indexación. El detalle de la causa se encuentra en el campo "Descripción del estado".
  - Analizando (Bloqueada): un spider está indexando la página. Sirve para el control de la concurrencia de spiders.
  - No pudo grabarse: la página fue recuperada pero hubo un fallo inesperado en el proceso de almacenamiento.
  - Pendiente - No accesible: no se pudo acceder a la página, pero va a intentarse nuevamente en el futuro.
- **Descripción del estado:** detalle del error en caso de rechazo de página.

Existe una tarea que recorre el conjunto de páginas pendientes de indexación (en estado Pendiente - No accesible) y determina si el valor del campo "Cantidad de no accesos" llegó al valor indicado por el parámetro correspondiente en la tabla de parámetros. En caso de llegar a este valor la tarea se encarga de eliminar la información de la página relacionada con la URL pendiente, sino la pasa al estado Pendiente para que el spider reintente accederla. Existe una planificación de esta tarea para que ejecute una vez por día (aunque debería determinarse en base a la carga del buscador).

Capítulo 8

Indexación de páginas

El motor de búsqueda de BIPI tiene un agente dedicado exclusivamente al proceso de indexación: el *Agente Indexador*. Este agente trabaja continuamente realizando la búsqueda, recuperación y almacenamiento de páginas en la Base de Datos.

El objetivo de este capítulo es explicar el flujo de la información de una página en las distintas etapas que hacen a su indexación. También se detallarán una serie de funcionalidades del motor que son relevantes en este procesamiento que realiza el spider.

8.1 Pasos en la indexación

En las tres subsecciones siguientes se especifica un detalle de cada una de las tres etapas seguidas en el tratamiento de una página a indexar. Estas pasos van desde la obtención de una URL, pasando por la recuperación de la página subyacente, y llegando al almacenamiento de la información que ella contiene.

8.1.1 Obtención de URL

Con este paso se inicia el proceso de indexación. Cada una de las URLs a indexar se obtienen de la tabla de páginas pendientes de la Base de Datos (ver Sección 7.5), mediante una función que recupera la siguiente URL a indexar que esté en estado Pendiente y que se ubique al principio de la cola de indexación (esta cola está ordenada por la fecha de orden de solicitud). Esta función a su vez marca como bloqueada a la solicitud, de tal forma que otro spider que ejecute concurrentemente se encargue de otro pedido.

La función puede no retornar una URL, ya que podemos llegar a un instante de tiempo en el cual todas las URLs almacenadas en la tabla de páginas pendientes estén procesadas (en estado distinto de Pendiente). En esta caso el spider reintentará obtener una URL en estado Pendiente un número finito de veces que es modificable por el administrador (es un valor de entorno del Agente Indexador).

Algo similar ocurre cuando el agente detecta que la Base de Datos no está accesible. El spider intentará acceder una cantidad fija de veces establecida por otra variable de entorno parametrizable.

8.1.2 Recuperación

A partir de la URL obtenida en la etapa anterior, el Agente Indexador recuperará el texto de la página indicada por la URL. Si no puede ser recuperado se intentará acceder hasta llegar a un valor establecido a través de una variable de entorno del spider. Si se alcanza este

valor la página será marcada como rechazada por acceso y el spider recomenzará el proceso de indexación obteniendo una nueva URL.

Una vez que se obtiene el texto completo de la página se pasa a través de un procedimiento que elimina todos aquellos textos que no son tenidos en cuenta en la indexación y las búsquedas. Estos textos son eliminados en base a los datos especificados en la tabla de *Comentarios* (ver Sección 7.2).

El texto, sin los datos irrelevantes, es verificado antes de ser enviado a la siguiente etapa de procesamiento. Esta verificación consiste en un chequeo de sintaxis "superficial" que sólo valida que todo descriptor de apertura tenga su correspondiente de cierre en el orden adecuado (p.ej: <P><B>...</B></P> y <P>
...</P> son extractos de texto sintácticamente correctos, y <P><B>...</P></B> no lo es). Si este chequeo no es superado, la URL es rechazada por sintaxis y marcada como tal en la tabla de páginas pendientes; y el proceso vuelve a comenzar.

8.1.3 Almacenamiento

En base a la forma en que almacena las páginas indexadas el motor de BIPI, podemos decir que utiliza un método de almacenamiento desglosado, o sea que la página que proviene de la etapa de recuperación es dividida en partes y cada una de esas partes son almacenadas por separado en la Base de Datos.

Las partes que almacena el spider, y la forma en que son distribuidas entre distintas tablas, fueron detalladas en la Sección 7.3.

Los pasos que se siguen en el desglose y almacenamiento conforman el tópico de esta sección y se describen a continuación.

- **Almacenamiento de la cabecera:** antes de iniciarse el desglose de la página en sí, se verifica que la página no esté almacenada en la Base de Datos; si no está los siguientes pasos realizarán inserciones, de lo contrario serán actualizaciones.
- **Almacenamiento del texto completo:** el texto de la página, tal cual es recuperado, es almacenado en una tabla.
- **Recuperación de descriptores válidos:** se cargan en memoria todos los descriptores que servirán para el desglose de la página.
- **Desglose de descriptores:** se recorre la página y se ubican, en una estructura volátil, todos los textos de los descriptores que fueron identificados en el paso anterior.
- **Procesamiento de descriptores:** consiste en verificar y/o almacenar en memoria permanente cada uno de los textos de los descriptores que fueron cargados en la estructura volátil. De acuerdo al grupo al que pertenece el descriptor el procesamiento varía. En adelante se describen las etapas seguidas en forma secuencial.
  - **Exclusión de indexación:** si el descriptor es el metatag "robots", se verifica una posible evitación de indexación. Si el contenido de este metatag fuera NOINDEX el proceso de indexación de esta página sería inmediatamente revocado.
  - **Obtención de la referencia base:** si el descriptor es "Base", se obtiene la referencia base de la página para su utilización dentro de este proceso de indexación.
  - **Almacenamiento de texto de descriptores:** si se llega a este estado del procesamiento de descriptores, el texto ubicado en la estructura volátil es almacenado como parte de la página en la tabla correspondiente.

  - **Almacenamiento de enlaces y cálculo de referencias cruzadas:** cada enlace identificado en la página es almacenado en la tabla de enlaces de la página. Por cada una de estas inserciones se calculan las referencias que este hace hacia otras páginas de la Base de Datos.
  - **Almacenamiento de páginas pendientes:** si el proceso que se está llevando a cabo para una página es del tipo de "indexación en profundidad" y el metatag "robots" no indica NOFOLLOW, los enlaces son almacenados en la tabla de páginas pendientes para su posterior tratamiento por parte del spider.

- **Calculo de spamming fijo:** una vez que cada parte de la página ha sido almacenada, se procede al cálculo del spamming. En este paso sólo se calcula una penalización fija por repetición de palabras, que influirá al momento de la búsqueda.

Si este desglose no puede proseguir con el almacenamiento debido a una falla en la comunicación con la Base de Datos, se actualiza el estado de la página a rechazada indicando que hubo un error en el momento del grabado. Si por el contrario, el desglose llega a su fin correctamente, la página se da por procesada y se actualiza el estado de la misma en la Base de Datos, llegando a su cierre el proceso de indexación.

8.2 Evitando la indexación

El motor de búsqueda de BIPI obedece al ***Estándar de Exclusión de Robot*** (ver Sección 2.5). Por ello antes de iniciar el procesamiento de una página verifica la existencia del archivo "robots.txt" que debe residir en el nivel más alto del espacio de documentos del servidor.

Si la página pertenece a un dominio que ya forma parte de la información residente en la Base de Datos, el Agente Indexador no necesita verificar el archivo "robots.txt", ya que los datos de este archivo se almacenan localmente en tablas especializadas.

A pesar de que exista este archivo, el protocolo establece que debe indicarse mediante un texto con un formato específico qué robot quiere mantenerse fuera del alcance de ciertos archivos y/o directorios. El modo de indicar que se desea que el motor de búsqueda de BIPI no realice indexación, es mediante el nombre de él: ***bipi***. O sea, que cualquier prohibición a agentes mediante "*" o mediante "bipi", hará que el motor de búsqueda respete las prohibiciones establecidas.

Además de esta exclusión utilizando el archivo "robots.txt", puede usarse la exclusión de páginas en particular mediante el metatag "robots". El motor reconoce las prohibiciones NOINDEX y NOFOLLOW, que respectivamente evitan la indexación de la página que contiene el metatag o la indexación en profundidad (el seguimiento de enlaces). No contempla la utilización de NOIMAGEINDEX, la tercer palabra clave del metatag, porque no realiza indexación de imágenes, videos o sonidos.

8.3 Reindexación

Las páginas indexadas se marcan como procesadas una vez que se cumplen todos los pasos descritos en la Sección 8.1. Al mismo tiempo se actualiza la fecha de cierre de la misma con la fecha y hora del momento de cumplimiento de la última de las tres etapas.

A medida que pasa el tiempo, una página puede ser modificada por los desarrolladores, por lo que es necesario que la información almacenada en la Base de Datos sea actualizada también, para que las búsquedas brinden resultados correctos (p.ej: inicialmente la página

puede contener información sobre frutas varias y entre ellas información de la frutilla. Supongamos que al tiempo el desarrollador decide que la información de la frutilla es errónea y decide eliminarla de la página. Si un usuario buscara la frase "+frutas +frutilla", aparecería esta página como resultado a pesar de que no se trata más el tema de la misma en tal página debido a una modificación).

¿Cómo sabe el motor cuándo es el momento adecuado para actualizar esta información?. No lo sabe, quizás podrían hacerse estadísticas de refresco de ciertas páginas para visitarlas a unas con mayor regularidad que a otras, pero inicialmente BIPI maneja un tiempo de refresco fijo. Este refresco es establecido a través de la página de parametrización tal cual se describió en la Sección 7.2.

El refresco utiliza como unidad los días, por lo que cada `n` días, se indica al spider que debe revisitar la página para su actualización. El elemento encargado de indicar al spider que debe revisitar una página es el Agente Autónomo de Mantenimiento, que recorre la lista de páginas almacenadas en la Base de Datos y, aplicando el Criterio de Expiración de Páginas, decide cuales deben ser actualizadas. Se indica cuales requieren de actualización colocando los datos adecuados en la tabla de páginas pendientes (el spider leerá de esta página en el futuro y revisitará la página).

8.3.1 Criterio de Expiración de Páginas

Este criterio está definido en una función en la Base de Datos. Esta función requiere un identificador de página como entrada y retorna verdadero o falso según sea necesaria la actualización de la información de la misma.

El procedimiento consiste en comparar la "Fecha de cierre" de la página a verificar (dato almacenado en la cabecera de la página) con la fecha del momento de comparación. Si la diferencia entre ambas supera el parámetro "Días de caducidad" (de la tabla de parámetros) se retorna verdadero, determinando que es necesario un refresco.

Capítulo 9

Búsqueda

La búsqueda es el proceso que se inicia cuando un usuario accede a alguna de las páginas a las cuales tiene acceso el Usuario Común, coloca una frase de búsqueda y presione el botón "Buscar". Esta búsqueda finaliza cuando se retorna al usuario una lista de resultados con URLs a páginas que tratan sobre el tópico establecido en la frase a buscar.

Todo este procesamiento está regido por una secuencialidad de tareas que desarrolla el motor que debe iniciarse correctamente a través de una frase de búsqueda válida.

9.1 Formato de Búsqueda Estandarizado

Existen diferentes formatos que definen los *buscadores* a través de los que un usuario puede especificar una búsqueda. Estos formatos poseen similitudes y diferencias entre los distintos buscadores, pero en resumen todos ellos permiten establecer cualquier búsqueda que un usuario desee realizar.

Por ejemplo, si un usuario deseara realizar la búsqueda de un documento que tratara sobre el dibujo animado *Batman* en el canal de televisión *Cartoon Network*, debería especificar la búsqueda de una de las siguientes formas, dependiendo de cual buscador utilizara para su resolución:

- **Altavista**

```
Batman and "Cartoon Network"
```

- **Yahoo**

```
+Batman +"Cartoon Network"
```

Debido a estas diferencias, BIPI persigue un formato estándar que hemos denominado: ***Formato de Búsqueda Estandarizado***. Este formato se basa en la sintaxis en BNF establecida en la Tabla 9.1 y a través del mismo se provee la posibilidad de realizar cualquier combinación de búsqueda. Se provee la posibilidad de realizar búsquedas en las que se establezcan distintas opciones sobre distintas palabras intervinientes en la frase; en las que se puede expresar la posibilidad de que haya palabras que deben estar obligatoriamente y otras que pueden o no estar, en las que se puede expresar que haya palabras que no deben estar en la página de resultados, etc. También se permite realizar la búsqueda de varias palabras que aparecen en la página en un orden determinado, con lo cual se permite el uso de comillas para expresar esta búsqueda.

```
<search> ::= <phrase> | <phrase> <whitespace> <search>

<phrase> ::= <sentence> | "<sentence>" | <operator> <sentence> |
<operator> "<sentence>"

<sentence> ::= <word> | <word> <whitespace> <sentence>

<word> ::= <character> | <character> <word>

<character> ::= <alphabetic> | <numeric> | <symbol>

<symbol> ::= , | . | ; | : | ! | ¡ | ? | ¿ | & | / | ' | = | < | >
| - | * | ^ | ~ | _ | @ | \ | $} | %

<operator> ::= + | -
```

Tabla 9.1: *Sintaxis BNF del Formato de Búsqueda Estandarizado.*

Ejemplos de búsquedas realizables en BIPI:

- Búsqueda de páginas que tengan la palabra Batman o la frase "Cartoon Network":

  ```
  Batman "Cartoon Network"
  ```

- Búsqueda de páginas que contengan la palabra Batman y que pueden o no tener la frase "Cartoon Network":

  ```
  +Batman "Cartoon Network"
  ```

- Búsqueda de páginas que deben tener la palabra Batman y la frase "Cartoon Network":

  ```
  +Batman +"Cartoon Network"
  ```

- Búsqueda de páginas que deben tener la palabra Batman y la palabra Robin:

  ```
  +Batman +Robin
  ```

- Búsqueda de páginas que deben tener la palabra Batman y que no deben tener la palabra Robin:

  ```
  +Batman -Robin
  ```

- Búsqueda de páginas que deben tener la frase "Cartoon Network":

  ```
  "Cartoon Network"
  ```

Cabe destacar que la frase de búsqueda aporta un factor dinámico al ranqueo del cálculo de la relevancia.

De tal modo, cuando es necesario especificar una búsqueda en BIPI, a través de la Interfaz de Usuario se deberá ingresar una sentencia de búsqueda siguiendo la especificación en BNF. Esta misma sentencia viajará hacia el coordinador de la búsqueda (Agente Coordinador) que la descompondrá para su análisis y búsqueda en los distintos buscadores que hayan sido elegidos en la interfaz. Entre esos buscadores posiblemente estará incluido el buscador de BIPI, que también recibe, como sentencia de búsqueda, frases que siguen la sintaxis del Formato de Búsqueda Estandarizado establecida.

9.2 Factores en el ranqueo

Cuando una búsqueda expresada en Formato de Búsqueda Estandarizado llega al motor de búsqueda de BIPI, todas las páginas almacenadas en la Base de Datos son evaluadas para verificar si deben ser o no presentadas como resultados de la búsqueda. Son muchos los parámetros que influyen para que una página que se encuentra indexada en la Base de Datos sea considerada como resultado y, sobre todo, que sea presentada en lo alto de la lista de resultados.

El factor principal que determina si una página es incluida en los resultados de búsqueda es el hecho de que todas las palabras marcadas como obligatorias (marcadas con un signo "+") se encuentren en la página (incluso las frases exactas, aquellas colocadas entre comillas) y que todas las palabras que no deben estar en la página (marcadas con un signo "-") no estén. Si no se cumple esto la página directamente no será incluida en la lista de resultados de búsqueda.

El resto de los factores (entre los cuales se encuentran: si la palabra fue encontrada dentro de un metatag, si la página está penalizada por spamming, si la página es popular, etc.) únicamente influirán en el ranqueo y en la posición en la cual la página aparecerá en la lista de resultados, pero no influirán en el hecho de que la página aparezca o no entre los resultados.

9.2.1 Metatags

Debemos evaluar qué tan importante son los metatags dentro del cálculo de relevancia de una página. Como ya se explicó en capítulos anteriores el texto colocado en los metatags es simplemente para contener descripciones de la página, palabras claves a ser tenidas en cuenta y algunos datos que sirven a propósitos descriptivos y de administración. Pero el texto de los metatags no se encuentra visible para el usuario que accede a la página, con lo cual no es demasiado relevante para él el texto que en ellos se encuentre.

Muchos buscadores dan baja importancia al texto que se encuentra en los metatags y dan más prioridad al texto contenido en el título y en el cuerpo. Nosotros en cambio, si bien compartimos esta opinión, decidimos que sea el administrador quien decida la prioridad existente entre metatags, título, cuerpo, texto descriptivos de imágenes, etc. Es decir que se provee al administrador la posibilidad de ir cambiando la prioridad en distintas etapas del buscador, de forma tal, que si las necesidades de búsqueda varían, también pueda variar la forma de cómo se priorizan los descriptores de la página.

La funcionalidad de cambiar la relevancia de los distintos metatags está disponible a través de la pantalla de administración de parámetros (ver Sección 6.4).

Los metatags que el sistema considera en la actualidad son: "keywords", "description", "title" y "content-type". No está limitado el hecho de que puedan ser agregados más metatags a ser tenidos en cuenta, lo cual se logra insertando nuevos registros en la tabla de descriptores.

El hecho de permitir al administrador "jugar" con la relación de relevancias entre los metatags y demás descriptores, puede llevar a que se de mucha prioridad a los metatags, con lo cual estaríamos en peligro de que los creadores de páginas tengan chances de realizar spamming, colocando repetidamente palabras muy buscadas por los usuarios de buscadores (p.ej: colocar diez veces la palabra "sexo" en el metatag "description" o "keywords"). Por lo tanto, esta flexibilidad está respaldada por una política de penalización de spamming que se detalla en la siguiente sección.

9.2.2 Spamming

La carrera por lograr que las páginas aparezcan en lo más alto de las listas de resultados de todos los buscadores, ha llevado a que la práctica del spamming sea intensiva. Esta práctica por lo general consiste en la repetición excesiva de palabras, que son muy buscadas, en sectores no visibles al usuario, de tal forma que los usuarios no las visualizan pero que los buscadores tienen en cuenta en sus ranqueos. Este "ocultamiento" va desde la incorporación de palabras con el mismo color de fondo que la página, colocándolas quizás en tamaño sumamente pequeño, hasta llegar a incluirlas en los metatags.

Debido a que esta práctica lleva a los buscadores a ranquear páginas en lo más alto, cuando en realidad no deberían estar allí, es que BIPI introduce una política de penalización de spamming.

Cuando una página es indexada se calcula la cantidad de repeticiones que existen por palabra y dicha cantidad es almacenada en la Base de Datos. Esta cantidad es tenida en cuenta para el cálculo de la penalización a ser aplicada a la página. La penalización es directamente proporcional a la diferencia entre la cantidad de palabras repetidas y el parámetro de "Spamming máximo" que fija el administrador del buscador a través de la página de administración; por lo que a mayor repetición mayor penalización. Mediante una correcta parametrización, del parámetro de spamming y de la prioridad de los metatags (lugar en el cual puede producirse spamming), puede obtenerse y mantenerse una combinación flexible en la lucha contra esta práctica.

El conteo de palabras repetidas se realiza para todo el texto de la página sin excepción. Con lo cual el spamming es controlado en la página completa.

Hasta el momento se ha explicado cómo el motor de búsqueda detecta la práctica de spamming y como obtiene los datos necesarios de la página para el cálculo de la penalización, pero otro parámetro que juega de manera crucial al momento de calcular la relevancia de una página, es el denominado "Factor de penalizaciones", que establece la fuerza que se dará a la penalización calculada de cada página. Este parámetro es cargado por el administrador y permite decidir el porcentaje con el cual la penalización afecta la relevancia calculada para la página. Por ejemplo, si en este parámetro se cargara el valor 0 cualquier penalización calculada no tendría efecto sobre la relevancia acumulada.

9.2.3 Enlaces

El motor de búsqueda de BIPI mantiene internamente la relación que existe entre las páginas que tiene almacenada en la Base de Datos. Estas interrelaciones permiten conocer la popularidad de una página dentro del catálogo del buscador.

Cuando se realiza el ranqueo de una página en una búsqueda, se calcula también su popularidad, para que esto afecte de manera positiva en la relevancia obtenida.

La fuerza con la cual es aplicado el beneficio dado por la popularidad está regido por un parámetro, configurable por el administrador, que permite identificar el porcentaje con el cual la popularidad de la página influye en la relevancia calculada para la misma. Con este porcentaje es posible regular de forma efectiva otra práctica desleal muy común, que consiste en agregar enlaces a páginas muy populares, con el fin de popularizar la página que los contiene.

9.3 Cálculo de relevancia

En esta sección se muestra en detalle como son seleccionadas las páginas que deben ser mostradas ante una búsqueda y como se realiza el cálculo de la relevancia para ranquear estas

páginas.

Este proceso es el que mantienen ocultos todos los buscadores, ya que un conocimiento del mismo haría posible que los creadores de páginas coloquen sus páginas en el tope del ranking. Además podría llevar a un desarrollador, a plagiar el motor de búsqueda.

El cálculo de la relevancia tiene como entrada una frase de búsqueda en Formato de Búsqueda Estandarizado y como salida una lista de páginas que cumplen con lo establecido en la frase, ordenadas por relevancia.

Podríamos dividir el cálculo de relevancia en dos subprocesos, uno general que se encarga de gestionar el cálculo completo y otro que gestiona el cálculo de cada subfrase especificada en la frase de búsqueda.

9.3.1 Cálculo completo

Los pasos que hacen a este proceso se desarrollan a continuación y se describen brevemente. Cada etapa será ejemplificada siguiendo el supuesto de la búsqueda: `-bandera cuadro +"cielo azul" +argentina`.

1. Separación en subfrases.

   Se separa el texto de la frase de búsqueda en todas las subfrases que lo componen y su operador correspondiente, ya sea "+", "-" o ninguno.

   En nuestro ejemplo: *-bandera*, *cuadro*, *+"cielo azul"* (como una única frase) y *+argentina*.

2. Búsqueda de cada subfrase por separado.

   Por cada subfrase separada en la etapa anterior, se dispara el segundo proceso, el proceso de "Búsqueda de subfrase" (ver Subsección 9.3.2).

   Si para alguno de los procesos disparados para una subfrase que contuviera el operador "+", de obligatoriedad, el cálculo de partes retornara una búsqueda vacía, finalizará automáticamente el proceso de búsqueda completo y retornará un conjunto de resultados finales vacío (no se retornarán páginas).

   En nuestro ejemplo, si la ejecución *BusquedaSubfrase(+"cielo azul")* retornara 0 resultados, el proceso finalizaría inmediatamente con la devolución de 0 resultados finales, ya que al no haber ubicado un dato obligatorio en ninguna de las páginas, no se requiere continuar con el resto de los pasos.

3. Eliminación de páginas que no deben aparecer.

   Se borran del conjunto de resultados aquellas páginas que fueron encontradas porque contenían una palabra que estaba marcada con el operador "-" (p.ej: si una página X contiene las palabras *bandera* y *cuadro*, será retirada de la lista de resultados porque no debe contener a la palabra *bandera*).

4. Gestión de búsquedas con datos obligatorios.

   Si hay al menos una subfrase de búsqueda que contiene un operador "+" entonces se procede con la siguiente gestión (p.ej: se sigue con este proceso porque *+"cielo azul"* contiene un operador "+").

   Se borran del conjunto de resultados parcial aquellas páginas que no contienen al menos una de todas las subfrases obligatorias, ya que si se solicita que tengan ***n*** palabras obligatorias, las ***n*** palabras deben aparecer en cada página resultante (p.ej: si la página

X contuviera la subfrase *cuadro* pero no la subfrase *+"cielo azul"*, sería eliminada en este paso).

Se obtiene un promedio de todas las relevancias calculadas para cada página a través de las distintas subfrases que contienen un operador "+". Es decir, que si en la búsqueda de *-bandera cuadro +"cielo azul" +argentina*, se obtiene una relevancia de 30 para la búsqueda de *+"cielo azul"* en la página X y una relevancia de 20 para *+argentina* en la misma página, entonces la relevancia de obligatoriedad de la página será 25.

Luego se obtiene la máxima relevancia de las calculadas para cada página a través de las distintas subfrases que no contienen operador (opción). Es decir, que si a la página X, en la búsqueda de *-bandera cuadro +"cielo azul" +argentina* se le otorga una relevancia de 10 para la búsqueda de *bandera* y una relevancia de 60 para *cuadro* entonces la relevancia de opción de la página X será 60.

Finalmente la relevancia final para cada página estará dada por el máximo entre la relevancia de obligatoriedad y la relevancia de opción. Para el ejemplo, la relevancia temporal para la página X será 60 (porque 60 es mayor que 25).

5. Penalizaciones.

   Se obtiene el factor de penalización de la tabla de parámetros. Se aplica el porcentaje de penalización calculado para la página a la relevancia de cada página, con el peso establecido en el factor de penalización.

   Además se agrega una penalización a la página, por cada palabra encontrada que esté en el grupo de palabras penalizadas (p.ej: si la página X contiene una penalización por spamming en la palabra *bandera* ya que está muy repetida, tendrá una penalización adicional porque *bandera* es una de las palabras buscadas).

6. Popularidad de enlaces.

   Se obtiene una proporción entre la cantidad de enlaces que apuntan a la página y la cantidad total de páginas almacenada en el catálogo, y se aplica este porcentaje como beneficio a la relevancia calculada en el punto anterior con el peso establecido en el factor de enlaces (de la tabla de parámetros).

La relevancia calculada en el último punto es considerada la relevancia final de la página y es la que se presenta al usuario en la lista de resultados.

9.3.2 Búsqueda de subfrase

Este proceso genera una búsqueda para cada palabra/subfrase que se le envíe como dato de entrada. A continuación se detallan los pasos siguiendo con el ejemplo descrito en la subsección anterior.

1. Separación en palabras.

   Se separa la subfrase de búsqueda en las palabras que la componen (p.ej: cuando se le pasa *"cielo azul"*, la divide en *cielo* y *azul*).

2. Conteo de coincidencias no exactas, por descriptor.

   Se cuenta que cantidad de veces se encuentra el texto buscado en cada descriptor. El texto no es buscado en forma exacta; no tiene en cuenta mayúsculas y minúsculas, ni verifica si es la palabra completa o una porción de la misma.

En nuestro ejemplo, si la subfrase de entrada es *cuadro*. Puede que se encuentre 2 veces en el metatag "description" (como *CUADRO* y *cuadros*), una vez en el título (como *cuadro*) y 1 vez como *recuadro* en el cuerpo.

3. Modificación de relevancia en base a conteos.

   Los conteos obtenidos en el paso anterior son multiplicados por el peso del descriptor correspondiente. Luego se suman y se multiplican por el factor de relevancia de descriptores (proveniente de la tabla de parámetros). Finalmente se le aplica un logaritmo al valor sumado 1, para que los resultados muy acertados estén próximos en el tope.

   Por ejemplo, si el peso del metatag "description" es 0,1 y el del cuerpo 0,8; tendríamos los valores 0,1 * 2 y 0,8 * 1. Su suma daría 1 y multiplicados por un hipotético factor de relevancia de descriptores de 90, nos daría un valor de 0,9 (ya que los factores siempre se dividen por 100). Al aplicar el logaritmo a 1 + 0,9 obtenemos un valor final de 0,28.

4. Conteo de ocurrencias exactas.

   Se incrementa la relevancia de cada página si es que se encuentra cada palabra buscada exactamente igual a como se encuentra en la página. Este incremento es un porcentaje de la cantidad de palabras exactas sobre el total de palabras de la página multiplicados por el peso que da el factor de cantidad de palabras (de la tabla de parámetros).

   En el ejemplo se incrementará la relevancia en este paso, ya que la palabra *cuadro*, se encuentra exactamente igual en el título de la página (no ocurre los mismo en los demás descriptores, ya que aparece como *CUADRO*, *cuadros* y *recuadro*).

Al finalizar este último paso, se tiene una relevancia de la subfrase en cada página. Esto servirá para que todas las relevancias de todas las páginas buscando cada subfrase, sea consolidada en el proceso de cálculo completo (ver Subsección 9.3.1).

9.4 Presentación de resultados

Cuando finaliza la búsqueda se retorna al Agente Compactador el conjunto de resultados obtenidos por el motor de búsqueda de BIPI. Estos resultados se encuentran ordenados por relevancia. Cada uno de los resultados contiene el título de la página, la descripción (o en su defecto las primeras 50 palabras), la relevancia calculada para dicha página y la URL de la misma.

La lista de resultados que devuelve el motor de BIPI es mezclada con las listas de resultados enviadas por cada uno de los buscadores que integran la "metabúsqueda". El conjunto total de resultados es ordenado por relevancia. Estos resultados son presentados a través de la pantalla de búsqueda (ver Sección 6.2).

En caso de que la búsqueda se hubiera realizado seleccionando únicamente el buscador de BIPI, el conjunto de resultados mostrado estaría conformado por la búsqueda realizada por el motor de búsqueda de BIPI y se podría observar el funcionamiento específico del buscador.

9.5 Diagramas de flujo

A continuación se describen las etapas que componen una búsqueda. En primera instancia se especifica un diagrama del funcionamiento del metabuscador en la resolución de una búsqueda. El segundo diagrama corresponde al comportamiento del motor de búsqueda de BIPI, como ampliación a la búsqueda global definida en el primer diagrama.

9.5.1 Búsqueda en el metabuscador

El flujo de una búsqueda, desde el comienzo hasta su resolución, se muestra en la Figura 9.1 y se detalla en los siguientes puntos.

En este flujo existirán comunicaciones entre los agentes descritos en las secciones del Capítulo 5, realizadas utilizando el lenguaje KQML (ver Subsección 3.6). La comunicación entre los agentes locales y remotos se hace a través del middleware (ver Subsección 5.4.4) donde los mensajes KQML son enviados por sockets (implementados con el paquete java.net del lenguaje Java descrito en la Sección A.3).

- **(1)** El usuario especifica en la pantalla de búsqueda de BIPI una frase a ser buscada y los buscadores a ser utilizados; estos parámetros son pasados al Sub-entorno de Búsqueda.
- **(2)** Este sub-entorno pasa los parámetros al Agente Coordinador que se ocupa de dirigir la búsqueda a partir de los datos suministrados.
- **(3)** El Agente Coordinador verifica cuales son los buscadores involucrados en la búsqueda y pasa la frase de búsqueda a los Agentes Específicos de Búsqueda de cada uno de los buscadores involucrados.
- **(4)** Cada Agente Específico de Búsqueda recupera las páginas de resultados y las pasa a su Agente Filtrador.
- **(5)** Cada Agente Filtrador identifica los resultados de las páginas de resultados y los envía al Agente Compactador.
- **(6)** El Agente Compactador toma los resultados de todos los buscadores involucrados y se ocupa de descartar resultados repetidos y de ordenar los mismos por relevancia. Envía el conjunto mínimo de resultados ordenados al Agente de Búsqueda Continua.
- **(7)** El Agente de Búsqueda Continua pasa el mismo conjunto que recibe del Agente Compactador al Agente Coordinador.
- **(7 bis)** El Agente de Búsqueda Continua controla con la Base de Datos si los resultados obtenidos ya se encuentran indexados; en caso de no estar indexados son almacenados para su posterior indexación ("aprendizaje").
- **(8)** El Agente Coordinador envía el conjunto de resultados al Sub-entorno de Búsqueda.
- **(9)** El Sub-entorno de Búsqueda presenta los resultados en la página correspondiente a la visualización de resultados (ver Sección 6.2).

9.5.2 Búsqueda en el motor de BIPI

A continuación se realiza una ampliación del modelo de flujo anteriormente descrito. En el mismo se muestra el funcionamiento del Grupo de Búsqueda perteneciente a BIPI (ver Figura 9.2).

- **(a)** El Agente Coordinador pasa la frase de búsqueda al Agente Específico de Búsqueda de BIPI.
- **(b)** El Agente Específico de Búsqueda llama a la página de consulta interna (query.jsp, que utiliza la tecnología JSP -ver Apéndice B- ya que debe generarse dinámicamente a partir de los resultados provenientes de la Base de Datos ante la petición de búsqueda) de BIPI pasándole la frase como parámetro de la generación de la página.

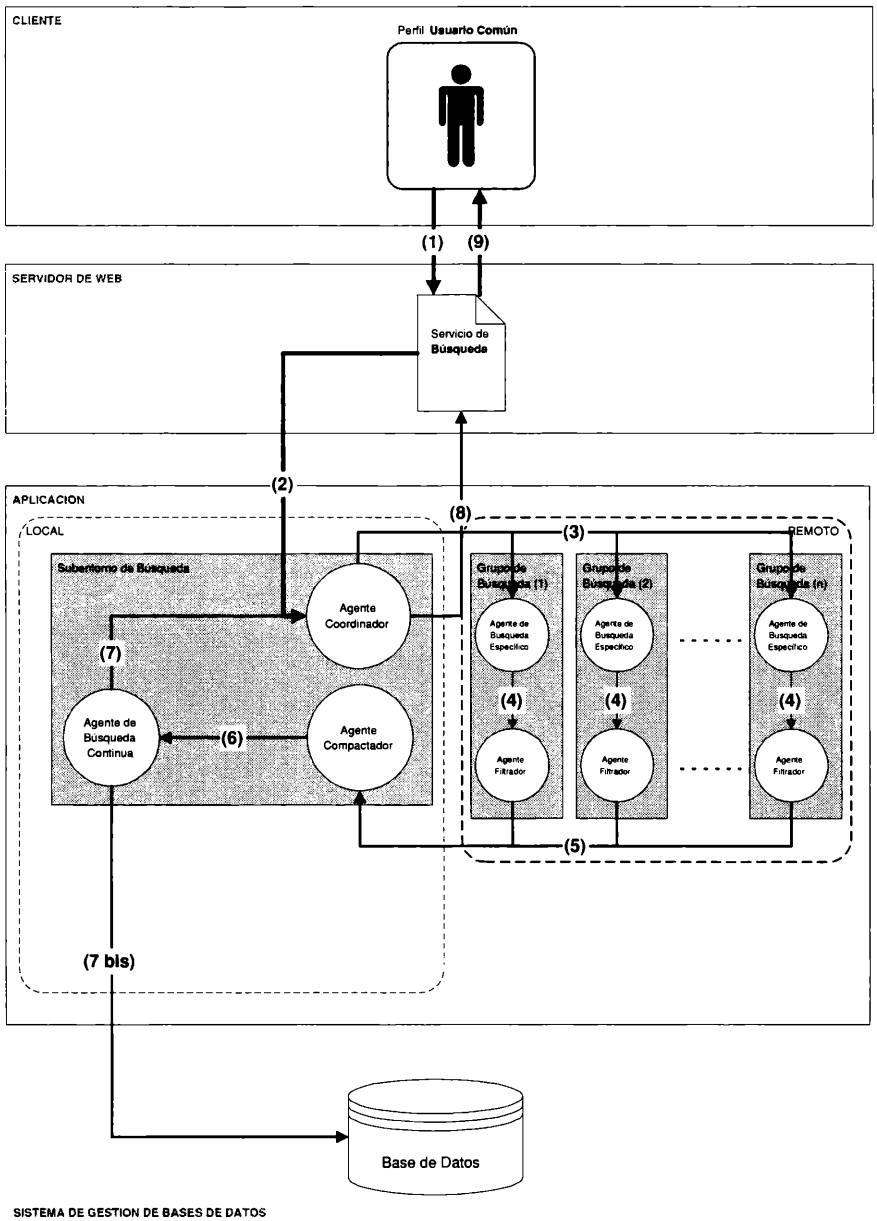

Figura 9.1: *Diagrama de flujo de una búsqueda.*

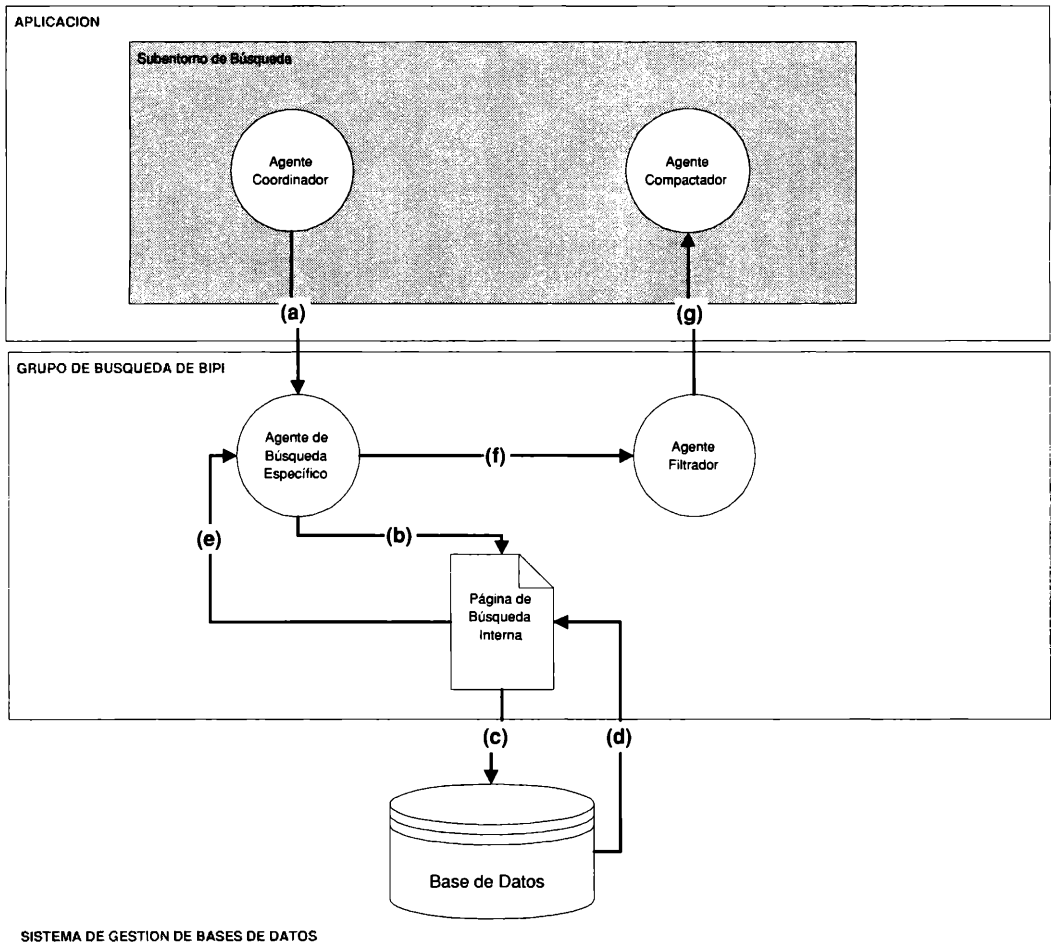

Figura 9.2: *Diagrama de flujo de una búsqueda en el motor de búsqueda.*

- **(c)** La página genera una búsqueda de la frase sobre la Base de Datos para obtener su contenido dinámico (los resultados).
- **(d)** Se genera un conjunto de resultados a partir de la búsqueda de la frase en la Base de Datos y se devuelve para conformar la página de consulta interna.
- **(e)** La página de resultados conformada es tomada por el Agente Específico de Búsqueda.
- **(f)** La página es pasada al Agente Filtrador para que este disgregue los resultados contenidos en la página.
- **(g)** Los resultados son enviados al Agente Compactador para que sean compactados con los demás resultados enviados por los diferentes buscadores involucrados en la búsqueda.

Parte IV

Notas finales

Capítulo 10

Perspectivas

10.1 Expansión del metabuscador

El sistema ha quedado definido de tal forma que es fácilmente expandible en lo que respecta a agregar nuevos buscadores que aporten resultados a una búsqueda. De esta forma el metabuscador puede crecer sencillamente hasta incluir todos los buscadores conocidos del mercado. Hay que tener en cuenta, en este punto, de no caer en la problemática de incluir "malos" buscadores que puedan llegar a "ensuciar" el conjunto de resultados. Por esto es que para la versión inicial del mismo se ha optado por incluir solamente tres buscadores, que se sabe retornan resultados relevantes.

10.1.1 Pasos técnicos de la expansión

Por ejemplo, si deseamos agregar el motor de búsqueda de `HotBot` al metabuscador, debemos seguir los siguientes tres simples pasos:

- **Creación del Grupo de Búsqueda para HotBot**: es necesario definir un Agente Específico de Búsqueda que acceda a la página de consulta de HotBot y un Agente Filtrador que pueda tratar esta página de resultados. Esto se hace en Java agregando una subclase a *SearchAgent* y otra a *FilterAgent*.

- **Indicar al Sub-entorno de Búsqueda que puede usar HotBot**: esto consiste en la incorporación de un método en la clase *BipiSearch* que instancie el Agente Específico de Búsqueda y el Agente Filtrador definidos en el paso anterior.

- **Modificar la interfaz**: consiste en la inclusión de un nuevo check que permita indicarle al metabuscador que se desea utilizar HotBot en la búsqueda. Además, se debe modificar el código de la página de búsqueda JSP, para que haga referencia al método definido en el Sub-entorno de Búsqueda.

10.2 Extensión de funcionalidad

El modelo de búsqueda que al día de hoy se encuentra definido, puede sufrir perfeccionamiento de las técnicas de spamming aplicadas y puede adoptar técnicas que surjan debido a la aparición de nuevas prácticas de spamming. La perfección del método de detección de spamming puede intentarse buscando, entre otras cosas, texto "oculto" (en el mismo color que el fondo o en letra muy pequeña) en las páginas.

Otra funcionalidad adicional que puede ser incluida, es permitir la realización de búsquedas definidas por operadores lógicos, las llamadas búsquedas avanzadas en otros buscadores. Actualmente en el buscador pueden obtenerse los mismos resultados que si se utilizaran los operadores lógicos, realizando la búsqueda con los operadores provistos y separándola en múltiples búsquedas cuando se requiere incluir el operador OR.

Nuestro trabajo futuro, sobre el modelo actualmente definido, tiende a la inclusión de las funcionalidades expresadas anteriormente, para que se transforme en una aplicación con mucho potencial y que pueda competir de igual a igual con cualquier metabuscador hoy en día desarrollado. Esperamos mejorar aún más la facilidad de expansión del metabuscador y ampliar las posibilidades de búsqueda (del metabuscador y del motor), incluyendo búsquedas por sitios, lenguajes (induciendo el lenguaje de las páginas indexadas), metatags, enlaces, etc. y diversificando la capacidad de indexación hacia imágenes y otros archivos multimediales.

Capítulo 11

Conclusiones

Llegamos a este capítulo con la firme idea de concentrar una serie de párrafos que describan adecuadamente el desarrollo de la investigación. Volcamos aquí las conclusiones que se determinan a lo largo de meses de investigación y desarrollo. Como añadido a este capitulo intentaremos respondernos a aquellas cuestiones que se pregonaban por aquellos tiempos que rodearon a la definición de la propuesta del trabajo de grado, sin remontarnos específicamente a cada una de ellas.

Hemos investigado y aprendido sobre buscadores, agentes, tecnología JSP, Java y muchas cosas más. Algunos temas hicieron a los conceptos preliminares del desarrollo y otros fueron los que nos brindaron la posibilidad de aplicarlos.

La tecnología que se ha usado en el desarrollo es de punta, cada vez más utilizada en aplicaciones web comerciales. Ha sido un desafío la inclusión de la tecnología JSP y la utilización de la plataforma Java 2, ya que nos condujo, en sus comienzos, a una cuesta empinada para su aprendizaje, pero nos brindó una facilidad creciente hasta el cierre de este trabajo.

Las herramientas utilizadas nos simplificaron la aplicación del concepto de agente móvil que creemos fue el adecuado para la separación y encapsulamiento de la funcionalidad. Las características de los agentes, tales como autonomía, cooperación, comunicación (mediante el lenguaje estándar de los agentes: KQML), organización en sistemas multiagente y movilidad, permitieron construir una herramienta con mínimo acoplamiento (con conexiones simples), con alta cohesion interna, fácilmente escalable y multiplataforma.

BIPI es un metabuscador que satisface todas las características de cualquier otro, permite que un usuario establezca una búsqueda y él se encarga de dividirla en distintos buscadores, controlarla y concentrarla nuevamente en él para otorgar los resultados al usuario. Hace uso del motor de búsqueda propietario, que es un buscador con muchas características de un motor de búsqueda, pero que podríamos definir como un niño dando sus primeros pasos, ya que puede expandirse en diversas direcciones como vimos en el último capítulo.

Finalmente destacamos la separación concreta entre un metabuscador y un motor de búsqueda independientes, pero al mismo tiempo con un leve acoplamiento. Estos dos elementos proveen la utilidad deseada al inicio de este trabajo: la posibilidad de realizar búsquedas hasta el nivel de especificación necesario para llegar a la información deseada, *con la seguridad de que son los resultados más relevantes obtenibles entre todos los resultados de los motores involucrados.*

Parte V

Apéndices

Apéndice A

Java

A.1 Introducción

Java surgió en 1991 cuando un grupo de ingenieros de *Sun Microsystems* trataron de diseñar un nuevo lenguaje de programación destinado a electrodomésticos. La reducida potencia de cálculo y memoria de los electrodomésticos llevó a desarrollar un lenguaje sencillo capaz de generar código de tamaño muy reducido.

Debido a la existencia de distintos tipos de CPUs y a los continuos cambios, era importante conseguir una herramienta independiente del tipo de CPU utilizada. Así es que se desarrolló un código "neutro" que no dependa del tipo de electrodoméstico, el cual se ejecuta sobre una "máquina hipotética o virtual" denominada Java Virtual Machine (JVM). Es la JVM quien interpreta el código neutro convirtiéndolo a código particular de la CPU utilizada. Esto permite lo que se ha convertido en el principal lema del lenguaje: "Write Once, Run Everywhere".

A pesar de los esfuerzos realizados por sus creadores, ninguna empresa de electrodomésticos se interesó por el nuevo lenguaje.

Java, como lenguaje de programación para computadores, se introdujo a finales de 1995. La clave fue la incorporación de un intérprete Java en el programa Netscape Navigator, versión 2.0, produciendo una verdadera revolución en Internet. Java 1.1 apareció a principios de 1997, mejorando sustancialmente la primera versión del lenguaje.

Al programar en Java no se parte de cero. Cualquier aplicación que se desarrolle "cuelga" (o se apoya, según como se quiera ver) en un gran número de clases preexistentes. Algunas de ellas las ha podido hacer el propio usuario, otras pueden ser comerciales, pero siempre hay un número muy importante de clases que forman parte del propio lenguaje (el API o Application Programming Interface de Java). Java incorpora muchos aspectos que en cualquier otro lenguaje son extensiones propiedad de empresas de software o fabricantes de ordenadores (threads, ejecución remota, componentes, seguridad, acceso a bases de datos, etc.). Esto es consecuencia de haber sido diseñado más recientemente y por un único equipo.

El principal objetivo del lenguaje Java es llegar a ser el "nexo universal" que conecte a los usuarios con la información, esté ésta situada en el ordenador local, en un servidor de Web, en una base de datos o en cualquier otro lugar. Java es un lenguaje muy completo (se está convirtiendo en un macro-lenguaje: Java 1.0 tenía 12 packages; Java 1.1 tenía 23 y Java 1.2 tiene 59). En cierta forma casi todo depende de casi todo.

La compañía Sun describe el lenguaje Java como "simple, orientado a objetos, distribuido, interpretado, robusto, seguro, de arquitectura neutra, portable, de altas prestaciones, multitarea y dinámico". Además de una serie de halagos por parte de Sun hacia su propia criatura, el hecho es que todo ello describe bastante bien el lenguaje Java.

A.2 Aplicaciones Java

Los programas desarrollados en Java presentan diversas ventajas frente a los desarrollados en otros lenguajes como C/C++. La ejecución de programas en Java tiene muchas posibilidades: ejecución como *aplicación independiente* (Stand-alone Application), ejecución como *applet*, ejecución como *servlet*, etc.. Un applet es una aplicación especial que se ejecuta dentro de un navegador o browser (por ejemplo Netscape Navigator o Internet Explorer) al cargar una página HTML desde un servidor Web. El applet se descarga desde el servidor y no requiere instalación en el ordenador donde se encuentra el browser. Un servlet es una aplicación sin interface gráfica que se ejecuta en un servidor de Internet. La ejecución como aplicación independiente es análoga a los programas desarrollados con otros lenguajes.

Además de incorporar la ejecución como applet, Java permite fácilmente el desarrollo tanto de arquitecturas cliente-servidor como de aplicaciones distribuidas, consistentes en crear aplicaciones capaces de conectarse a otros ordenadores y ejecutar tareas en varios ordenadores simultáneamente, repartiendo por lo tanto el trabajo. Aunque también otros lenguajes de programación permiten crear aplicaciones de este tipo, Java incorpora en su propio API estas funcionalidades.

A.3 Java en la red

A diferencia de otros lenguajes de programación, Java presenta de una forma estándar para todas las plataformas y sistemas operativos, un conjunto de clases que permiten la comunicación entre aplicaciones que se ejecutan en distintos ordenadores. El *package java.net* del API de Java incluye las clases necesarias para establecer conexiones, crear servidores, enviar y recibir datos, y para el resto de operaciones utilizadas en las comunicaciones a través de redes de ordenadores. Existen además otros APIs independientes preparados especialmente para realizar unas determinadas tareas, como son Servlets, RMI y Java IDL. Muchos de estos APIs utilizan internamente las clases presentes en java.net.

A.4 Java en el servidor: Servlets

Los *Servlets* son módulos que permiten sustituir o utilizar el lenguaje Java en lugar de los programas CGI escritos en otros lenguajes como C/C++ o Perl. Los programas CGI son aplicaciones que se ejecutan en un servidor Web en respuesta a una acción de un browser remoto (petición de una página HTML, envío de los datos de un formulario, etc.). Permiten generar páginas HTML dinámicas, esto es, páginas HTML cuyo contenido puede variar y que por lo tanto no pueden almacenarse en un fichero en el servidor. Los Servlets no tienen entorno gráfico ya que se ejecutan en el servidor. Reciben unos datos y su salida o respuesta son principalmente ficheros de texto HTML. Los servlets son desarrollados utilizando el API Java Servlet, que es una extensión de Java que hasta la fecha no forma parte de ninguno de los JDK. Es necesario instalar el software específico de Java Servlet.

Los Servlets son generados como resultado de la traducción/compilación de una página JSP.

A.5 Acceso a bases de datos (JDBC)

JDBC (Java DataBase Connectivity) es el estándar de Java para conectarse con bases de datos. Se estima que aproximadamente la mitad del software que se crea en la actualidad incorpora operaciones de lectura y escritura con bases de datos. JDBC está diseñado para

ser independiente de la plataforma e incluso de la base de datos sobra la que se desee actuar. Para conseguir esta independencia, JDBC ofrece un sistema estándar de interconexión con las bases de datos, muy similar al SQL (Structured Query Language). Los distintos vendedores de bases de datos crean los elementos necesarios que actúan como puente entre JDBC y la propia base de datos. La versión JDBC 1.0 forma parte del JDK 1.1. Después de distintas revisiones, actualmente ha aparecido la versión JDBC 2.0, incluida en el JDK 1.2.

La última versión de JDBC se utiliza en la interconexión entre ciertos componentes de BIPI (Agente de Búsqueda Continua, Agente Indexador, Agente de Incorporación de URL, etc.) y su Base de Datos SQL Server.

A.6 Multitarea: threads

Los procesadores y los Sistemas Operativos modernos permiten la *multitarea*, es decir, la realización simultánea de dos o más actividades (al menos aparentemente). En la realidad, un ordenador con una sola CPU no puede realizar dos actividades a la vez. Sin embargo los Sistemas Operativos actuales son capaces de ejecutar varios programas "simultáneamente" aunque sólo se disponga de una CPU: reparten el tiempo entre dos (o más) actividades, o bien utilizan los tiempos muertos de una actividad (por ejemplo, operaciones de lectura de datos desde el teclado) para trabajar en la otra. En ordenadores con dos o más procesadores la multitarea es real, ya que cada procesador puede ejecutar un ***hilo*** o ***thread*** diferente.

Un proceso es un programa ejecutándose de forma independiente y con un espacio propio de memoria. Un Sistema Operativo multitarea es capaz de ejecutar más de un proceso simultáneamente. Un thread o hilo es un flujo secuencial simple dentro de un proceso. Un único proceso puede tener varios hilos ejecutándose. Por ejemplo el programa Netscape sería un proceso, mientras que cada una de las ventanas que se pueden tener abiertas simultáneamente trayendo páginas HTML estaría formada por al menos un hilo.

Un sistema multitarea da realmente la impresión de estar haciendo varias cosas a la vez y eso es una gran ventaja para el usuario. Sin el uso de threads hay tareas que son prácticamente imposibles de ejecutar, particularmente las que tienen tiempos de espera importantes entre etapas.

Los threads o hilos de ejecución permiten organizar los recursos del ordenador de forma que pueda haber varios programas actuando en paralelo. Un hilo de ejecución puede realizar cualquier tarea que pueda realizar un programa normal y corriente. Bastará con indicar lo que tiene que hacer en el método *run()*, que es el que define la actividad principal de las threads.

Los threads pueden ser *daemon* o *no daemon*. Son daemon aquellos hilos que realizan en background (en un segundo plano) servicios generales, esto es, tareas que no forman parte de la esencia del programa y que se están ejecutando mientras no finalice la aplicación. Un thread daemon podría ser por ejemplo aquél que está comprobando permanentemente si el usuario pulsa un botón. Un programa de Java finaliza cuando sólo quedan corriendo threads de tipo daemon. Por defecto, y si no se indica lo contrario, los threads son del tipo no daemon.

Apéndice B

JSP

B.1 La tecnología JSP

La tecnología de las *Java Server Pages* es la tecnología Java en la plataforma J2EE para la construcción de aplicaciones conteniendo contenido web dinámico tal como HTML, DHTML, XHTML y XML. Esta tecnología permite crear páginas web que crean su contenido dinámico fácilmente pero con el máximo poder y flexibilidad.

B.1.1 Características

La tecnología JSP provee una descripción textual para la creación de la respuesta (response) para cada petición (request). La tecnología se basa en los siguientes conceptos:

- Datos de plantilla

  Se permite la gestión de manera natural de los datos fijos.

- Adición de datos dinámicos

  La tecnología JSP permite agregar datos dinámicos a los datos de plantilla de forma simple pero poderosa.

- Encapsulamiento de funcionalidad

  La tecnología JSP provee dos modos relacionados para el encapsulamiento de funcionalidad: la arquitectura estándar de JavaBeans y el mecanismo de librería de descriptores.

- Buen soporte de herramientas

  Una buen conjunto de herramientas de soporte lleva a mejorar significativamente la productividad. La tecnología JSP tiene ciertas características que permiten que se desarrollen herramientas para su gestión.

B.1.2 Beneficios de la tecnología JSP

La tecnología JSP ofrece un número de beneficios.

- Escribir una vez, ejecutar en cualquier lugar

  Es una plataforma independiente, tanto en sus páginas web dinámicas, sus servidores web y los componentes de servidor subyacentes. Se pueden crear páginas JSP en cualquier plataforma, ejecutarlas en cualquier servidor web o servidor de aplicaciones con facilidades web, y accederlas desde cualquier navegador. También pueden construirse componentes de servidor en cualquier plataforma y ejecutarlos en cualquier servidor.

- Herramientas de soporte de alta calidad

  Las características *Write Once, Run Anywhere* (Escribir una vez, ejecutar en cualquier lugar) de JSP hace que el usuario pueda elegir la mejor herramienta. Adicionalmente, un objetivo explícito del diseño de páginas JSP, es que permite la creación de herramientas portables de alta calidad.

- Separación de roles

  JSP soporta la separación de roles: los *desarrolladores* escriben componentes que interactúan con los objetos del lado del servidor; los *autores* colocan datos estáticos y contenido dinámico juntos para crear las presentaciones que mejor se adecúen a las audiencias. Cada uno de estos roles marca diferentes tipos de habilidades y, aunque estas habilidades pueden estar presentes en el mismo individuo, generalmente no lo estarán. Un subconjunto de la comunidad de desarrolladores puede enfocarse en la creación de componentes reusables que serán usados por los autores.

- Reusabilidad de componentes y librerías de descriptores

  La tecnología JSP enfatiza el uso de componentes reusables tales como: componentes JavaBeans, componentes Enterprise JavaBeans y librerías de descriptores. Estos componentes pueden ser usados en herramientas interactivas para desarrollo de componentes y composición de páginas. Esto ahorra tiempo considerable de desarrollo mientras que da el poder de las plataformas cruzadas y la flexibilidad del lenguaje Java y otros lenguajes de scripting.

- Separación de contenido estático y dinámico

  Permite la separación de contenido estático de la plantilla estática y el contenido dinámico. Esto simplifica la creación de contenidos. La separación es soportada por beans específicamente diseñados para la interacción con objetos del lado del servidor y, especialmente, por el mecanismo de extensión de descriptores.

- Soporte de scripting y acciones

  La tecnología JSP da cabida a mecanismos de scripting tanto como a acciones. Las acciones permiten el *encapsulamiento* de funcionalidad útil dentro de un formulario adecuado que puede ser manipulado por herramientas; los scripts proveen un mecanismo de concentración de toda esta funcionalidad en una página.

- Capa de acceso web para arquitectura(s) de aplicaciones empresariales N-tier

  La tecnología JSP es una parte integral de la Edición Empresarial de la Plataforma Java 2 (J2EE), que lleva la tecnología Java a la computación empresarial. Ahora se pueden desarrollar poderosas aplicaciones servidor (de capas medias), usando un sitio web, que usa JSP como interfaz para componentes Enterprise JavaBeans en un entorno compatible J2EE.

B.2 Conceptos básicos

Esta sección introduce los conceptos básicos necesarios para interpretar los elementos que componen la tecnología JSP.

B.2.1 ¿Qué es una página JSP?

Una página JSP es un documento de texto que describe como debe procesarse un *request* para crear una *response*. La descripción entremezcla datos de plantilla con algunas acciones dinámicas con la plataforma Java 2.

Las características de la tecnología JSP da sustento a un número de diferentes paradigmas para la creación de contenido dinámico.

B.2.2 Aplicaciones web

El concepto de aplicación web es heredado de la especificación de servlets. Una aplicación web puede estar compuesta por:

- Java Runtime Environment(s) ejecutándose en el servidor (obligatorio)
- Páginas JSP, que gestionan las peticiones y generan contenido dinámico
- Servlet(s), que gestionan las peticiones y generan contenido dinámico
- JavaBeans (del lado del servidor) que encapsulan comportamiento y estado
- Páginas estáticas HTML, DHTML, XHTML, XML y similares
- Applets de Java, JavaBeans y otras archivos .class de Java (del lado del cliente)
- Java Runtime Environment(s) ejecutándose en los clientes.

La especificación de páginas JSP hereda de la especificación de servlets los conceptos de: aplicaciones, contextos de servlets, sesiones, requests y responses.

B.2.3 Componentes y contenedores

Las páginas JSP y las clases de Servlets son denominados conjuntamente como *componentes web*. Las páginas JSP son entregadas a un *contenedor* que provee los servicios necesarios para su gestión.

B.2.4 Traducción y pasos de ejecución

Las páginas JSP son componentes de texto. Pasan a través de dos fases: una fase de *traducción* y una fase de *petición*. La traducción es realizada una vez por página. La fase de petición es realizada una vez por petición.

El resultado de la fase de traducción es la creación de la clase Servlet (ver Sección A.4): la *clase de implementación de la página JSP* que será instanciada ante una solicitud. El *objeto de implementación de la página JSP* gestiona las solicitudes y crea respuestas.

Es posible realizar la fase de traducción con anterioridad (lo que comúnmente se denomina compilación de las páginas JSP en Servlets) y colocarlas en una aplicación web, transparentemente, una clase Servlet que se comportará como la representación textual de la página JSP.

La fase de traducción puede se hecha por el contenedor JSP en *tiempo de desarrollo* o *sobre-demanda* cuando una petición alcanza una página JSP que aún no ha sido traducida.

B.2.5 Rasgos de las JSP

Las características principales de las páginas JSP son:

- Directivas estándar
- Acciones estándar
- Elementos de scripting
- Mecanismo de extensión de descriptores
- Contenido de plantilla

Apéndice C

Documentación técnica de la aplicación

C.1 All Packages

- package ar.com.bipi.agents
- package ar.com.bipi.connection
- package ar.com.bipi.consoles
- package ar.com.bipi.prueba
- package ar.com.bipi.servers
- package ar.com.bipi.system
- package ar.com.bipi.types

C.2 Package ar.com.bipi.agents

- AddURLAgent
- CompactAgent
- ContinuesSearchAgent
- CoordinatorAgent
- FilterAgent
- FilterAltavistaAgent
- FilterBipiAgent
- FilterExciteAgent
- FilterWebCrawlerAgent
- MaintenanceAgent
- SearchAgent

- SearchAltavistaAgent
- SearchBipiAgent
- SearchExciteAgent
- SearchWebCrawlerAgent
- SpiderAgent

Class ar.com.bipi.agents.AddURLAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.AddURLAgent

public class AddURLAgent extends java.lang.Thread
```

La clase AddUrlAgent permite realizar el alta de una nueva página en la Base de Datos. Las páginas agregadas por esta clase serán luego analizadas por el Spider de Bipi. En el alta de una página se podrán incorporar los siguientes datos referentes a la página: URL, descripción, e-mail, keywords y si debe indexarse el sitio completo.

```
Variables:

    private java.lang.String url
    private java.lang.String eMail
    private java.lang.String description
    private java.lang.String keywords
    private int indexSite
    private boolean state
    private boolean wait

Constructor:
    public AddURLAgent()
          Constructor por defecto.

Métodos públicos:
    public int getIndexSite()
          Recupera la marca de Indexar Sitio.
    public boolean getState()
          Sincroniza el acceso al estado de la operación de Alta de Página.
    public void run()
          Genera el alta de la página sobre la Base de Datos.
    public void setDescription(java.lang.String newDescription)
          Setea la Descripción de la página que se desea agregar.
    public void setEMail(java.lang.String newEMail)
          Setea el E-mail de la página que se desea agregar.
    public void setIndexSite(int newIndexSite)
          Setea si la página debe indexar todo el sitio.
    public void setKeywords(java.lang.String newKeywords)
          Setea las Keywords de la página que se desea agregar.
    private void setState(boolean newState)
          Sincroniza el acceso al estado de la operación de Alta de Página.
    public void setUrl(java.lang.String newUrl)
          Setea la URL de la página que se desea agregar.
```

Class ar.com.bipi.agents.CompactAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.CompactAgent

public class CompactAgent extends java.lang.Thread
```

La clase CompactAgent encapsula el funcionamiento del Agente Compactador que recibe por sus canales de entrada los resultados de búsqueda y provee por su canal de salida un conjunto compactado de dichos resultados.

```
Variables:
    private java.util.Vector inChannels
    private ExchangeChannel outChannel

Constructor:
    public CompactAgent()
          Constructor por defecto.

Métodos públicos:
    public void addInChannel(ExchangeChannel inChannel)
          Agrega un nuevo canal de entrada para recibir resultados de búsqueda.
    public void run()
          Recupera los resultados de búsqueda de los distintos canales de entrada, elimina
          los resultados repetidos y devuelve el conjunto de resultados ordenados
          por relevancia.
    public void setOutChannel(ExchangeChannel outChannel)
          Setea el canal de salida por donde devolverá el conjunto mínimo de resultados.
```

Class ar.com.bipi.agents.ContinuesSearchAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.agents.ContinuesSearchAgent

public class ContinuesSearchAgent extends java.lang.Thread
implements java.lang.Runnable
```

La clase ContinuesSearchAgent encapsula el funcionamiento del Agente de Búsqueda Continua que recibe por su canal de entrada un conjunto de resultados y los devuelve por su canal de salida. Almacenando en la Base de Datos aquellos resultados que no existan en la misma.

```
Variables:
    private ExchangeChannel inChannel
    private ExchangeChannel outChannel

Constructor:
    public ContinuesSearchAgent(ExchangeChannel inChannel, ExchangeChannel outChannel)
          Constructor por defecto.

Métodos públicos:
    public void run()
          Recibe por su canal de entrada un conjunto de resultados y los devuelve por
          su canal de salida. Almacena en la Base de Datos las URLs, para su posterior
          indexación, que no se encuentren en la misma.
```

Class ar.com.bipi.agents.CoordinatorAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.agents.CoordinatorAgent

public class CoordinatorAgent extends java.lang.Thread implements
java.lang.Runnable
```

La clase CoordinatorAgent encapsula el funcionamiento del Agente Coordinador.

```
Variables:
    private java.lang.String frase
    private ExchangeChannel inChannel
    private ExchangeChannel returnChannel
```

```
    private java.util.Vector outChannels
    private AddressData address

Constructor:
    public CoordinatorAgent(java.lang.String frase,AddressData address)
          Constructor por defecto que setea la frase de búsqueda y la dirección IP
          del host donde corre.

Métodos públicos:
    public void addOutChannel(ExchangeChannel outChannel)
          Agrega un canal de salida para el envío de la frase de búsqueda a un nuevo
          buscador.
    public AddressData getAddress()
          Retorna la dirección IP del host donde corre.
    public void run()
          Envía por todos los canales de salida: la dirección IP del host en donde
          esta corriendo el CompactAgent correspondiente y la frase de búsqueda.
          Espera los resultados de búsqueda por su canal de entrada y los envía por
          su canal de salida especial hacia la interfaz.
    public void setInChannel(ExchangeChannel inChannel)
          Setea el canal de entrada.
    public void setReturnChannel(ExchangeChannel returnChannel)
          Setea el canal de salida especial.
```

Class ar.com.bipi.agents.FilterAgent

```
     java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.agents.FilterAgent

public abstract class FilterAgent extends java.lang.Thread
implements java.lang.Runnable
```

Clase abstracta que encapsula el funcionamiento del Agente Filtrador.

```
Variables:
    private ExchangeChannel inChannel
    private ExchangeChannel outChannel
    protected int position
    protected int slice

Constructor:
    public FilterAgent()
          Constructor por defecto.
    public FilterAgent(ExchangeChannel inChannel, ExchangeChannel outChannel)
          Constructor que setea el canal de entrada y el canal de salida.

Métodos públicos:
    protected abstract SearchLink[] filter(java.lang.String page)
          Método abstracto que debe implementarse en las subclases y que debe implementar
          el parseo de la página devuelta por el buscador correspondiente.
    public ExchangeChannel getInChannel()
          Retorna el canal de entrada.
    public ExchangeChannel getOutChannel()
          Retorna el canal de salida.
    public int getSlice()
          Retorna el slice.
    public void run()
          Recupera la página a procesar del canal de entrada, le aplica el filtro
          definido por el método filter y retorna los resultados devueltos por el mismo.
    public void setInChannel(ExchangeChannel newInChannel)
          Setea el canal de entrada.
    public void setOutChannel(ExchangeChannel newOutChannel)
          Setea el canal de salida.
    public void setSlice(int newSlice)
          Setea el slice.
```

Class ar.com.bipi.agents.FilterAltavistaAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----ar.com.bipi.agents.FilterAgent
                   |
                   +----ar.com.bipi.agents.FilterAltavistaAgent
```

```
public final class FilterAltavistaAgent extends FilterAgent
```

Encapsula el método de filtrado y desglose de las páginas de resultados devueltas por Altavista.

```
Constructor:
    public FilterAltavistaAgent()
          Constructor por defecto.
    public FilterAltavistaAgent(ExchangeChannel inChannel,ExchangeChannel outChannel)
          Constructor que setea el canal de entrada y el canal de salida.

Métodos públicos:
    protected SearchLink[] filter(java.lang.String doc)
          Implementa el parseo y desglose de la página pasada como parámetro
          correspondiente al buscador Altavista.
```

Class ar.com.bipi.agents.FilterBipiAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----ar.com.bipi.agents.FilterAgent
                   |
                   +----ar.com.bipi.agents.FilterBipiAgent
```

```
public final class FilterBipiAgent extends FilterAgent
```

Encapsula el método de filtrado y desglose de las páginas de resultados devueltas por BIPI.

```
Constructor:
    public FilterBipiAgent()
          Constructor por defecto.
Métodos públicos:
    protected SearchLink[] filter(java.lang.String doc)
          Implementa el parseo y desglose de la página pasada como parámetro
          correspondiente al buscador BIPI.
```

Class ar.com.bipi.agents.FilterExciteAgent

```
java.lang.Object
   |
   +----java.lang.Thread
           |
           +----ar.com.bipi.agents.FilterAgent
                   |
                   +----ar.com.bipi.agents.FilterExciteAgent
```

```
public final class FilterExciteAgent extends FilterAgent
```

Encapsula el método de filtrado y desglose de las páginas de resultados devueltas por Excite.

```
Constructor:
    public FilterExciteAgent()
          Constructor por defecto.
    public FilterExciteAgent(ExchangeChannel inChannel, ExchangeChannel outChannel)
          Constructor que setea el canal de entrada y el canal de salida.

Métodos públicos:
    protected SearchLink[] filter(java.lang.String doc)
          Implementa el parseo y desglose de la página pasada como parámetro
          correspondiente al buscador Excite.
```

Class ar.com.bipi.agents.FilterWebCrawlerAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.FilterAgent
                    |
                    +----ar.com.bipi.agents.FilterWebCrawlerAgent

public final class FilterWebCrawlerAgent extends FilterAgent
```

Encapsula el método de filtrado y desglose de las páginas de resultados devueltas por WebCrawler.

```
Constructor:
    public FilterWebCrawlerAgent()
          Constructor por defecto.
    public FilterWebCrawlerAgent(ExchangeChannel inChannel, ExchangeChannel outChannel)
          Constructor que setea el canal de entrada y el canal de salida.

Métodos públicos:
    protected SearchLink[] filter(java.lang.String doc)
          Implementa el parseo y desglose de la página pasada como parámetro
          correspondiente al buscador WebCrawler.
```

Class ar.com.bipi.agents.MaintenanceAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.MaintenanceAgent

public class MaintenanceAgent extends java.lang.Thread
```

La clase MaintenanceAgent encapsula el funcionamiento del Agente de Mantenimiento.

```
Constructor:
    public MaintenanceAgent()
          Constructor por defecto.

Métodos públicos:
    public void run()
          Dispara el proceso de mantenimiento de la Base de Datos con el cual se revisa
          la caducidad de las páginas almacenadas para permitir que sean reindexadas.
          Gestiona el CEP (\CEP).
```

Class ar.com.bipi.agents.SearchAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.SearchAgent

public abstract class SearchAgent extends java.lang.Thread
```

La clase abstracta SearchAgent encapsula el funcionamiento del Agente de Búsqueda Específico que recibe la frase de búsqueda y recupera los resultados del buscador correspondiente.

```
Variables:
    protected java.lang.String frase
    private int maxResults
    private SearchLink[] links
    private ExchangeChannel inChannel
    private ExchangeChannel outChannel
    private int slice
```

```
Constructor:
    public SearchAgent()
          Constructor por defecto.
    public SearchAgent(ExchangeChannel inChannel, ExchangeChannel outChannel)
          Constructor que setea el canal de entrada y el canal de salida.
Métodos públicos:
    protected WordKey[] divideFrase()
          Divide una frase de búsqueda en las palabras y subfrases que la componen.
    public ExchangeChannel getInChannel()
          Retorna el canal de entrada.
    public int getMaxResults()
          Retorna la cantidad máxima de resultados a devolver.
    public ExchangeChannel getOutChannel()
          Retorna el canal de salida
    public int getSlice()
          Retorna el slice de la busqueda
    protected abstract boolean isLastResult(java.lang.String doc)
          Método abstracto a ser implementado en las subclases, que debe devolver si es
          la ultima página de resultados.
    protected abstract java.lang.String makeNextUrl(int pos)
          Método abstracto a ser implementado en las subclases, que debe devolver la
          URL de la siguiente página de resultados.
    protected abstract java.lang.String makeUrl()
          Método abstracto a ser implementado en las subclases, que debe devolver la
          URL de la primer página de resultados.
    public void run()
          Recupera la frase de búsqueda del canal de entrada, genera la URL de donde
          recuperar la primer página de resultados a partir del método makeUrl(),
          recupera la página correspondiente y la envía por el canal de salida.
          Sigue recuperando páginas de resultados cuyas URLs fueron constituidas por el
          método makeNextUrl() mientras el método isLastResult() devuelve false. Todos los
          métodos mencionados son implementados en las subclases.
    public void setInChannel(ExchangeChannel newInChannel)
          Setea el canal de entrada.
    public void setMaxResults(int newMaxResults)
          Setea la cantidad máxima de resultados a devolver.
    public void setOutChannel(ExchangeChannel newOutChannel)
          Setea el canal de salida.
    public void setSlice(int newSlice)
          Setea el slice.
    protected java.lang.String[] split(java.lang.String wordSet)
          Retorna las palabras que componen una frase.
```

Class ar.com.bipi.agents.SearchAltavistaAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.agents.SearchAgent
                        |
                        +----ar.com.bipi.agents.SearchAltavistaAgent
```

```
public final class SearchAltavistaAgent extends SearchAgent
```

Encapsula el funcionamiento del Agente Específico de Búsqueda de Altavista y recupera sus resultados.

```
Constructor:
    public SearchAltavistaAgent()
          Constructor por defecto.

Métodos públicos:
    protected boolean isLastResult(java.lang.String doc)
          Retorna si es la última página de resultados.
    protected java.lang.String makeNextUrl(int pos)
          Retorna la URL de la página de resultados a partir del resultado en la posición
          pos.
    protected java.lang.String makeUrl()
          Retorna la URL de la primer página de resultados.
```

Class ar.com.bipi.agents.SearchBipiAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.SearchAgent
                    |
                    +----ar.com.bipi.agents.SearchBipiAgent
```

```
public final class SearchBipiAgent extends SearchAgent
```

Encapsula el funcionamiento del Agente Específico de Búsqueda de BIPI y recupera sus resultados.

```
Constructor:
    public SearchBipiAgent()

Métodos públicos:
    protected boolean isLastResult(java.lang.String doc)
          Retorna si es la última pagina de resultados.
    protected java.lang.String makeNextUrl(int pos)
          Retorna la URL de la página de resultados a partir del resultado en la posición
          pos.
    protected java.lang.String makeUrl()
          Retorna la URL de la primer página de resultados.
```

Class ar.com.bipi.agents.SearchExciteAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.SearchAgent
                    |
                    +----ar.com.bipi.agents.SearchExciteAgent
```

```
public final class SearchExciteAgent extends SearchAgent
```

Encapsula el funcionamiento del Agente Específico de Búsqueda de Excite y recupera sus resultados.

```
Constructor:
    public SearchExciteAgent()
          searchAltavistaAgent constructor comment.

Métodos públicos:
    protected boolean isLastResult(java.lang.String doc)
          Retorna si es la última pagina de resultados.
    protected java.lang.String makeNextUrl(int pos)
          Retorna la URL de la página de resultados a partir del resultado en la posición
          pos.
    protected java.lang.String makeUrl()
          Retorna la URL de la primer página de resultados.
```

Class ar.com.bipi.agents.SearchWebCrawlerAgent

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.agents.SearchAgent
                    |
                    +----ar.com.bipi.agents.SearchWebCrawlerAgent
```

```
public final class SearchWebCrawlerAgent extends SearchAgent
```

Encapsula el funcionamiento del Agente Específico de Búsqueda de WebCrawler y recupera sus resultados.

```
Constructor:
    public SearchWebCrawlerAgent()
          searchAltavistaAgent constructor comment.

Métodos públicos:
    protected boolean isLastResult(java.lang.String doc)
          Retorna si es la última pagina de resultados.
    protected java.lang.String makeNextUrl(int pos)
          Retorna la URL de la página de resultados a partir del resultado en la posición
          pos.
    protected java.lang.String makeUrl()
          Retorna la URL de la primer página de resultados.
```

Class ar.com.bipi.agents.SpiderAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.agents.SpiderAgent

public class SpiderAgent extends java.lang.Thread
```

Esta clase encapsula el funcionamiento del Agente de Indexación.

```
Variables:
    private static int ACCESS_COUNT
    private static int RETRY_COUNT
    private static int EMPTY_COUNT
    private Parser parser
    private java.lang.String extensionList
    private boolean endCicle

Constructor:
    public SpiderAgent()
          Constructor por defecto.

Métodos públicos:
    public static final void accessCount(int value)
          Retorna la cantidad de intentos de acceso a realizar sobre cada URL.
    public static final void emptyCount(int value)
          Retorna la cantidad de intentos de acceso a realizar para verificar que no
          hay más páginas por indexar.
    public boolean isEndCicle()
          Retorna si el Spider fue detenido.
    public static final void retryCount(int value)
          Retorna la cantidad de intentos de acceso a realizar cuando la Base de Datos
          no responda.
    public void run()
          Lee la Base de Datos para obtener los datos necesarios para realizar la
          indexación. Los pasos que sigue el Spider con cada una de las páginas pendientes
          de indexación almacenadas en la Base de Datos son los siguientes: recupera la
          página de la Base de Datos, marca la página como Analizando, accede a la página
          a partir de la URL recuperada de la BD y la recupera si está permitido indexarla
          (lógica de robots.txt), chequea la sintaxis y parsea la página a través de la
          clase Parser, graba la página en la Base de Datos. Marca la página como Indexada
          en la Base de Datos. Si se produjera algún error en el desarrollo de los pasos
          se marca la página como Rechazada con la descripción de rechazo correspondiente.
    public void setEndCicle(boolean newEndCicle)
          Setea que el Spider debe detenerse.
```

C.3 Package ar.com.bipi.connection

- ConnectionAgent
- ConnectionSQLAgent

Class ar.com.bipi.connection.ConnectionAgent

```
java.lang.Object
    |
    +----ar.com.bipi.connection.ConnectionAgent

public abstract class ConnectionAgent extends java.lang.Object
```

Clase abstracta que encapsula el acceso a la Base de Datos.

```
Variables:
    protected java.sql.Connection connection
    private java.sql.ResultSet recordSet
    private boolean EOF

Constructor:
    public ConnectionAgent()
          Constructor por defecto.

Métodos públicos:
    public boolean beginTransaction()
          Setea el comienzo de una trasacción.
    public boolean close()
          Cierra la conexión.
    public boolean commitTransaction()
          Realiza el commit de la trasacción.
    public boolean executeQuery(java.lang.String query)
          Ejecuta una consulta sobre la Base de Datos retornando si la misma pudo
          realizarse.
    public boolean executeUpdate(java.lang.String update)
          Ejecuta una actualización sobre la Base de Datos retornando si la misma pudo
          realizarse.
    public void first()
          Se posiciona al comienzo del conjunto de resultados devuetos por una consulta
          a la BD.
    public Converter getField(java.lang.String field)
          Retorna el campo correspondiente al nombre pasado como parámetro para el registro
          sobre el cual se encuentra posicionado.
    public boolean isEof()
          Retorna si hay más resultados
    public void next()
          Se posiciona en el siguiente registro del conjunto de resultados devuetos por una
          consulta a la BD.
    public void rollbackTransaction()
          Vuelve atras una transacción.
```

Class ar.com.bipi.connection.ConnectionSQLAgent

```
java.lang.Object
    |
    +----ar.com.bipi.connection.ConnectionAgent
            |
            +----ar.com.bipi.connection.ConnectionSQLAgent

public final class ConnectionSQLAgent extends ConnectionAgent
```

Encapsula la conexión al Motor de Base de Datos SQLServer.

```
Constructor:
    public ConnectionSQLAgent(java.lang.String driver, java.lang.String user,
                              java.lang.String password)
          Constructor por defecto que inicializa la conexión con los drivers
          correspondientes al motor SQL Server.
```

C.4 Package ar.com.bipi.consoles

- AdreessServerConsole

- AltavistaGroupSearchServerConsole
- BipiGeneralManager
- BipiGroupSearchServerConsole
- ExciteGroupSearchServerConsole
- MaintenanceAgentRun
- SpiderAgentRun
- WebCrawlerGroupSearchServerConsole

Class ar.com.bipi.consoles.AdreessServerConsole

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.AdreessServerConsole

public class AdreessServerConsole extends java.lang.Object
```

Consola de manejo del Agente Servidor de Direcciones

```
Constructor:
    public AdreessServerConsole()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Agente Servidor de Direcciones y lo dispara.
```

Class ar.com.bipi.consoles.AltavistaGroupSearchServerConsole

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.AltavistaGroupSearchServerConsole

public class AltavistaGroupSearchServerConsole extends
java.lang.Object
```

Consola de manejo del Servidor de Grupos de Búsqueda de Altavista.

```
Constructor:
    public AltavistaGroupSearchServerConsole()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Servidor del Grupo de Búsqueda de Altavista y lo dispara.
          Parámetros:
          host: host sobre el cual debe correr el servidor.
          portMultiServer: port por el cual escucha el servidor.
          portSearchAgent: port de inicio por donde escucha el SearchAgent.
```

Class ar.com.bipi.consoles.BipiGeneralManager

```
java.lang.Object
   |
   +----java.awt.Component
           |
           +----java.awt.Container
                   |
                   +----java.awt.Window
                           |
                           +----java.awt.Frame
```

```
                                    |
                                    +----javax.swing.JFrame
                                            |
                                            +----ar.com.bipi.consoles.BipiGeneralManager

public class BipiGeneralManager extends javax.swing.JFrame
```

Consola visual para el manejo de todos los Servidores.

```
Constructor:
    public BipiGeneralManager()
          Constructor por defecto
    public BipiGeneralManager(java.lang.String title)
          Constructor que setea el título de la consola.

Métodos públicos:
    public void initAddressServer()
          Inicializa y dispara el Agente Servidor de Direcciones
    public void initAltavistaServer(java.awt.event.ActionEvent actionEvent)
          Inicializa y dispara el Servidor de Grupos de Búsqueda de Altavista
    public void initBipiServer()
          Inicializa y dispara el Servidor de Grupos de Búsqueda de Bipi
    public void initExciteServer(java.awt.event.ActionEvent actionEvent)
          Inicializa y dispara el Servidor de Grupos de Búsqueda de Excite
    public void initMaintenanceAgent(java.awt.event.ActionEvent actionEvent)
          Inicializa y dispara el \AAM.
    public void initSpiderAgent(java.awt.event.ActionEvent actionEvent)
          Inicializa y dispara el Agente de Indexación.
    public void initWebCrawlerServer()
          Inicializa y dispara el Servidor de WebCrawler.
    public static void main(java.lang.String args[])
          Instancia y ejecuta la aplicación Stand Alone.
    public void showAboutBox()
          Muestra la pantalla "Acerca de".
    public void stopAddressServer(java.awt.event.ActionEvent actionEvent)
          Detiene el Agente Servidor de Direcciones.
    public void stopAltavista(java.awt.event.ActionEvent actionEvent)
          Detiene el Servidor de Grupos de Búsqueda de Altavista.
    public void stopBipi(java.awt.event.ActionEvent actionEvent)
          Detiene el Servidor de Grupos de Búsqueda de Bipi.
    public void stopExcite(java.awt.event.ActionEvent actionEvent)
          Detiene el Servidor de Grupos de Búsqueda de Excite.
    public void stopSpiderAgent(java.awt.event.ActionEvent actionEvent)
          Detiene el Agente de Indexación.
    public void stopWebCrawler(java.awt.event.ActionEvent actionEvent)
          Detiene el Servidor de Grupos de Búsqueda de WebCrawler.
    public void viewStatusBar()
          Muestra la barra de estado.
```

Class ar.com.bipi.consoles.BipiGroupSearchServerConsole

```
    java.lang.Object
        |
        +----ar.com.bipi.consoles.BipiGroupSearchServerConsole

public class BipiGroupSearchServerConsole extends java.lang.Object
```

Consola de manejo del Servidor de Grupos de Búsqueda de Bipi

```
Constructor:
    public BipiGroupSearchServerConsole()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el servidor de Grupos de Búsqueda de BIPI y lo dispara. Parámetros:
          host: host sobre el cual debe correr el servidor.
          portMultiServer: port por el cual escucha el servidor.
          portSearchAgent: port de inicio por donde escucha el SearchAgent.
```

Class ar.com.bipi.consoles.ExciteGroupSearchServerConsole

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.ExciteGroupSearchServerConsole

public class ExciteGroupSearchServerConsole extends
java.lang.Object
```

Consola de manejo del Servidor de Grupos de Búsqueda de Excite.

```
Constructor:
    public ExciteGroupSearchServerConsole()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Server del Grupo de Busqueda de Excite y lo dispara. Parámetros:
          host: host sobre el cual debe correr el Server.
          portMultiServer: port por el cual escucha el Server.
          portSearchAgent: port de inicio por donde escucha el SearchAgent.
```

Class ar.com.bipi.consoles.MaintenanceAgentRun

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.MaintenanceAgentRun

public class MaintenanceAgentRun extends java.lang.Object
```

Consola de manejo del Agente Autónomo de Mantenimiento.

```
Constructor:
    public MaintenanceAgentRun()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Agente de Mantenimiento y lo dispara.
```

Class ar.com.bipi.consoles.SpiderAgentRun

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.SpiderAgentRun

public class SpiderAgentRun extends java.lang.Object
```

Consola de manejo del Spider.

```
Constructor:
    public SpiderAgentRun()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Agente de Indexación y lo dispara. Parámetros:
          emptyCount: cantidad de intentos para encontrar una página a indexar en la BD.
          Default = 1.
          retryCount: cantidad de intentos ante una caída de la BD. Default = 1.
          accessCount: cantidad de intentos de acceso a la página en Internet. Default = 3
```

Class ar.com.bipi.consoles.WebCrawlerGroupSearchServerConsole

```
java.lang.Object
    |
    +----ar.com.bipi.consoles.WebCrawlerGroupSearchServerConsole

public class WebCrawlerGroupSearchServerConsole extends
java.lang.Object
```

Consola de manejo del Servidor de Grupos de Búsqueda de WebCrawler.

```
Constructor:
    public WebCrawlerGroupSearchServerConsole()
          Constructor por defecto.

Métodos públicos:
    public static void main(java.lang.String args[])
          Instancia el Server del Grupo de Busqueda de WebCrawler y lo dispara. Parámetros:
          host: host sobre el cual debe correr el Server.
          portMultiServer: port por el cual escucha el Server.
          portSearchAgent: port de inicio por donde escucha el SearchAgent.
```

C.5 Package ar.com.bipi.servers

- AddressServerAgent
- SearchGroupAltavistaServerAgent
- SearchGroupBipiServerAgent
- SearchGroupExciteServerAgent
- SearchGroupServerAgent
- SearchGroupWebCrawlerServerAgent

Class ar.com.bipi.servers.AddressServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.AddressServerAgent

public class AddressServerAgent extends java.lang.Thread
```

Encapsula el funcionamiento del Agente Servidor de Direcciones.

```
Variables:
    private java.util.Hashtable addressTable
    private java.util.Hashtable incrementalTable
    private static AddressServerAgent addressServerAgent

Constructor:
    private AddressServerAgent()
          AddressServerAgent constructor comment.

Métodos públicos:
    public static AddressServerAgent getInstance()
          Devuelve una única instancia de la clase.
    public void run()
          Inicializa el server y comienza a escuchar en el host:port por defecto y procesa
          las siguientes peticiones:
          Incorporación de dirección: al recibir un mensaje de incorporación de dirección
          la almacena en su base interna
          Pedido de dirección: devuelve la dirección correspondiente al elemento sobre el
          cual preguntan de su base interna
          Eliminación de dirección: elimina la dirección correspondiente al elemento
          informado de su base interna
```

Class ar.com.bipi.servers.SearchGroupAltavistaServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.SearchGroupServerAgent
                        |
                        +----ar.com.bipi.servers.SearchGroupAltavistaServerAgent

public final class SearchGroupAltavistaServerAgent extends
SearchGroupServerAgent
```

Grupo de Búsqueda de Altavista.

```
Constructor:
    public SearchGroupAltavistaServerAgent(java.lang.String host, int port,
                                           int firstPortSearch)
          Constructor por defecto que setea:
          host: sobre el cual corre el Grupo de Búsqueda.
          port: en el cual escucha el Grupo de Búsqueda.
          firtsPortSearch: puerto inicial sobre el cual escucha el componente \ABE.

Métodos públicos:
    protected void createSearchGroup(AddressData address, int portWait)
          createSearchGroup method comment.
```

Class ar.com.bipi.servers.SearchGroupBipiServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.SearchGroupServerAgent
                        |
                        +----ar.com.bipi.servers.SearchGroupBipiServerAgent

public final class SearchGroupBipiServerAgent extends
SearchGroupServerAgent
```

Grupo de Búsqueda de Bipi.

```
Constructor:

public SearchGroupBipiServerAgent(java.lang.String host, int port,
                                  int firstPortSearch)
          Constructor por defecto que setea:
          host: sobre el cual corre el Grupo de Búsqueda.
          port: en el cual escucha el Grupo de Búsqueda.
          firstPortSearch: puerto inicial sobre el cual escucha el componente Agente
          de Búsqueda.

Métodos públicos:
    protected void createSearchGroup(AddressData address, int portWait)
          Crea un grupo de búsqueda.
```

Class ar.com.bipi.servers.SearchGroupExciteServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.SearchGroupServerAgent
                        |
                        +----ar.com.bipi.servers.SearchGroupExciteServerAgent

public final class SearchGroupExciteServerAgent extends
SearchGroupServerAgent
```

Grupo de Búsqueda de Excite.

```
Constructor:
    public SearchGroupExciteServerAgent(java.lang.String host, int port,
                                        int firstPortSearch)
          Constructor por defecto que setea:
          host: sobre el cual corre el Grupo de Búsqueda.
          port: en el cual escucha el Grupo de Búsqueda.
          firstPortSearch: puerto inicial sobre el cual escucha el componente Agente
          de Búsqueda.

Métodos públicos:
    protected void createSearchGroup(AddressData address, int portWait)
          Crea un grupo de búsqueda.
```

Class ar.com.bipi.servers.SearchGroupServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.SearchGroupServerAgent

public abstract class SearchGroupServerAgent extends
java.lang.Thread
```

Clase abstracta que implementa el funcionamiento de un Grupo de Búsqueda.

```
Variables:
    private static java.lang.String host
    private static int port
    private static int portSearch

Constructor:
    protected SearchGroupServerAgent(java.lang.String host, int port,
                                     int firstPortSearch)
          Constructor por defecto que setea:
          host: sobre el cual corre el Grupo de Búsqueda.
          port: en el cual escucha el Grupo de Búsqueda.
          firstPortSearch: puerto inicial sobre el cual escucha el componente \ABE.
          Este método es únicamente utilizado por los constructores de
          las subclases

Métodos públicos:
    protected abstract void createSearchGroup(AddressData address, int portWait)
          Método abstracto que debe implementarse en las subclases
    protected void createSearchGroup(AddressData address, int portWait,
                           java.lang.String classSearch, java.lang.String classFilter)
          Crea un grupo de búsqueda seteando la direccion, el port por donde escucha y
          el Agente de Búsqueda y Filtrado
    protected void createSearchGroup(AddressData address, int portWait,
            java.lang.String classSearch, java.lang.String classFilter, int slice)
          Crea un grupo de búsqueda seteando la dirección, el port por donde escucha,
          el Agente de Búsqueda y Filtrado y el slice.
    public void run()
          Recibe una frase de búsqueda y crea un grupo de Búsqueda para
          satisfacer la búsqueda
```

Class ar.com.bipi.servers.SearchGroupWebCrawlerServerAgent

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.servers.SearchGroupServerAgent
                        |
                        +----ar.com.bipi.servers.SearchGroupWebCrawlerServerAgent

public final class SearchGroupWebCrawlerServerAgent extends
SearchGroupServerAgent
```

Grupo de Búsqueda de WebCrawler.

```
Constructor:
    public SearchGroupWebCrawlerServerAgent(java.lang.String host, int port,
                                            int firstPortSearch)
          Constructor por defecto que setea:
          host: sobre el cual corre el Grupo de Búsqueda.
          port: en el cual escucha el Grupo de Búsqueda.
          firstPortSearch: puerto inicial sobre el cual escucha el componente \ABE.

Métodos públicos:
    protected void createSearchGroup(AddressData address,
                                     int portWait)
          Crea un grupo de búsqueda.
```

C.6 Package ar.com.bipi.system

- BipiAdmin
- BipiSearch

Class ar.com.bipi.system.BipiAdmin

```
    java.lang.Object
        |
        +----ar.com.bipi.system.BipiAdmin

public class BipiAdmin extends java.lang.Object
```

Encapsula la actualización y consulta de los grupos de descriptores y de los parámetros que influyen en la indexación y ranqueo de páginas.

```
Variables:
    private int penalizationFactor
    private int linksFactor
    private int relevanceTagFactor
    private int cantWordsFactor
    private int minLengthWord
    private int caducityDays
    private int minSpammingWords
    private java.util.Vector allTags
    private int posTag
    private GroupTag itemTag

Constructor:
    public BipiAdmin()
          Constructor por defecto.

Métodos públicos:
    public int getCaducityDays()
          Retorna el parámetro días de caducidad de una página.
    public int getCantWordsFactor()
          Retorna el factor de cantidad de palabras.
    public int getLinksFactor()
          Retorna el factor de enlaces.
    public int getMinLengthWord()
          Retorna la longitud mínima de palabra tenida en cuenta.
    public int getMinSpammingWords()
          Retorna la cantidad mínima de palabras que es considerada spamming.
    public int getPenalizationFactor()
          Retorna el factor de penalizaciones.
    public int getRelevanceTagFactor()
          Retorna el factor de relevancia de descriptores.
    public java.lang.String getTagDescription()
          Retorna la descripción del grupo de descriptores sobre el cual se está posicionado.
    public int getTagIdGroupTag()
          Retorna el identificador del grupo de descriptores sobre el cual se está posicionado.
```

```
    public int getTagRelevance()
          Retorna la relevancia del grupo de descriptores sobre el cual se está posicionado.
    public boolean isEofTag()
          Retorna es el último grupo de descriptores del conjunto.
    public boolean load()
          Carga los parámetros y el conjunto de grupos de descriptores existentes en la BD.
    public void nextTag()
          Se posiciona en el siguiente grupo de descriptores del conjunto de descriptores.
    public boolean save()
          Actualiza los parámetros y relevancia de los grupos de descriptores de la
          Base de Datos.
    public void setCaducityDays(int newCaducityDays)
          Setea el parámetro días de caducidad de una página.
    public void setCantWordsFactor(int newCantWordsFactor)
          Setea el factor de cantidad de palabras.
    public void setLinksFactor(int newLinksFactor)
          Setea el factor de cantidad de enlaces.
    public void setMinLengthWord(int newMinLengthWord)
          Setea la longitud mínima de palabra tenida en cuenta.
    public void setMinSpammingWords(int newMinSpammingWords)
          Setea la cantidad mínima de palabras que es considerada spamming.
    public void setPenalizationFactor(int newPenalizationFactor)
          Setea el factor de penalizaciones.
    public void setRelevanceTag(int idGroupTag, int relevance)
          Setea la relevancia del grupo de descriptores indicado en idGroupTag
    public void setRelevanceTagFactor(int newRelevanceTagFactor)
          Setea el factor de relevancia del descriptor.
```

Class ar.com.bipi.system.BipiSearch

```
    java.lang.Object
        |
        +----ar.com.bipi.system.BipiSearch

public class BipiSearch extends java.lang.Object
```

Encapsula una búsqueda. Realiza la creación de los componentes necesarios para satisfacer una búsqueda específica. Es el Sub-entorno de Búsqueda que corre en la aplicación local.

```
Variables:
    private CoordinatorAgent aCoordinador
    private CompactAgent aCompactador
    private ExchangeChannel returnChannel
    private ContinuesSearchAgent aABC
    private AddressData address
    private java.lang.String frase
    private boolean ready
    private java.util.Vector results
    private java.util.Vector searchEngines

Constructor:
    public BipiSearch()
          Constructor por defecto.

Métodos públicos:
    public int getCountResults()
          Retorna la cantidad de resultados obtenidos por la búsqueda realizada.
    public java.lang.String getFrase()
          Retorna la frase de búsqueda.
    public SearchLink[] getResults()
          Retorna el conjunto de resultados de la búsqueda realizada.
    public SearchLink[] getResults(int first, int last)
          Retorna los resultados de la búsqueda realizada comprendidos entre las
          posiciones first y last.
    public void searchFor(java.lang.String frase)
          Realiza los seteos necesarios para la búsqueda de la frase pasada como parámetro,
          inicializando el Agente Coordinador, el Agente Compactador, el Agente de Búsqueda
          Continua y los canales de comunicación interna.
    public void start()
          Dispara el Agente Coordinador, el Agente Compactador, el Agente de Búsqueda
```

```
      Continua y se pone a la espera de los resultados de la búsqueda. A medida que
      llegan los resultados los almacena en su vector.
public void useAltavista()
      Indica que se debe utilizar el motor de búsqueda de Altavista para satisfacer la
      búsqueda
public void useBipi()
      Indica que se debe utilizar el motor de búsqueda de Bipi para satisfacer la
      búsqueda.
public void useExcite()
      Indica que se debe utilizar el motor de búsqueda de Excite para satisfacer la
      búsqueda.
public void useWebCrawler()
      Indica que se debe utilizar el motor de búsqueda de WebCrawler para satisfacer la
      búsqueda.
public boolean wasUsed(java.lang.String searchEngineName)
      Retorna si el buscador de nombre searchEngineName que está siendo utilizado.
```

C.7 Package ar.com.bipi.types

- KQMLMessageInterface
- AddressData
- Constants
- Converter
- ExchangeChannel
- ExchangeChannelClient
- ExchangeChannelElement
- ExchangeChannelMultiServer
- ExchangeChannelServer
- GroupTag
- ItemFullTag
- KQMLMessage
- OrderedCollection
- Pair
- Parser
- RequestAddressData
- SearchData
- SearchLink
- SocketExchangeChannel
- SocketExchangeChannelClient
- SocketExchangeChannelMultiServer

- SocketExchangeChannelMultiServerUnit
- SocketExchangeChannelServer
- UrlManager
- WordKey

Interface ar.com.bipi.types.KQMLMessageInterface

```
public abstract interface KQMLMessageInterface
```

Interface que deben implementar todas las clases que deseen poder convertirse a mensajes KQML.

```
Métodos públicos:
    public java.lang.Object fromKQMLMessage(java.lang.String message)
          Instancia un Objeto a partir del contenido de un mensaje KQML.
    public java.lang.String toKQMLMessage()
          Genera el contenido de un mensaje KQML.
```

Class ar.com.bipi.types.AddressData

```
    java.lang.Object
        |
        +----ar.com.bipi.types.AddressData

public class AddressData extends java.lang.Object implements
KQMLMessageInterface
```

Encapsula una dirección IP y un puerto para ser utilizados en los sockets. Esta clase es usada en el intercambio de información entre agentes, por ello, implementa la interfaz KQMLMessageInterface.

```
Variables:
    private java.lang.String host
    private int port

Constructor:
    public AddressData()
          Constructor por defecto.
    public AddressData(java.lang.String host, int port)
          Constructor que inicializa la clase con los valores host y port.

Métodos públicos:
    public java.lang.Object fromKQMLMessage(java.lang.String message)
          Genera un AddressData a partir del contenido de un mensaje KQML.
    public java.lang.String getHost()
          Retorna el host.
    public int getPort()
          Retorna el port.
    public void setHost(java.lang.String newHost)
          Setea el host.
    public void setPort(int newPort)
          Setea el port.
    public java.lang.String toKQMLMessage()
          Genera el contenido de un mensaje KQML a partir de si mismo.
```

Class ar.com.bipi.types.Constants

```
    java.lang.Object
        |
        +----ar.com.bipi.types.Constants

public class Constants extends java.lang.Object
```

Clase en la que se almacenan las constantes del proyecto ar.com.bipi.

```
Constructor:
    public Constants()
```

Class ar.com.bipi.types.Converter

```
java.lang.Object
    |
    +----ar.com.bipi.types.Converter

public class Converter extends java.lang.Object
```

Conversor utilizado para realizar el casting de tipos básicos.

```
Variables:
    private java.lang.Object obj

Constructor:
    public Converter()
          Constructor por defecto.
    public Converter(java.lang.Object obj)
          Constructor que se inicia con el objeto a ser convertido.

Métodos públicos:
    public boolean toBoolean()
          Se retorna a sí mismo como boolean.
    public float toFloat()
          Se retorna a sí mismo como float.
    public int toInt()
          Se retorna a sí mismo como int
    public java.lang.Object toObject()
          Se retorna a sí mismo.
    public java.lang.String toPrintableFloat(int decimals)
          Se retorna a sí mismo como cadena con formato numérico con decimales.
    public java.lang.String toString()
          Convierte el objeto a string.
```

Class ar.com.bipi.types.ExchangeChannel

```
java.lang.Object
    |
    +----ar.com.bipi.types.ExchangeChannel

public class ExchangeChannel extends java.lang.Object
```

Implementa un canal de comunicación. Sobre el canal se pueden realizar inserciones y recuperaciones de elementos. La recuperación de elementos bloquea al consumidor hasta tanto haya elementos en el canal.

```
Variables:
    private java.util.Vector buffer
    private boolean isClosed

Constructor:
    public ExchangeChannel()
          Constructor por defecto.

Métodos públicos:
    public void close()
          Cierra el canal.
    public boolean get(ExchangeChannelElement element)
          Recupera un elemento. El consumidor queda bloqueado mientras no haya elementos en
          el canal y este no se cierre.
    public boolean isClosed()
          Retorna si el canal esta cerrado.
    public boolean isEmpty()
          Retorna si el canal esta vacio.
    public void put(ExchangeChannelElement element)
          Inserta un elemento en el canal.
```

Class ar.com.bipi.types.ExchangeChannelClient

```
java.lang.Object
```

```
        |
        +----ar.com.bipi.types.ExchangeChannel
                |
                +----ar.com.bipi.types.ExchangeChannelClient

public class ExchangeChannelClient extends ExchangeChannel
```

Implementa un canal que es utilizado por un cliente en una arquitectura Cliente/Servidor. La comunicación se realiza a través de sockets. Trabaja como cliente de SocketExchangeChannelServer.

```
Variables:
    private SocketExchangeChannelClient socket

Constructor:
    public ExchangeChannelClient(java.lang.String host, int port)
          Constructor que instancia Socket para conectarse como emisor de datos hacia
          el host y port pasados como parámetros.
```

Class ar.com.bipi.types.ExchangeChannelElement

```
    java.lang.Object
        |
        +----ar.com.bipi.types.ExchangeChannelElement

public class ExchangeChannelElement extends java.lang.Object
```

Encapsula el elemento a ser trasportado a través de un canal.

```
Variables:
    private java.lang.Object element

Constructor:
    public ExchangeChannelElement()
          Constructor por defecto.
    public ExchangeChannelElement(java.lang.Object element)
          Constructor que almacena el elemento a encapsular.

Métodos públicos:
    public java.lang.Object element()
          Recupera el elemento encapsulado.
    public void element(java.lang.Object element)
          Setea el elemento a encapsular.
    public KQMLMessageInterface KQMLElement()
          Retorna el elemento como contenido de un mensaje KQML.
    public java.lang.String toString()
          Retorna un string como parte visible de sí mismo.
```

Class ar.com.bipi.types.ExchangeChannelMultiServer

```
    java.lang.Object
        |
        +----ar.com.bipi.types.ExchangeChannel
                |
                +----ar.com.bipi.types.ExchangeChannelMultiServer

public class ExchangeChannelMultiServer extends ExchangeChannel
```

Encapsula el funcionamiento de un canal para acceso como MultiServer, de esta forma acepta la conexión de varios clientes SocketExchangeChannelClient en una arquitectura Cliente/Servidor.

```
Variables:
    private SocketExchangeChannelMultiServer socket

Constructor:
    public ExchangeChannelMultiServer(int port)
          Constructor que crea un SocketExchangeChannelMultiServer para escuchar en el
```

```
        port pasado como parámetro.

Métodos públicos:
    public void close()
        Cierra el canal.
    public void closeFather()
        Cierra el padre.
    public void waitForClients(int count)
        Setea por cuantos clientes debe esperar.
```

Class ar.com.bipi.types.ExchangeChannelServer

```
    java.lang.Object
        |
        +----ar.com.bipi.types.ExchangeChannel
                |
                +----ar.com.bipi.types.ExchangeChannelServer

public class ExchangeChannelServer extends ExchangeChannel
```

Implementa un canal que es utilizado por un servidor en una arquitectura Cliente/Servidor. La comunicación se realiza a través de sockets y acepta solo un cliente SocketExchangeChannelClient.

```
Variables:
    private SocketExchangeChannelServer socket

Constructor:
    public ExchangeChannelServer(int port)
        Constructor que crea una instancia de Socket para escuchar en el port pasado
        como parámetro.
```

Class ar.com.bipi.types.GroupTag

```
    java.lang.Object
        |
        +----ar.com.bipi.types.GroupTag

public class GroupTag extends java.lang.Object
```

Encapsula los datos que hacen a la definición de un grupo de descriptores.

```
Variables:
    private int idGroupTag
    private java.lang.String description
    private int relevance

Constructor:
    public GroupTag()
        Constructor por defecto.

Métodos públicos:
    public boolean equals(GroupTag itemTag)
        Compara dos grupos de descriptores y retorna verdadero
        si tienen el mismo identificador.
    public boolean equals(java.lang.Object itemTag)
        Compara un grupo de descriptores con un Object y retorna verdadero
        si tienen el mismo identificador.
    public java.lang.String getDescription()
        Retorna la descripción del grupo de descriptores.
    public int getIdGroupTag()
        Retorna el identificador del grupo de descriptores.
    public int getRelevance()
        Retorna la relevancia del grupo de descriptores.
    public void setDescription(java.lang.String newDescription)
        Setea la descripción del grupo de descriptores.
    public void setIdGroupTag(int newIdGroupTag)
        Setea el identificador del grupo de descriptores.
    public void setRelevance(int newRelevance)
        Setea la relevancia del grupo de descriptores.
```

Class ar.com.bipi.types.ItemFullTag

```
    java.lang.Object
        |
        +----ar.com.bipi.types.ItemFullTag

public class ItemFullTag extends java.lang.Object
```

Encapsula la información de un descriptor, más ciertas características adicionales.

```
Variables:
    private int idGroupTag
    private int idTag
    private java.lang.String sTag
    private boolean isRepetible
    private boolean isDoubleTag
    private java.lang.String sDoubleTag
    private boolean isMetaTag
    private boolean isLink

Constructor:
    public ItemFullTag()
          Constructor por defecto.

Métodos públicos:
    public static ItemFullTag createFrom(ItemFullTag itemTag)
          Constructor que genera un ItemFullTag a partir de otro.
    public boolean equals(ItemFullTag itemTag)
          Compara dos ItemFullTag y retorna verdadero si tienen el mismo id.
    public boolean equals(java.lang.Object itemTag)
          Compara un ItemFullTag con un Object y retorna verdadero si tienen el mismo id.
    public int getIdGroupTag()
          Retorna el identificador del grupo al cual pertenece el
          descriptor.
    public int getIdTag()
          Retorna el identificador del descriptor.
    public java.lang.String getSDoubleTag()
          Retorna la descripción de un descriptor doble.
    public java.lang.String getSTag()
          Retorna la descripción del descriptor.
    public boolean isIsDoubleTag()
          Retorna si el descriptor es doble.
    public boolean isIsLink()
          Retorna si el descriptor es del grupo enlaces.
    public boolean isIsMetaTag()
          Retorna si es un descriptor del tipo Meta.
    public boolean isIsRepetible()
          Retorna si el descriptor puede aparecer más de una vez en cada página.
    public void setIdGroupTag(int newIdGroupTag)
          Setea el identificador del grupo al cual pertenece el descriptor.
    public void setIdTag(int newIdTag)
          Setea el identificador del descriptor.
    public void setIsDoubleTag(boolean newIsDoubleTag)
          Setea si el descriptor es doble.
    public void setIsLink(boolean newIsLink)
          Setea si el descriptor es del grupo enlaces.
    public void setIsMetaTag(boolean newIsMetaTag)
          Setea si es un descriptor del tipo Meta.
    public void setIsRepetible(boolean newIsRepetible)
          Setea si el tag se puede repetir en la página.
    public void setSDoubleTag(java.lang.String newSDoubleTag)
          Setea la palabra que compone la descripción de un
          descriptor doble.
    public void setSTag(java.lang.String newSTag)
          Setea la descripción del descriptor.
    public java.lang.String toString()
          Retorna la forma visible del ItemFullTag.
```

Class ar.com.bipi.types.KQMLMessage

```
java.lang.Object
    |
    +----ar.com.bipi.types.KQMLMessage

public class KQMLMessage extends java.lang.Object
```

Encapsula la estructura y funcionamiento de un mensaje KQML.

```
Variables:
    public static java.lang.String KQML_SET_ADDRESS
    public static java.lang.String KQML_GET_ADDRESS
    public static java.lang.String KQML_REMOVE_ADDRESS
    public static java.lang.String KQML_GET_GS_ADDRESSES
    public static java.lang.String KQML_GET_ADDRESS_RESPONSE
    public static java.lang.String KQML_STOP
    public static java.lang.String KQML_SET_PHRASE
    public static java.lang.String KQML_SET_RESULT_ADDRESS
    public static java.lang.String KQML_SET_RESULT
    private java.lang.String performative
    private java.lang.String receiver
    private java.lang.String sender
    private java.lang.String language
    private java.lang.String ontology
    private java.lang.String typeClass
    private java.lang.String empty
    private java.lang.String content

Constructor:
    public KQMLMessage()
          KQMLMessage constructor comment.

Métodos públicos:
    public static KQMLMessage fromString(java.lang.String message)
          Constructor que genera un mensaje KQML a partir de un String.
    public java.lang.String getContent()
          Retorna el contenido del mensaje.
    public java.lang.String getEmpty()
          Retorna si el contenido del mensaje es vacio.
    public java.lang.String getLanguage()
          Retorna el lenguaje del mensaje.
    public java.lang.String getOntology()
          Retorna la ontologia del mensaje.
    public java.lang.String getPerformative()
          Retorna la performativa del mensaje.
    public java.lang.String getReceiver()
          Retorna el receptor del mensaje.
    public java.lang.String getSender()
          Retorna el emisor del mensaje.
    public java.lang.String getTypeClass()
          Retorna la clase del objeto contenido en el mensaje.
    public boolean isEmpty()
          Retorna si el mensaje es vacio.
    public void setContent(java.lang.String newContent)
          Setea el contenido del mensaje.
    public void setLanguage(java.lang.String newLanguage)
          Setea el lenguaje del mensaje.
    public void setNullContent()
          Vacia el contenido del mensaje.
    public void setOntology(java.lang.String newOntology)
          Setea la ontología del mensaje.
    public void setPerformative(java.lang.String newPerformative)
          Setea la perfomativa del mensaje.
    public void setReceiver(java.lang.String newReceiver)
          Setea el receptor del mensaje.
    public void setSender(java.lang.String newSender)
          Setea el emisor del mensaje.
    public void setTypeClass(java.lang.String newTypeClass)
          Setea la clase del objeto contenido en el mensaje.
    public java.lang.String toString()
```

```
        Retorna la forma visible del mensaje.
```

Class ar.com.bipi.types.OrderedCollection

```
    java.lang.Object
        |
        +----ar.com.bipi.types.OrderedCollection

public class OrderedCollection extends java.lang.Object
```

Encapsula la estructura y el funcionamiento de una colección que puede ser recorrida en orden, del primer al último elemento.

```
Variables:
    private int actual
    private java.util.Vector elements

Constructor:
    public OrderedCollection()
          Constructor por defecto.

Métodos públicos:
    public void add(java.lang.Object value)
          Agrega un nuevo elemento a la colección.
    public void addElement(java.lang.Object value)
          Agrega un nuevo elemento a la colección.
    public void begin()
          Se posiciona en el primer elemento de la colección.
    public boolean contains(java.lang.Object element)
          Retorna si el elemento existe en la colección.
    public java.lang.Object firstElement()
          Retorna el primer elemento de la colección.
    public java.lang.Object get(java.lang.Object value)
          Retorna un elemento a partir de el mismo.
    public boolean isEof()
          Retorna si hay más elementos que recorrer en la colección.
    public java.lang.Object lastElement()
          Retorna el último elemento de la colección.
    public java.lang.Object next()
          Retorna el elemento actual de la colección y se
          posiciona en el siguiente.
    public boolean removeElement(java.lang.Object element)
          Elimina el elemento de la colección.
    public int size()
          Retorna la cantidad de elementos de la colección.
    public java.lang.Object[] toArray()
          Genera un arreglo con los objetos de la colección.
    public java.lang.String toString()
          Convierte la colección a String.
```

Class ar.com.bipi.types.Pair

```
    java.lang.Object
        |
        +----ar.com.bipi.types.Pair

public class Pair extends java.lang.Object
```

Encapsula la estructura de un par de objetos.

```
Variables:
    private java.lang.Object first
    private java.lang.Object second

Constructor:
    public Pair()
          Constructor por defecto.
    public Pair(java.lang.Object first, java.lang.Object second)
          Constructor que setea los dos objetos que componen el par.
```

```
Métodos públicos:
    public java.lang.Object getFirst()
          Retorna el primer objeto del par.
    public java.lang.Object getSecond()
          Retorna el segundo objeto del par.
    public void setFirst(java.lang.Object newFirst)
          Setea el primer objeto del par.
    public void setSecond(java.lang.Object newSecond)
          Setea el segundo objeto del par.
    public java.lang.String toString()
          Convierte el par a String para poder ser visualizado.
```

Class ar.com.bipi.types.Parser

```
    java.lang.Object
        |
        +----ar.com.bipi.types.Parser

public class Parser extends java.lang.Object
```

Encapsula el funcionamiento del parseador de páginas. Este clase permite desglosar una página en sus partes.

```
Variables:
    private int position
    private OrderedCollection allTags
    private OrderedCollection allConverts
    private OrderedCollection allComments
    private java.util.Vector noRepetibleTags
    private java.util.Hashtable doubleTags
    private java.util.Hashtable simpleTags
    private java.lang.String upperText
    private java.lang.String text

Constructor:
    public Parser()
          Constructor por defecto.
    public Parser(java.lang.String texto)
          Constructor que setea el texto de la página a ser parseado.

Métodos públicos:
    public void addBDTag(int idGroupTag, int idTag, java.lang.String sTag,
                    boolean isRepetible, boolean isDoubleTag, java.lang.String sDoubleTag,
                    boolean isMetaTag, boolean isLink)
          Agrega un nuevo descriptor (con todas sus características) al conjunto de
          descriptores que mantiene internamente
    public void addComment(java.lang.String startTag, java.lang.String endTag)
          Agrega un nuevo comentario (indicando como comienza y como termina) al conjunto
          de comentarios que mantiene internamente.
    public void addConvert(java.lang.String source, java.lang.String target)
          Carga una nueva conversión de caracteres y palabras (indicando origen y destino).
    public void beginAllTags()
          Se posiciona al principio del conjunto de descriptores de la página.
    public boolean checkSintax()
          Chequea inicio-fin de tags y que no se produzcan repeticiones de descriptores marcados
          como no repetibles
    public void createAllTags()
          Genera el conjunto de descriptores que constituyen la página. Desglosa la página
          en una lista de descriptores.
    public java.lang.String getText()
          Retorna el texto de la página gestionada.
    public boolean isEofAllTags()
          Retorna si hay más descriptores en el conjunto de descriptores de la página
    public ItemFullTag nextAllTags()
          Retorna el siguiente descriptor del conjunto de descriptores de la página.
    public void setText(java.lang.String newText)
          Setea el texto de la página a ser parseada a la cual se le quita el texto que se
          encuentra entre comentarios.
    public java.lang.String spammingText()
```

```
        Retorna todas las palabras del texto en mayúsculas
```

Class ar.com.bipi.types.RequestAddressData

```
    java.lang.Object
        |
        +----ar.com.bipi.types.RequestAddressData

public class RequestAddressData extends java.lang.Object
implements KQMLMessageInterface
```

Encapsula la estructura de una respuesta del Agente Servidor de Direcciones. Almacena el nombre de un agente a buscar en la lista de direcciones del Agente Servidor de Direcciones y el host y puerto de devolución de respuesta.

```
Variables:
    private java.lang.String agent
    private java.lang.String host
    private int port

Constructor:
    public RequestAddressData()
          Constructor por defecto.
    public RequestAddressData(java.lang.String agent, AddressData address)
          Constructor que setea el agente y la dirección (host:port) correspondiente
    public RequestAddressData(java.lang.String agent, java.lang.String host, int port)
          Constructor que setea el agente, el host y el port correspondiente

Métodos públicos:
    public java.lang.Object fromKQMLMessage(java.lang.String message)
          Genera un RequestAddressData a partir del contenido de un mensaje KQML (es
          necesario porque implementa la interfaz KQMLMessageInterface)
    public java.lang.String getAgent()
          Retorna el agente.
    public java.lang.String getHost()
          Retorna el host.
    public int getPort()
          Retorna el port.
    public void setAgent(java.lang.String newAgent)
          Setea el agente.
    public void setHost(java.lang.String newHost)
          Setea el host.
    public void setPort(int newPort)
          Setea el port.
    public java.lang.String toKQMLMessage()
          Genera el contenido de un mensaje KQML (es necesario porque implementa la
          interfaz KQMLMessageInterface)
    public java.lang.String toString()
          Convierte el RequestAddressData a String.
```

Class ar.com.bipi.types.SearchData

```
    java.lang.Object
        |
        +----ar.com.bipi.types.SearchData

public class SearchData extends java.lang.Object implements
KQMLMessageInterface
```

Encapsula una frase de búsqueda para poder ser intercambiada entre agentes remotos.

```
Variables:
    private java.lang.String frase

Constructor:
    public SearchData()
          Constructor por defecto.
    public SearchData(java.lang.String newFrase)
          Constructor que setea la frase de búsqueda.
```

```
Métodos públicos:
    public java.lang.Object fromKQMLMessage(java.lang.String message)
          Genera un SearchData a partir del contenido de un mensaje KQML (es necesario
          porque implementa la interfaz KQMLMessageInterface).
    public java.lang.String getFrase()
          Retorna la frase de búsqueda.
    public void setFrase(java.lang.String newFrase)
          Setea la frase de búsqueda.
    public java.lang.String toKQMLMessage()
          Genera el contenido de un mensaje KQML (es necesario porque implementa la
          interfaz KQMLMessageInterface).
    public java.lang.String toString()
          Convierte un SearchData a String.
```

Class ar.com.bipi.types.SearchLink

```
    java.lang.Object
        |
        +----ar.com.bipi.types.SearchLink

public class SearchLink extends java.lang.Object implements
KQMLMessageInterface
```

Encapsula la estructura de un resultado de búsqueda.

```
Variable:
    private java.lang.String SL_SEPARATOR
    private java.lang.String url
    private java.lang.String desc
    private java.lang.String descLong
    private float relevance

Constructor:
    public SearchLink()
          Constructor por defecto.
    public SearchLink(java.lang.String url, java.lang.String desc,
                      java.lang.String descLong)
          Constructor que setea la URL, la descripción, titulo y la descripción
          larga del resultado.
    public SearchLink(java.lang.String url, java.lang.String desc,
                      java.lang.String descLong, float relevance)
          Constructor que setea la URL, el título, la descripción, descripción larga y
          relevancia del resultado.

Métodos públicos:
    public java.lang.String desc()
          Retorna la descripción del resultado.
    public void desc(java.lang.String desc)
          Setea la descripción del resultado.
    public java.lang.String descLong()
          Retorna la descripción larga del resultado.
    public void descLong(java.lang.String descLong)
          Setea la descripción larga del resultado.
    public java.lang.Object fromKQMLMessage(java.lang.String message)
          Genera un SearchLink a partir del contenido de un mensaje KQML (es necesario
          porque implementa la interfaz KQMLMessageInterface)
    public boolean isGreater(SearchLink searchlink)
          Retorna verdadero si la relevancia es mayor que la del SearchLink pasado
          como parámetro.
    public float relevance()
          Retorna la relevancia del resultado.
    public void relevance(float relevance)
          Setea la relavancia del resultado.
    public java.lang.String toKQMLMessage()
          Genera el contenido de un mensaje KQML (es necesario porque implementa la
          interfaz KQMLMessageInterface)
    public java.lang.String toString()
          Se retorna como String.
    public java.lang.String url()
```

```
        Retorna la URL del resultado.
    public void url(java.lang.String url)
        Setea la URL del resultado.
```

Class ar.com.bipi.types.SocketExchangeChannel

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.types.SocketExchangeChannel

public abstract class SocketExchangeChannel extends
java.lang.Thread
```

Clase abstracta que encapsula el funcionamiento de un ExchangeChannel para ser usado como parte de una comunicación por sockets.

```
Variables:
    protected int port
    protected ExchangeChannel channel

Constructor:
    public SocketExchangeChannel()
        Constructor por defecto.
    public SocketExchangeChannel(java.lang.String nameThread)
        Constructor que setea el nombre del Thread.

Métodos públicos:
    public ExchangeChannel getChannel()
        Retorna el canal con el cual interacciona.
    public int getPort()
        Retorna el port.
    public abstract void run()
        Método abstracto implementado en las subclases.
    public void setChannel(ExchangeChannel newChannel)
        Setea el canal con el cual interacciona.
    public void setPort(int newPort)
        Setea el port.
    public abstract java.lang.String toString()
        Método abstracto que convierte el objeto a String.
```

Class ar.com.bipi.types.SocketExchangeChannelClient

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.types.SocketExchangeChannel
                        |
                        +----ar.com.bipi.types.SocketExchangeChannelClient

public class SocketExchangeChannelClient extends
SocketExchangeChannel
```

Encapsula el funcionamiento de un Socket cliente que mantiene comunicación local con un canal del tipo ExchangeChannelClient.

```
Variables:
    protected java.lang.String host

Constructor:
    public SocketExchangeChannelClient(ExchangeChannel channel, java.lang.String host,
                                       int port)
        Constructor que setea el canal con el cual interacciona conjuntamente con el host
        y el port a donde envía.

Métodos públicos:
    public java.lang.String getHost()
```

```
        Retorna el host.
    public void run()
        Toma los valores que se encuentran en el canal con el cual interacciona y los
        envía hacia (host:port).
    public void setHost(java.lang.String newHost)
        Setea el host.
    public java.lang.String toString()
        Convierte a String.
```

Class ar.com.bipi.types.SocketExchangeChannelMultiServer

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.types.SocketExchangeChannel
                        |
                        +----ar.com.bipi.types.SocketExchangeChannelMultiServer

public class SocketExchangeChannelMultiServer extends
SocketExchangeChannel
```

Encapsula el funcionamiento de un servidor que atiende múltiples clientes, a través de un socket, y que interacciona con un canal de tipo ExchangeChannelMultiServer.

```
Variables:
    private int count
    private boolean listening
    private java.net.ServerSocket serverSocket

Constructor:
    public SocketExchangeChannelMultiServer(ExchangeChannel channel, int port)
        Constructor que setea el canal con el cual interacciona y el port por el cual
        escucha.

Métodos públicos:
    private void decCount()
        Decrementa el contador de server-units.
    private int getCount()
        Devuelve el contador de server-units.
    public java.lang.String getHost()
        Retorna el host por el cual escucha.
    public void run()
        Cuando recibe un mensaje de un cliente genera un
        SocketExchangeChannelMultiServerUnit para que atienda el llamado y se queda a la
        espera de un nuevo mensaje.
    public java.lang.String toString()
        Convierte a String
    public void waitForClients(int count)
        Setea con cuantos clientes se va a comunicar.
```

Class ar.com.bipi.types.SocketExchangeChannelMultiServerUnit

```
    java.lang.Object
        |
        +----java.lang.Thread
                |
                +----ar.com.bipi.types.SocketExchangeChannelMultiServerUnit

private class
SocketExchangeChannelMultiServer.SocketExchangeChannelMultiServerUnit
extends java.lang.Thread
```

Encapsula el funcionamiento de un Socket que atiende un cliente y que interacciona con un canal de tipo ExchangeChannelMultiServer.

```
Variables:
    private java.net.Socket clientSocket
```

```
Constructor:
    SocketExchangeChannelMultiServerUnit(java.net.Socket clientSocket)
          Constructor que setea el socket por el cual le corresponde escuchar.

Método:
    public void run()
          Recibe el mensaje por el socket y lo envía al canal correspondiente.
```

Class ar.com.bipi.types.SocketExchangeChannelServer

```
java.lang.Object
    |
    +----java.lang.Thread
            |
            +----ar.com.bipi.types.SocketExchangeChannel
                    |
                    +----ar.com.bipi.types.SocketExchangeChannelServer

public class SocketExchangeChannelServer extends
SocketExchangeChannel
```

Encapsula el funcionamiento de un Socket que atiende un solo cliente y mantiene referencia del canal del tipo ExchangeChannelServer con el cual interacciona.

```
Constructor:
    public SocketExchangeChannelServer(ExchangeChannel channel, int port)
          Constructor que setea el canal con el cual interacciona y el port en donde
          escucha.

Métodos públicos:
    public void run()
          Toma el valor que recibe por el socket (host:port) en donde escucha y se lo envía
          al canal con el cual interacciona.
    public java.lang.String toString()
          Convierte a String
```

Class ar.com.bipi.types.UrlManager

```
    java.lang.Object
        |
        +----ar.com.bipi.types.UrlManager

public class UrlManager extends java.lang.Object
```

Encapsula el acceso a una página en la Web.

```
Variables:
    private java.lang.String url

Constructor:
    public UrlManager(java.lang.String url)
          Constructor que setea la URL de la página a recuperar.

Métodos públicos:
    public java.lang.String getDocument() throws BipiException
          Retorna el contenido de la página correspondiente.
    public java.lang.String getDocumentWithCR() throws BipiException
          Retorna el contenido de la página correspondiente adicionando un retorno de
          línea por cada línea de la página.
    public java.lang.String getDocumentWithRobotsFormat() throws BipiException
          Retorna el contenido del archivo Robots.txt del sitio correspondiente.
    public boolean ping()
          Verifica si la página es accesible.
    public final void url(java.lang.String url)
          Retorna la URL de la página a recuperar de la Web.
```

Class ar.com.bipi.types.WordKey

```
    java.lang.Object
        |
        +----ar.com.bipi.types.WordKey

public class WordKey extends java.lang.Object
```

Encapsula la estructura de una palabra dentro de una frase de búsqueda.

```
Variables:
    private java.lang.String operador
    private java.lang.String word

Constructor:
    public WordKey(java.lang.String ope, java.lang.String word)
          Constructor que setea el operador y la palabra.

Métodos públicos:
    public java.lang.String operador()
          Retorna el operador.
    public void operador(java.lang.String ope)
          Setea el operador.
    public java.lang.String word()
          Retorna la palabra.
    public void word(java.lang.String word)
          Setea la palabra.
```

Bibliografía

[Bon88] L. Gasser, A.H. Bond. *Readings in Distributed Artificial Intelligence.* Morgan Kaufmann Publishers, Los Altos, CA, 1988.

[Etz96] O. Etzioni. Moving Up the Information Food Chain: Deploying Softbots on the World Wide Web. *Appears in AAAI-96*, 1996.

[Fik92] Michael R. Genesereth, Richard E. Fikes. *Knowledge Interface Format. Reference Manual.* Logic Group, Junio 1992.

[Gra96] S. Franklin, A. Graesser. Is it an agent, or just a program? A taxonomy for autonomous agents, 1996. http://www.msci.members.edu/~franklin.

[Ket94] N. Genesereth, S. Ketchpel. Software Agents. *Communications of the ACM*, 1994.

[Kos95] Martijn Koster. Robots in the Web: threat or treat? Proceedings of the ACM CHI'96 Conference, Abril 1995. http://info.webcrawler.com/mak/projects/robots/robots.html.

[Lab99] R. Gómez Labrador. Agentes Móviles y CORBA. *Departamento de Lenguajes y Sistemas Informáticos - Universidad de Sevilla. Ramon.gomez@fie.us.es*, Febrero 1999. http://www.fie.us.es/~ramon/tesis/CORBA/Seminario-MASIF/.

[Liea] Henry Lieberman. Autonomous Interface Agents.

[Lieb] Henry Lieberman. Letizia: An Agent That Assists Web Browsing.

[Mae99] J. Pavón Maestras. Agentes Móviles. *Departamento de Sistemas Informáticos y Programación - Universidad Complutense de Madrid. Jpavon@sip.ucm.es*, 1999. http://pcdec.dec.usc.es/SEID99/pavon.html.

[Mar99] Evangelos P. Markatos. On Caching Search Engine Results. Technical Report 241, Institute of Computer Science (ICS), Enero 1999. http://www.ccsf.caltch.edu/~markatos/ arch-vlsi/html_papers/TR241/paper.html.

[McE] Tim Finin, Rich Fritzon, Don McKay, Robin McEntire. KQML - A Language and Protocol for Knowledge and Information Exchange. Technical Report CS-94-02, Computer Science Department, University of Maryland, UMBC. http:// www.cs.umbc.edu/kqml/papers/kbkshtml/kbks.html.

[Mou] Alexandros Moukas. Amalthea: Information Discovery and Filtering using a Multiagent Evolving Ecosystem.

[Nwa96] H.S. Nwana. Software Agents: An Overview, 1996. http://www.labs.bt.com/projects/agents/ publish/papers/.

[Paz96] Brian Starr, Mark S. Ackerman, Michael Pazzani. Do-I-Care: A Collaborative Web Agent. Proceedings of the ACM CHI'96 Conference, Abril 1996. http:// www.ics.uci.edu/CORPS/dica.html.

[Ray] D. Benech, T. Desprats, Y. Raynaud. A KQML-CORBA based Architecture for Intelligent Agents Communication in Cooperative Service and Network Management. *IRIT/SIERA, Université Paul Sabatier.*

[Viv] Henry Lieberman, Neil W. Van Dyke, Adriana S. Vivacqua. Let's Browse: A Collaborative Web Browsing Agent.

[Wel94] O. Etzioni, D. Weld. A Softbot-Based Interface to the Internet. *Appears in Communications of ACM*, Julio 1994.

[Wel95a] O. Etzioni, D. Weld. Intelligent Agent On The Internet: Fact, Fiction, and Forecast. *Departament of Computer Science and Engineering - University of Washington*, Mayo 30 1995.

[Wel95b] R. Doorenbos, O. Etzioni, D. Weld. A Scalable Comparison-Shopping Agent for the World Wide Web. *Departament of Computer Science and Engineering - University of Washington*, 1995.

[Woo96] R. Beale, A. Wood. Agent-Based Interaction. *School of Computer Science - University of Birmingham*, 1996.

[Wu] Daniel Siaw Weng Ngu, Xindong Wu. SiteHelper: A Localized Agent that Helps Incremental Exploration of the World Wide Web. Department of Software Deployment, Monash University. http://www.scope.gmd.de/info/www6/technical/paper068/ paper68.html.