Active Finder

Aplicación inteligente para la búsqueda online y en background de diversos tipos de Eventos en la Web

Licenciatura en Informática

Facultad de Informática

Universidad Nacional de La Plata

TIPO DE INVESTIGACIÓN

INVESTIGACIÓN APLICADA

Alumnos:

Arias, Oscar Pedro N° Alumno 1249/2

Juamperez, Leandro Ariel N° Alumno 1701/1

Directores de Tesis:

Lic. Javier Díaz

Lic. Ivana Harari

PREFACIO

El nacimiento de Internet significó un cambio revolucionario que en nuestros tiempos es comparado por su magnitud como la revolución industrial de la década del 30' u otros eventos que marcaron un antes y un después en la historia del hombre. Por primera vez se pudo acceder a un volumen de información prácticamente infinita para la capacidad de consumo del ser humano. Esto representó una ventaja y un inconveniente a la vez. Por un lado, cualquier usuario se beneficia con la existencia de toda esta información. Por el otro, se hace muy complicado organizar y clasificar la información de forma tal que sea legible y útil.

En un ámbito caótico como la red, la información buscada podría no estar disponible en el momento de efectuar la búsqueda, pero si un momento después. De manera que seria deseable tener algún tipo de proceso automático que nos avise en el momento en que surja o se encuentre disponible esta información.

Dentro de la ciencia de la información, un nuevo tipo de tecnología surge cada vez con mayor fuerza. Sus componentes son denominados ***agentes***.

Al plantearnos un desarrollo para nuestra tesis de grado, se nos enfrentó con el desafío de cómo encontrar información dentro del caos de datos lo más sintética posible y que esté lo más cercana posible al resultado que desea el usuario.

Nos pusimos entonces como objetivo primordial la utilizar software de desarrollo de aplicaciones basadas en agentes para poder desarrollar una aplicación amigable de búsqueda de información. Hicimos que los agentes ayudaran al usuario a organizar búsquedas de información de acuerdo a gustos del usuario, las programara y esperara los resultados. Además, los agentes fueron programados para aprender del usuario, haciendo más fácil las búsquedas posteriores.

A pesar de que nos planteamos una tesis aplicada, que hace escala en muchas tecnologías, eso no implica que la aplicación terminada sea perfecta o utilice las mejores

técnicas conocidas, lo que sobre todo buscamos con esta tesis era demostrar que se podía hacer, usando las tecnologías existentes, y que el aplicativo terminado tenía bases que pudieran llevarse a la práctica, aunque en la actualidad eso no es posible.

ÍNDICE

PREFACIO 2
ÍNDICE 4
1. AGENTES. 6
1.1 Agentes 6
1.2 Agentes de Software 7
1.3 Características: 7
1.4 Propiedades y atributos: 8
1.5 Clasificación de Agentes 9
1.5.1 Agentes de Interfase 10
1.5.2 Agentes Colaborativos 10
1.5.3 Agentes Móviles. 10
1.5.4 Agentes de Recuperación de Información 12
1.6 Agentes de software Inteligentes 13
1.7 Software para la construcción de agentes 15
1.7.1 AgentTalk 15
1.7.2 AgentBuilder 16
1.7.3 Agentx 16
1.7.4 DirectIA SDK 17
1.7.5 Gossip 17
1.7.6 Aglets 17
1.7.7 Grasshopper (IKV++) 18
1.7.8 iGEN 18
1.7.9 Intelligent Agent Factory 18
1.7.10 JACK Intelligent Agents 18
1.7.11 JAM 19
2. AGLETS 20
2.1 Aglets y Applets 20
2.2 Ciclo de Vida 21
2.3 Serialización 22
2.4 El API de Aglets 23
2.4.1 com.ibm.aglet.Aglet 24
2.4.2 com.ibm.aglet.AgletID 24
2.4.3 com.ibm.aglet.AgletProxy 25
2.4.4 com.ibm.aglet.AgletContext 26
2.4.5 com.ibm.aglet.Message 26
2.4.6 com.ibm.aglet.Ticket 26
2.5 Escribiendo un Aglet 27
2.6 El modelo de callback 27
2.7 Interacción entre el Aglet y el host 28
2.8 Interacción entre Aglets 28
2.9 Seguridad 30
2.10 Construyendo aplicaciones con Aglets 30
2.10.1 Escenarios 30
3. BUSCADORES 32
4. MOTORES DE BÚSQUEDA 33
4.1 Componentes 33
4.2 Indexación y almacenamiento 34
4.2.1 Indexación a nivel Submorfológico 34
4.2.2 Indexación por palabra clave 35
4.2.3 Indexación por conceptos. 36
4.2.4 Indexación por Hiperenlaces: 37

4.3 Tipos de indexadores 40
4.3.1 Manual 40
4.3.2 Robots/Crawlers 40
4.3.3 Metabuscadores 41
5. SISTEMAS DE RECOMENDACIÓN 42
5.1 Modelado de las necesidades de información del usuario 44
5.2 Mecanismos de recomendación 44
5.2.1 Recomendación basada en contenido 44
5.2.2 Recomendación colaborativa 47
5.3 Sistemas de Recomendación enfocados a Internet 49
5.3.1 FAB [Balavanovic 1998a] 50
5.3.2 Science Forum 50
6. SISTEMAS DE CALIFICACION: PICS 51
6.1 Descripción 51
6.2 Arquitectura PICS 53
6.3 Formato General 54
7. ACTIVE FINDER - DISEÑO E IMPLEMENTACIÓN 56
7.1 Historia del desarrollo 56
7.2 Diseño y modelo propuesto del aplicativo. 57
8. DISEÑO FUNCIONAL 59
8.1 Cliente: 59
8.1.1 Servicios requeridos: 59
8.2 Servidor de Búsqueda: 60
8.2.1 Servicios prestados: 60
8.3 Servidor de Base de Datos: 60
8.3.1 Servicios prestados: 61
8.4 Servidor de Indexación: 61
8.5 Servidor de Adquisición de datos: 61
8.6 Middleware 61
9. DISEÑO TÉCNICO. 63
9.1 Método de calificación 63
9.2 Tag ActiveFinder 64
9.2.1 Categorías 65
9.3 Detalle técnico 66
9.3.1 Servidor de adquisición de datos – Spider – 66
9.3.2 Indexador 66
9.3.3 Cliente y Servidor de Búsqueda 67
9.3.4 Servidor de Datos 71
9.3.5 Interfaz del monitoreo de Servidores y Clientes en línea. 72
9.3.6 Diagrama técnico de interacción de los servicios y el cliente 73
10. COMPARACIÓN CON OTROS BUSCADORES 74
11. REFERENCIAS 77

1. AGENTES.

1.1 Agentes

Una definición general dice que un agente es un ente que posee la habilidad, capacidad y autorización para actuar en nombre de otro. Los agentes deben poseer las características de *delegación, competencia y adaptabilidad.*

- **Delegación** es la capacidad para actuar discreta e independientemente en nombre de un tercero, cliente o usuario. Esta acción incluye tomar decisiones, emplear recursos y realizar tareas persiguiendo el objetivo.

- **Competencia** tiene que ver con la capacidad para llevar a cabo las tareas que se le asignaron dentro del entorno en el que se mueve. También incluye la capacidad de comunicarse con otros de modo eficiente.

- **Adaptabilidad** tiene que ver la capacidad de adaptarse a las condiciones cambiantes del medio donde se mueve en pro de realizar la tarea encomendada.

Ejemplos en la realidad humana serían los de un obrero que construye una pared siguiendo las instrucciones del arquitecto, una secretaria que organiza la agenda de su jefe, un contador que lleva los libros de una empresa, etc.

1.2 Agentes de Software

Como dijimos, los agentes son capaces de llevar a cabo tareas para las cuales fueron entrenados. El usuario delega en un agente la realización de una tarea (a menudo reiterativa o compleja) y se queda esperando el resultado. Si bien existen varias definiciones de lo que significa agente, actualmente no existe un consenso que unifique estas definiciones en una concreta. Wooldridge y Jennings definen agentes como un programa auto-contenido capaz de controlar su proceso de toma de decisiones y de actuar, basado en la percepción de su ambiente en persecución de uno o varios objetivos. Nwana dice que un agente es un componente de software y o hardware capaz de actuar para ejecutar tareas en nombre de un usuario.

Otros investigadores agregan el concepto de autonomía a las características de un agente. En este caso, el agente habita en un universo complejo del que percibe estímulos y reacciona en forma autónoma para cumplir las tareas para las que fue creado.

Para Russel, un agente puede ser visto como algo que percibe su ambiente a través de sensores y actúa contra este a través de efectores.

1.3 Características:

En base a las definiciones anteriores podemos extraer características generales que los agentes deben poseer:

- Deben ser parte de un ambiente o entorno.
- Deben poder sensar su entorno
- Deben poder actuar sobre el entorno
- Deben responder a una serie de objetivos.

El entorno de un agente de software tiene que ver con los sistemas operativos, aplicaciones, bases de datos, redes y dominios virtuales.

Como en el caso de los agentes genéricos, podemos encontrar aquí las tres propiedades definidas:

- **Delegación** tiene que ver sobre todo con la persistencia. Persistencia que hace a la supervivencia. El usuario lo dispara y deja hacer.

- **Competencia** tiene que ver con conocimiento de los diversos protocolos existentes dentro del entorno, como los de red, base de datos, Http, llamadas a la API del sistema operativo subyacente, etc.

- **Adaptabilidad** tiene que ver en muchos casos con la capacidad para adaptarse a un cambio de sistema operativo o de las condiciones de software. En la mayoría de los casos, sobre todo en los agentes de software no inteligentes tiene que ver mas que nada con la posibilidad de detectar estos cambios y avisarlos para que se efectúe la actualización del agente en cuestión.

1.4 Propiedades y atributos:

Es después de analizar diferentes definiciones que hay para agentes, se pueden rescatar una serie de propiedades y atributos que los agentes deben satisfacer:

- **Autonomía:** actuar sin intervención directa de otro agente o persona. El agente lleva a cabo estrategias para llegar a una meta y analiza los resultados de la aplicación de estas. Puede tratar de cambiar de estrategia si percibe que no llega a los resultados esperados con la primera aplicada.

- **Habilidad Social:** Debe poder comunicarse con otros agentes o con otros seres humanos mediante un protocolo común, para poder comunicar los resultados obtenidos o interactuar con otros para llevar a cabo sus tareas.

- **Reactividad**: El agente debe poder sensar el ambiente en que se encuentra para reaccionar a los cambios en el mismo.

- **Orientación por objetivos**: El agente no solo reacciona al medio sino que también actúa en función de cumplir sus objetivos. Un agente busca satisfacer permanentemente su agenda interna de tareas.

- **Continuidad temporal**: El agente es un proceso temporalmente continuo. A diferencia de un programa convencional del cual se conoce su inicio y su fin, un agente debe ejecutarse hasta que se halla alcanzado el conjunto de objetivos solicitados o bien mientras su ciclo perdure o su usuario no desee detenerlo.

1.5 Clasificación de Agentes

Los agentes pueden clasificarse de varias maneras, teniendo en cuenta alguna de las propiedades que poseen o bien haciendo hincapié en alguna en particular. Se propone entonces un clasificación de acuerdo a las ramas de investigación y desarrollo de agentes:

- Agentes de Interfase
- Agentes Colaborativos
- Agentes Móviles
- Agentes de recuperación de información

1.5.1 Agentes de Interfase

Los agentes de interfase son agentes cuasi-inteligentes que interactúan entre el usuario y diversas aplicaciones actuando como asistentes personales. Ejemplo de ellos son algunos de los motores de búsqueda de información o alguna de las aplicaciones de filtrado de mensajes de correo electrónico. El agente observa y monitorea las tareas realizadas por el usuario, aprende atajos y sugiere mejores formas de hacer las tareas. El objetivo es que el agente pueda adaptarse a las preferencias y hábitos del usuario.

1.5.2 Agentes Colaborativos

Múltiples agentes colaboran e interactúan para satisfacer los requerimientos de los usuarios, deben tener para esto un protocolo de comunicación, poseer un alto grado de autonomía que resuelve las partes simples de una tarea compleja. El objetivo es desarrollar un sistema basado en un conjunto de componentes relativamente fáciles de codificar y mantener. Además un sistema más confiable y gracias a la posibilidad de ejecutar en paralelo estos agentes hacer más veloz la ejecución de la aplicación.

1.5.3 Agentes Móviles.

También conocidos como agentes viajeros, son agentes capaces de viajar por una red de computadores, moviendo su estado y su código, con capacidad de recolectar y procesar información y retornar a su origen. El modelo típico de programación con estos agentes es el cliente-servidor. El agente, en el rol de cliente, requiere un volumen de información al servidor, generalmente de base de datos, que luego el agente entrega a su vez al cliente. Estos agentes forman un nivel más de abstracción para el usuario, detrás del cual se enmascaran cuestiones técnicas complejas tales como la distribución; es decir como manejar recursos computacionales distribuidos. Este concepto oculta parcialmente la distribución de los recursos al usuario.

La noción de movilidad surge de la necesidad de reducir el tráfico de red al mínimo necesario. Esto reduce también los costos de comunicación y permite al usuario acceder, al mismo tiempo, a los recursos computacionales de toda la red.

Para ver la utilidad de un agente móvil pensemos en el siguiente ejemplo. Se posee una base de datos distribuida geográficamente en una Wan. Si el usuario deseará recuperar una determinada información, debería ejecutar la sentencia mediante la aplicación correspondiente. El usuario debería esperar hasta que la aplicación accediera a todas las bases y recuperara la información. Si nuestra aplicación contara con agentes móviles, lo que debería hacer el usuario es programar la operación de recuperación deseada, lanzar los agentes y dejar que estos se desplazaran a las diversas bases de datos para procesar la consulta localmente. Una vez terminada de procesar, los agentes volverían al origen con los resultados y los entregarían al usuario.

Si ahora analizamos esta misma posibilidad para un operación que efectúa un proceso intensivo de la información, quizás solo retornando los mensajes de resultado, notaremos fácilmente la potencia de los agentes móviles.

Pero soportar una arquitectura de agentes requiere brindar un Middleware que soporte el pasaje del código y el estado del agente. Se requiere que exista un "**medio ambiente**" o entorno donde los agentes puedan **"vivir"**. También se requiere de mecanismos que permitan que el entorno confíe en el agente en particular.

La construcción de sistemas basados en agentes móviles deben satisfacer algunas cuestiones fundamentales:

- **Transporte:** Se debe determinar como se ha de efectuar el traslado del agente y su código de un lugar a otro de la red.
- **Ejecución:** Como se ha de ejecutar el agente en un entorno remoto.
- **Autenticación:** Debe brindarse un método para asegurarse de que el agente es el que dice ser.
- **Privacidad:** Debe brindarse un método para asegurar que el agente pueda mantener a salvo su estado interno, evitando la intrusión dentro del mismo.

- **Seguridad:** Debe brindarse un mecanismo para evitar que el agente sea atacado por otros agentes, virus. A su vez brindando un mecanismo para recuperarlo de loops infinitos o fallas de sistema.

1.5.4 Agentes de Recuperación de Información

Este es un tipo especial de agente dedicado específicamente a la recuperación de información para el usuario. Dado que Internet brinda al usuario una cantidad inmensa de información, se da la paradoja de que el usuario ya no puede encontrar la información que necesita.

El crecimiento de la red presenta al usuario una cantidad de información distribuida, almacenada en forma muy heterogénea que la hace difícil de procesar. Se presenta el fenómeno denominado sobrecarga de la información. Este fenómeno no es del todo indeseado ya que brinda costados muy interesantes. Lo que se busca es lograr mecanismos para administrar esta sobrecarga sin eliminarla. Se pretende separar la información basura de la información relevante.

Hasta ahora, los mecanismos para tratar con la sobrecarga de la información se han basado en la construcción de *motores de búsqueda* y en mecanismos de indexado multi-nivel asociados con herramientas de interfase adecuadas - conocidos como *directorios*- cuya clasificación depende del autor/organizador del directorio. Ambos métodos tienen serias deficiencias. Los motores tienen una tendencia a recuperar demasiada información indeseable. Causa de esto es que los motores de búsqueda basan su funcionamiento en contar las ocurrencias de determinadas palabras claves buscadas y el dentro de los documentos. Los directorios tienden a no recuperar todas las palabras claves dado que un indexado en profundidad es muy costoso en tiempo y tamaño.

La idea tras esto es crear índices personalizados de alta calidad para la recuperación de información para el usuario. Una solución es usar agentes de recuperación. Estos colaboran con el usuario novato guiándolo en el proceso de definición de la consulta dándole guías para la definición de consultas avanzadas. Además el agente puede acceder a medios heterogéneos ubicados en lugares dispares.

Los agentes de recuperación poseen mecanismos para acceder y recuperar la información en forma rápida y eficiente. Tienen la función de administrar, y efectuar fácilmente consultas avanzadas. Permiten administrar, manipular y juntar información de medios heterogéneos distribuidos. Pueden tener funcionalidades de búsqueda y navegación flexibles y algoritmos de clasificación poderosos.

A la larga lo que se busca es construir agentes capaces de conocer al usuario de forma de armar un diario de búsquedas que pueda ejecutarse repetidamente.

1.6 Agentes de software Inteligentes

Son agentes de software que usan mecanismos de inteligencia artificial para lograr los objetivos perseguidos por el usuario.

La Inteligencia artificial consiste en imitar los mecanismos de razonamiento humanos mediante mecanismos computacionales. Los clientes pueden reducir su trabajo delegando tareas que normalmente realizarían ellos en Agentes Inteligentes.

De esta manera, muchos investigadores que iniciaron sus trabajos en IA, actualmente trabajan con agentes de software inteligentes. Así es como también al hablar de agentes se implica de alguna manera a los agentes de software inteligentes. Las características que normalmente se aplican a los agentes afectan también a los agentes inteligentes:

Delegación es un termino más absoluto para los Agentes de Software Inteligentes. Dadas las características de IA que poseen estos agentes, el usuario puede, o debería poder, delegar mas funciones en el agente. Este debería poder actuar mejor y generar una mayor respuesta de confianza por parte del usuario.

Competencia es potenciada por la capacidad de los Agentes de Software Inteligentes de mezclar funciones de alto orden con sus capacidades intrínsecas. Los Agentes de Software Inteligentes pueden a menudo analizar la información cruda para obtener nuevos y ocultos patrones de conocimiento de esta. Asimismo, los Agentes de Software Inteligentes pueden también extraer conocimientos de otros componentes de su entorno.

Adaptabilidad tiene que poder combinar las características de auto monitoreo del agente con las de su capacidad de aprendizaje para obtener mejor perfomance a medida que el tiempo pasa. Además de esto debemos agregar que el agente podría usar técnicas de inferencias que le permitan entender cada vez mejor al cliente. Incluso al punto de saber que es lo que el cliente quiere aun mejor que lo que el cliente mismo sabe que quiere.

El dominio de aplicación para Agentes de Software Inteligentes incluye aplicaciones donde el pensamiento parecido al humano que estos agentes pueden llevar a cabo es requerido para operar de forma autónoma.

La lista de estas aplicaciones involucra algunas de las aplicaciones más difíciles de llevar a cabo y que caen dentro del ambiente de la Inteligencia Artificial:

- Inferencia y deducción lógica
- Reconocimiento contextual de dominios
- Reconocimiento de patrones
- Aprendizaje y adaptabilidad.

Una vez que la inteligencia artificial es programada, los Agentes de Software Inteligentes pueden procesar volúmenes de datos increíblemente superiores a los que los humanos podrían procesar. Además, los Agentes de Software Inteligentes pueden analizar y reaccionar a cambios en su entorno de una manera mucho más veloz que lo que cualquier humano podría hacer.

1.7 Software para la construcción de agentes

La tecnología de agentes evoluciona rápidamente, en la actualidad existen numerosas herramientas para la construcción de agentes de software. En este capitulo solo mencionaremos algunos de ellos:

1.7.1 AgentTalk

AgentTalk, de NTT and Ishida es un lenguaje de descripción de protocolos para sistemas multiagente. A medida que la aplicación de agentes en el área de la inteligencia artificial crece, surge la necesidad de tener protocolos que coordinen a los agentes. AgentTalk permite definir y configurar estos protocolos y extenderlos con una cierta facilidad aplicando mecanismos de herencia. AgentTalk es una colaboración entre la Laboratorio de Ciencias de la Comunicación NTT y el Laboratorio Ishida, del gobierno de Japón, esta desarrollado en **Common Lisp** y solo se distribuye en Japón.

1.7.2 AgentBuilder

Esta es una suite de herramientas para la construcción de agentes inteligentes. AgentBuilder consiste de dos componentes principales: el Toolkit y el sistema runtime. El primero brinda una serie de herramientas para el control del proceso de desarrollo de software basado en agentes, desde el análisis del dominio de los agentes, diseñar y desarrollar redes de comunicación entre agentes, definir conductas y depurar la aplicación así desarrollada. El software runtime consta de un motor de agentes que permite crear un entorno donde estos operen. AgentBuilder usa como protocolo de comunicación el conocido KQML (Knowledge Query and Manipulation Language), soportando las definiciones de este protocolo, además, permite definir sub-lenguajes adaptados a las necesidades de la aplicación.

Todos los componentes de AgentBuilder están desarrollados usando el lenguaje Java, por lo que una aplicación construida en este lenguaje puede correr en cualquier máquina que ejecute Java.

1.7.3 Agentx

Agentx es un lenguaje de segunda generación desarrollado como un conjunto de librerías poderosas, rápidas y escalables. Estas librerías proveen funcionalidad ORB (Object Request Broker) más rápida, fácil, compacta y altamente funcional que reemplace a las librerías RMI que vienen con el SDK de Sun. Agentx no requiere de un IDL ni de definir un Stub. Además es totalmente compatible con las maquinas virtuales de Sun y de Microsoft. Además de esto, Agentx brinda un soporte para la programación y liberación de agentes de software autónomos. Los agentes se pueden mover por la red, disparar la ejecución de otros agentes independientemente. Esto permite que software de gran complejidad pueda ser roto en pequeñas unidades independientes. Por estar desarrollado en Java, las aplicaciones desarrolladas con Agentx pueden ejecutar en cualquier dispositivo con soporte para Java. Una aplicación desarrollada con Agentx puede ser usada para simular las funcionalidades de un gran mainframe sobre una red de computadoras personales.

1.7.4 DirectIA SDK

DirectIA (Direct Intelligent adaptation) es un SDK de alto nivel que permite que un agente o grupo de agentes puedan aprender, anticipar y seleccionar sus acciones. Un agente controlado con este motor es capaz de adaptarse a entornos dinámicos, descubrir nuevas conductas y estudiar su eficiencia en tiempo real. Detrás de este motor se encuentra una tecnología llamada biomimética con un desarrollo de más de diez años. Se basa en tecnología de vida artificial propietaria muy distinta de los demás desarrollos en IA. DirectIA brinda las conductas típicas de un agente más las habilidades que son de esperar en oponentes inteligentes. Se brinda como una librería en C++ para plataformas Windows. Este software está destinado a desarrollo de juegos, simuladores, aplicaciones de entrenamiento militares, avatares y web bots.

1.7.5 Gossip

Gossip es una aplicación basada en agentes móviles que comercian información electrónica para el usuario en la Internet. El usuario indica que información de interés desea al agente y este se mueve a un Server en la Internet llamado el **Mercado**. En el mercado, el agente Gossip busca otros agentes con intereses similares. Cuando estos agentes son encontrados, se reúnen e intercambian información.

1.7.6 Aglets

Un Aglet es un objeto Java que puede moverse entre un Host en la Internet y otro. Un Aglet que ejecuta en un host determinado puede, repentinamente, suspender su ejecución, despacharse hacia otro host remoto y resumir su ejecución allí. Cuando el Aglet se mueve, toma todo su código y su estado (datos) y los mueve con él. También provee un mecanismo de seguridad que evita que Aglets no confiables ingresen a un entorno. Dado que la tesis hará un desarrollo basándose en esta tecnología, hablaremos más detalladamente de ellos más adelante.

1.7.7 Grasshopper (IKV++)

Grasshopper es el primer entorno para agentes móviles que cumple con el estándar MASIF (Mobile Agent System Interoperability). Este estándar esta montado sobre CORBA, proveyendo una integración entre el paradigma de programación Cliente-Servidor y la tecnología de agentes móviles. La estandarización asegura que la misma aplicación de agente pueda ejecutar en diferentes entornos. Grasshopper permite el desarrollo de aplicaciones distribuidas basadas en agentes móviles que toman ventaja de la velocidad de las redes locales.

1.7.8 iGEN

Este es un Toolkit y Workbench basado en un modelo programable de experiencias humanas. Incluye herramientas de alto nivel para el desarrollo de aplicaciones inteligentes.

1.7.9 Intelligent Agent Factory

Intelligent Agent Factory permite reducir el tiempo necesario para desarrollar soluciones basadas en agentes. La inteligencia de los agentes se basan en reglas desarrolladas en Jess (CLISP), un sistema de reglas encadenadas. Los agentes y las reglas son generados a partir de especificaciones simples de workflow. Existe un framework basado en KQML. La librería también permite soportar agentes móviles.

1.7.10 JACK Intelligent Agents

Es un sistema de agentes de tercera generación que provee la arquitectura y la capacidad para desarrollar y ejecutar aplicaciones distribuidas basadas en agentes de software. Esta escrito en Java y se programa usando JACK Agent Language, que es una extensión a Java y que por ello provee los beneficios del lenguaje Java.

1.7.11 JAM

Es una arquitectura de agentes creencias-deseos-intenciones basada en el Sistema de Razonamiento Procedural (PRS) de Georgeff, Ingrand, Rao, Lansky y otros. A diferencia de otros agentes disponibles hoy en día, que son utiles en dominios de aplicación restringidos, JAM es aplicable a cualquier dominio de aplicación. Soporta top-down y razonamiento basado en objetivos y además bottom-up y razonamiento orientado a los datos.

2. AGLETS

Los Aglets son agentes de software que pueden detener su propia ejecución, enviarse a otra computadora en la red – Intranet o Internet -, y continuar su ejecución desde el punto en que habían dejado suspendido.

Un ejemplo sencillo: supongamos tener un Aglet que tiene como estado interno un contador que incrementa en uno su valor hasta llegar a diez. Hecho esto, suspende su ejecución y se desplaza a otro host en la red. Allí resume su actividad incrementando su valor. Si este objeto fuera un programa tradicional, la cuenta reiniciaría desde uno, en el caso de un Aglet este continuará desde donde dejo, incrementando su valor a once.

Los Aglets son agentes autónomos ya que ellos pueden decidir donde van a ir y que van a hacer. Ellos controlan su tiempo de vida. Cada Aglet puede recibir peticiones de fuentes externas, tales como otros agentes, y son capaces de decidir si contestarán o no ese requerimiento. También pueden decidir que acciones pueden llevar a cabo, independientemente de los requerimientos externos.

2.1 Aglets y Applets

Los Aglets Java extienden el modelo de aplicación de red móvil de Java, los conocidos applets. Como los applets, el código de un Aglet puede migrar a través de la red. Pero, a diferencia de estos, cuando el Aglet migra, también se lleva su estado. Un Applet es código que puede migrar a través de la red desde un servidor a un cliente. Un Aglet es un programa java que está ejecutando (con su código y su estado de variables) que se puede mover de un host a otro dentro de una red, cambiando de destino de acuerdo a algún criterio determinado, pudiendo volver a su host origen.

Un Aglet, al igual que cualquier Applet puede ejecutar dentro de uno o varios threads dentro de un host que ejecuta Java. Para ejecutar un Applet, un navegador Web dispara la ejecución de una aplicación Java la cual carga los Applets necesarios y que además establece un mecanismo de seguridad para restringir la actividad de los Applets no confiables. Para cargar los archivos de clase de los Applets, se crea un cargador de clases que sabe como solicitar las clases al servidor HTTP.

De un modo similar, un Aglet necesita de una aplicación Java que actúa como host y que debe estar ejecutándose antes de que el Aglet pueda visitar ese host. Cuando un Aglet migra lo hace entre hosts de Aglets en maquinas diferentes. Como en el caso de los Applets, los hosts Aglets instalan un manejador de seguridad para regular la actividad de los Aglets no confiables. Por otro lado, crean también un cargador de clases que saben como recuperar los archivos de clase y el estado del Aglet desde un host de Aglets remoto.

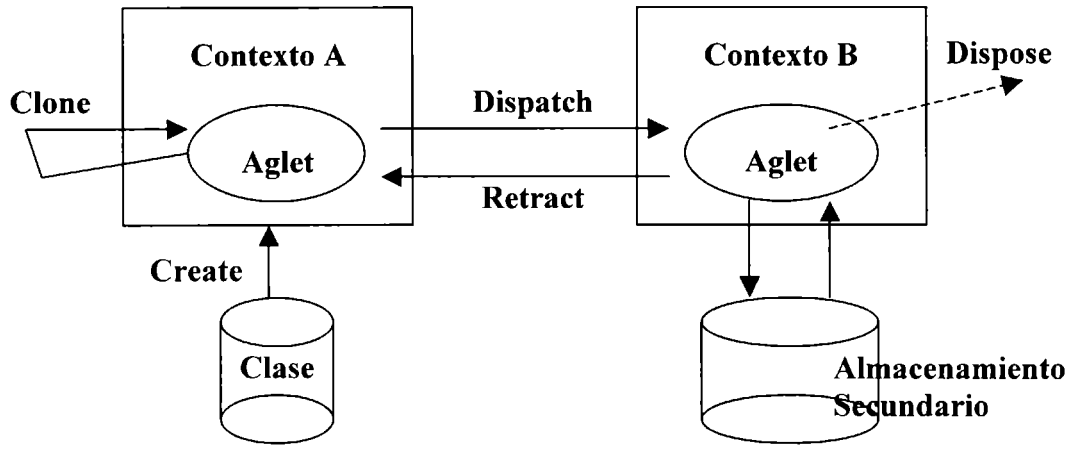
2.2 Ciclo de Vida

Un Aglet puede experimentar muchos eventos en su vida. Puede ser:

- **Creado:** un nuevo Aglet nace, su estado es inicializado y comienza a ejecutar.
- **Clonado:** se crea un Aglet gemelo del primero en todo igual al original.
- **Despachado:** el Aglet se mueve a un nuevo host.
- **Retraído:** el Aglet, previamente despachado, es traído de nuevo al host origen.
- **Desactivado:** se envía al Aglet a "dormir", es decir, se almacena su estado en disco en alguna parte y se lo remueve de la memoria.

Activado: un Aglet desactivado es vuelto a la vida, es decir, se restaura su estado desde el disco.

Destruido: el Aglet muere, su estado muere con el.



Esquema de Ciclo de vida de un Aglet

2.3 Serialización

Toda operación del Aglet que involucre movimiento entre hosts o hacia un almacenamiento involucra **serialización**.

Los hosts Aglets utilizan la serialización para exportar el estado del Aglet a un Stream de bytes que es lo que se transfiere entre hosts en la red. A través de este proceso, el Aglet y todos los objetos internos serializables son escritos en un Stream. Como sabemos, un objeto es serializable si implementa las interfaces *Serializable* o *Externalizable*. En el otro extremo del camino el Aglet es reconstruido desde un Stream de bytes. La serialización permite guardar la Heap del proceso y luego reconstruirlo a partir del mismo Stream. Sin embargo, la serialización no alcanza las pilas ni los contadores de programas de los threads involucrados en el Aglet. Por ello, cuando el Aglet es despachado, clonado, desactivado, etc., toda información relevante contenido en la pila o en los contadores de programa será perdido.

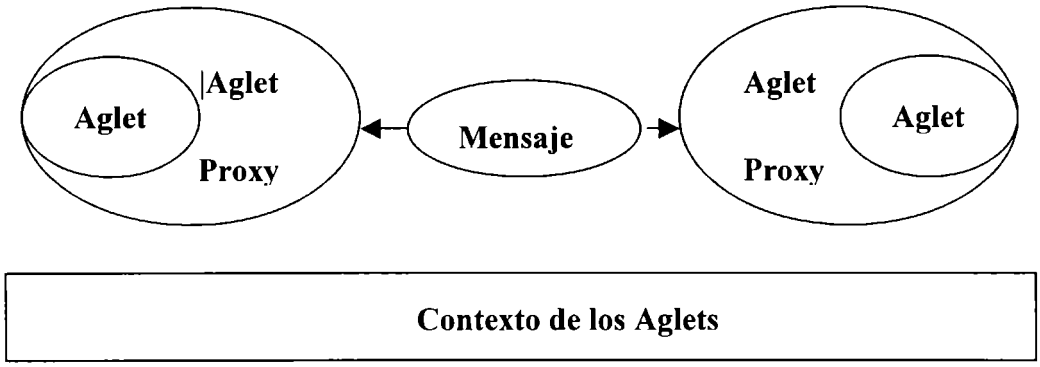
Aunque en teoría un Aglet al migrar debería poder llevarse **todo** su estado, incluido pilas de ejecución y contadores de programa, en la practica, esta posibilidad esta limitada por el modelo de seguridad subyacente en la propia JVM que es la que controla que ningún proceso Java pueda acceder directamente a su estado interno de ejecución.

Antes de ser serializado, un Aglet debe poder guardar todo su estado en la Heap de modo de poder ser resucitado correctamente. No puede dejar nada en la pila de ejecución dado que esta no podrá ser reproducido dentro de otro entorno. Es por ello que el host notifica al Aglet que ha de ser serializado, de manera que este pueda prepararse.

De todas maneras, la necesidad de guardar estados de contadores y pilas de ejecución no es tan acuciante como la de finalizar algún proceso pendiente antes de ser serializado. El Aglet puede, al recibir el aviso correspondiente, rechazar la orden o hacer cualquier otra cosa.

2.4 El API de Aglets

La API de Aglets define las principales funcionalidades de estos agentes. Se pueden observar los principales elementos de esta API en el siguiente diagrama:



Esta es una relación de las clases mas importantes:

2.4.1 com.ibm.aglet.Aglet

Esta clase Abstracta define los métodos fundamentales usados para controlar la movilidad y el ciclo de vida de un agente móvil. Para definir un agente móvil usando Aglets, se debe extender esta clase.

La clase Aglet es también usada para acceder a atributos asociados con un aglet. Por ejemplo, el objeto com.ibm.aglet.AgletInfo se puede obtener invocando al método primitivo Aglet.getAgletInfo().

2.4.2 com.ibm.aglet.AgletID

Toda instancia de un Aglet tiene un identificador unívoco que es inmutable a través de toda la vida del Aglet. Ese identificador no es solo un número, sino que contiene una serie de atributos como el ID. del usuario, el tipo de agente, el sistema de origen. El AgletID es un objeto que mantiene el identificador del Aglet, a la vez que encapsula su representación interna.

2.4.3 com.ibm.aglet.AgletProxy

Este objeto interfaz representa un handle para acceder al objeto Aglet al que representa. Dado que un Aglet no puede acceder a los métodos definidos en otro por razones de seguridad – muchos métodos críticos de control de un Aglet son definidos públicos -, el primero debe obtener, primero, el AgletProxy que representa al segundo y luego usar esta interfaz como medio para interactuar. En otras palabras, el AgletProxy, actúa como un escudo para evitar que Aglets maliciosos afecten a otros Aglets.

Cuando se invoca desde un Aglet al proxy del otro utilizando el método correspondiente, se consulta al servidor de seguridad para ver si el invocador tiene derecho a recibir el proxy reclamado.

La otra cuestión es la referida a la transparencia de localización. Si el Aglet reside en un host remoto, o en el mismo host que el invocador, el AgletProxy se encarga de enviarle los mensajes.

El AgletProxy se puede obtener de una cierta cantidad de maneras:

- Mediante los métodos getAgletProxy() y getAgletProxies()
- Vía el AgletID, el cual es pasado como argumento al método AgletContext.getAgletProxy(AgletID) o Aglets.getAgletProxy(String Contexto, AgletID).

Recuperando el AgletProxy de un mensaje. Se puede colocar un AgletProxy como argumento de un objeto Message para luego pasarlo por referencia. Poniendo el AgletProxy como una propiedad del Contexto. La runtime de Aglets se encarga de implementar esta interfase por lo que el usuario no debe implementar nada.

2.4.4 com.ibm.aglet.AgletContext

Esta clase representa una interfaz con el entorno de ejecución donde se encuentra ubicado el Aglet. Cualquier Aglet puede acceder al contexto donde está ejecutando mediante el método Aglet.getAgletContext() y obtener información acerca del host donde ejecuta, el entorno, variables, crear nuevos Aglets en el contexto, etc. Si el Aglet es despachado, la referencia a su contexto es reemplazada por la del contexto adonde ha arribado.

2.4.5 com.ibm.aglet.Message

Los Aglets se comunican intercambiando objetos de la clase Message. El objeto Message tiene una propiedad – de tipo String - que especifica el tipo de mensaje; Un mensaje se puede enviar llamando a AgletProxy.sendMessage(), que envía un mensaje de tipo **sincrónico**, AgletProxy.sendAsyncMessage() que como su nombre indica, envía mensajes de tipo **asincrónico;** o AgletProxy.sendOnewayMessage(), mensajes sin acuse de recibo. Los mensajes recibidos son pasados como argumento por referencia al método Aglet.handleMessage() que es donde se codifica la respuesta a cada uno.

2.4.6 com.ibm.aglet.Ticket

Un objeto de la clase Ticket es usado para especificar el destino y las condiciones adonde ha de dirigirse el Aglet. El Ticket es una metáfora del pasaje que cualquier viajero puede tener, indicando modo de llegar y salida, camino y destino, define el protocolo y el tiempo de vencimiento de la sesión, el nivel de seguridad y confidencialidad, etc. Este objeto se crea cuando se especifica una URL como medio de enviar al Aglet.

2.5 Escribiendo un Aglet

Escribir un Aglet puede parecer en mucho a escribir un Applet. Primero se debe crear una clase que herede de **Aglet**. Se debe sobrescribir el método **onCreation()** para escribir código a ejecutarse durante la inicialización del Aglet. Este método es invocado solo una vez cuando el Aglet es creado.

También existe un método **run()** que es invocado cada vez que el Aglet es reiniciado en su ejecución, ya sea por haber arribado a un nuevo host o porque ha sido activado. Cada vez que un Aglet arriba a un host, se dispara un evento **onArrival()**.

Cuando se crea el Aglet, se dispara el evento **onCreation**(), inmediatamente después se llama al método **run**(). Cuando el Aglet llega a un nuevo host, **onArrival** () es disparado, seguido de **run** (). Esto demuestra la incapacidad de los Aglets para mantener su ejecución sin una cierta ayuda.

2.6 El modelo de callback

Antes de que alguno de los eventos mayores en la vida de los Aglets se lleve a cabo, algún método de callback es llamado, este método puede servir para efectuar un procesamiento antes de llevar a cabo el evento, o para rechazar el mismo. Por ejemplo, como dijimos antes, previo al despacho de un Aglet a otro host se llama al evento onDispatch(). Cuando se vuelva de este método, el objeto empezará su proceso de serialización. onDispatch() es un método callback ya que el host invoca este método un momento antes de que se dispare el método dispatch(). El modelo de callback es muy usado en Java, y Aglet hace uso del mismo.

La clase Aglet define cinco métodos callback que pueden ser sobrecargados para redefinir la conducta del aglet:

- **onCloning():** Se dispara previo a una operación de clonado del Aglet.
- **onDispatch():** se dispara previo al envío del Aglet a otro host.

- **onReverting():** se dispara antes de retraer el Aglet.
- **onDeactivating():** antes de desactivar el Aglet.
- **onDisposing():** antes de destruir el Aglet.

Para cada uno de estos procesos, la clase Aglet tiene un método asociado que dispara la acción correspondiente: clone(), dispatch(), retract(), deactivate(), dispose(). Cada vez que un Aglet inicia su ejecución en un host, se disparan una serie de métodos de inicialización en el Aglet. Cuando se efectuó la inicialización se dispara el método run(). De acuerdo al evento que precipito el reinicio de la vida del Aglet el host usará alguno de estos métodos para llamar al Aglet:

- **onCreation():** se dispara la primera vez que se crea al Aglet.
- **onClone():** se dispara cuando se procede a clonar al Aglet.
- **onArrival():** llamado después de un dispatch() o retract()
- **onActivation():** llamado después de una activación.

2.7 Interacción entre el Aglet y el host

Un Aglet interactúa con su entorno a través de un objeto llamado **AgletContext**. El Aglet obtiene el andel de su contexto invocando al método **getAgletContext()**. El AgletContext tiene eventos que permiten crear o destruir Aglets, obtener un **AgletProxy** a los mismos, etc.

2.8 Interacción entre Aglets

La interacción entre Aglets, no se lleva a cabo invocando el uno directamente los métodos del otro. En lugar de eso, un Aglet que desea comunicarse con otro obtiene el AgletProxy de este. Un AgletProxy es una especie de "puntero a" un Aglet.

La clase AgletProxy contiene métodos que permiten a los Aglets solicitarse entre si servicios tales como dispatch(), clone(), deactivate(), dispose(). El Aglet que recibe el pedido puede aceptarlo o negarse a llevarlo a cabo. Por intermedio de un AgletProxy se pueden enviar *mensajes* al Aglet al que cada Proxy representa. Cada mensaje contiene un pequeño objeto de la clase **Message**. Cada objeto de esta clase posee un nombre/tipo de mensaje, más una serie de objetos adicionales que componen el contenido del mensaje. Para enviar un mensaje a otro Aglet, el remitente debe crear un objeto de la clase Message, asignar los parámetros y enviarlos ya sea en forma de mensaje sincrónico con sendMessage(), o asincrónico con sendAsyncMessage().

Los Aglets deben interactuar solo mediante AgletProxys, aun cuando ambos estén en el mismo host. Esto es en parte, dado que todos los métodos callback y de inicialización de un Aglet son definidos públicos. Si se pudiera acceder directamente al Aglet, otro Aglet torpe o malicioso podría invocar métodos indebidos en ese Aglet.

El AgletProxy puede apuntar a un Aglet local o remoto, pero el proxy es siempre local. Si un Aglet montado sobre el host **A** desea comunicarse con un Aglet sobre el host **B**, entonces debe obtener un proxy local que referencie al Aglet remoto. Una vez obtenido el AgletProxy, la comunicación es transparente de la localización del Aglet "físico".

Se puede obtener el proxy a un Aglet por uno de estas tres vías:

- Creando el Aglet usando createAglet(), lo que retorna el proxy al Aglet recién creado.
- Buscando mediante la función getAgletProxies() que retorna una enumeración de los proxies dentro de un contexto.
- Proveyendo el **AgletID** a la función getAgletProxy(). El AgletID es un identificador univoco para cada Aglet.

2.9 Seguridad

Sistemas basados en agentes móviles requieren de un alto grado de seguridad, ya que se podrían convertir en un camino para enviar programas peligrosos. Para una difusión de los Aglets se debe brindar una gran infraestructura de hosts **seguros**, que eviten que Aglets dañinos invadan y causen daño, a la vez que provean acceso a los recursos para los Aglets confiables. Con los desarrollos actuales, ataques como los de saturación de memoria (denial of service) son todavía posibles.

2.10 Construyendo aplicaciones con Aglets

La librería Aglets provee a los desarrolladores con un conjunto de APIs para construir aplicaciones. Usando estas APIs se puede construir una aplicación que contenga un visor, una interfase centralizada o simplemente lanzar un Aglet sin tener ninguna capacidad servidora.

2.10.1 Escenarios

Existen varios escenarios para la construcción de aplicaciones Aglets. Un servidor que actué como host para la ejecución de Aglets; un cliente que pueda crear y controlar remotamente un Aglet; Acceso a Aglets a través del navegador Web.

2.10.1.1 Aplicaciones basadas solo en Aglets

En este caso, no se requiere API externa alguna. La aplicación consiste en un conjunto de Aglets que interactúan. Normalmente existirá un Aglet estacionado con derechos de acceso a algún recurso local y que brinda a los otros Aglets permisos de acceso a los mismos.

Los Aglets que llegan a este host requieren servicios del Aglet privilegiado por medio de mensajes. Los Aglets clientes pueden entonces enviar los resultados por medio de mensajes remotos o moviéndose a donde se encuentra el solicitante.

El Aglet puede ser configurado con restricciones de acceso a los recursos. Esto permitiría, por ejemplo, tener una sola conexión a una base de datos a la que accedería el Aglet privilegiado. Los otros Aglets efectuarían las consultas a través de llamados al Aglet privilegiado. Esto reduciría la cantidad de conexiones necesarias y los recursos asociados.

2.10.1.2 Visor de Aglets customizado

Un visor de un servidor de Aglets puede configurarse usando un contexto y la API de Listener asociada a un servidor de Aglets. Estos visores son herramientas muy útiles para administrar entornos y dar interfases especificas a los usuarios. Por medio de estas herramientas se pueden brindar servicios a los Aglets entrantes y permiten a los usuarios administrar en forma amigable su interacción con el Aglet.

2.10.1.3 Aplicaciones con capacidades servidoras embebidas.

Este escenario es típico modo de agregar funcionalidad a un servidor Aglet. Es muy útil para aplicaciones que deben actuar como hosts para un numero de Aglets. Un ejemplo típico es una aplicación groupware que permite administrar relaciones entre usuarios usando Aglets como avatares y helpers. En estas aplicaciones, un servidor es inadecuado para administrar las relaciones Aglet/usuario, Aglet/Aglet.

La API de servidor de Aglets permite embeber funcionalidades servidoras en la aplicación desarrollada, trapear los errores y eventos surgidos en la aplicación y responder a ellos.

2.10.1.4 Aplicación cliente

Una aplicación cliente no tiene capacidades servidoras, solo funcionalidades para comunicarse, crear y controlar remotamente Aglets, y mandar mensajes remotos. Este tipo de aplicaciones requiere menos recursos y requiere menos controles de seguridad, dado que no se baja código compilado desde el servidor, permitiendo solo utilizar el poderoso recurso de la movilidad de los Aglets.

3. BUSCADORES

Debido al incremento inusitado de la cantidad de información disponible en Internet, era indispensable crear un sistema de clasificación y búsqueda de la información que permitiera identificarla ya sea por el tema o por una palabra específica.

Actualmente, han ido apareciendo diferentes sistemas para realizar búsquedas de información, cada uno con su método particular para hacerlo. De los tipos de búsqueda, se puede decir que hay 2 tipos principales: los de directorio y las máquinas o motores de búsqueda.

Los directorios de páginas Web, clasifican la información en forma de índices de registros con los datos enviados directamente por los creadores de las páginas, mientras que los motores de búsqueda, recorren permanentemente toda la WWW, por medio de programas automáticos o **robots**, también llamados **Spiders**, relacionando los nombres de las páginas o direcciones URL con su contenido. A continuación y a lo largo de todo el siguiente capítulo, nos ocuparemos exclusivamente de los motores de búsqueda.

4. MOTORES DE BÚSQUEDA

Un motor de búsqueda es un mecanismo que se encarga de la extracción de datos útiles para el usuario - que se encuentran originalmente en una gran colección de datos desordenados como es la red de de hosts de *Internet* - a partir de un subconjunto de estos que se encuentran almacenados dentro de las estructuras de datos internas al servicio en cuestión.

Generalmente los datos se adquieren a partir de la información textual publicada en forma de páginas *Web,* pero esto no es siempre así, ya que hoy se puede buscar información dentro de los servicios de *news,* directorios *FTP,* documentos en formato *PDF,* etc. Estos servicios de búsqueda de datos se pueden subdividir en dos grandes grupos: los *índices temáticos* y los *robots.*

4.1 Componentes

Los tres componentes de un motor de búsqueda son:

Robot. Un programa especial que "atraviesa" la red saltando de vínculo en vínculo. Este programa puede recibir diferentes nombres: spider, robot, worm (gusano), wanderer (vagabundo) o crawler (reptil). El programa lee el contenido de los archivos para incorporarlo a la base de datos, y busca nuevos vínculos que visitar para obtener más archivos. El mismo programa continuará visitando periódicamente los mismos archivos para detectar si se los han actualizado o eliminado.

Índice. Base de datos que contiene una copia completa o parcial de los documentos reunidos por el robot, y la información derivada de ellos por medio de programas especiales que facilita la labor de los mecanismos de búsqueda y le da significación a los resultados de la misma.

Mecanismo de Búsqueda. Un software que permite al usuario indagar el índice a través de una página web y que devuelve resultados significativos a la búsqueda, habitualmente ordenados según la relevancia asignada.

4.2 Indexación y almacenamiento

La indexación, es la operación destinada a representar los resultados del análisis de contenido de un documento o de una parte del mismo, mediante elementos (denominados genéricamente 'términos de indexación') de un lenguaje documental o natural, generalmente para facilitar la recuperación.

A diferencia de los directorios, cuya indexación es intelectual, la mayoría de buscadores del Web realiza una indexación automática. La principal finalidad de estos últimos es incluir en sus índices de trabajo la información acerca de la mayor cantidad de páginas, alardeando así de ser 'los más grandes' y con ello atraer más publicidad que, al fin y al cabo, constituye su fuente de ingresos.

Los buscadores utilizan distintos métodos para indexar los recursos que incorporan a sus bases de datos. La indexación puede plantearse en tres niveles: **Submorfológico**, **por palabra clave** y **por conceptos**.

4.2.1 Indexación a nivel Submorfológico

La indexación en el nivel **submorfológico**, esto es, sin análisis morfológico, sintáctico o semántico, ofrece un método muy flexible para la recuperación. Así las fuentes de información se indexan como patrones de bits o *bit patterns* de manera que texto, sonido e imágenes en movimiento, pueden indexarse y recuperarse usando la misma forma de representación. Algunas herramientas de consulta comienzan a incorporar sistemas como, por ejemplo, Excalibur Visual RetrievalWare, que ofrecen recuperación de imágenes y de texto.

Sin embargo, la **indexación por palabra clave** o **por conceptos** son las que se utilizan principalmente para la representación e indexación de la información. Estos métodos se desarrollan gracias a la aplicación de técnicas estadísticas de Recuperación de Información – RI - ahora incorporadas a una amplia gama de buscadores (Barlow, 1997).

4.2.2 Indexación por palabra clave

Es la forma más común de indexación de textos en la WWW. Para que los programas de búsqueda puedan encontrar los datos con rapidez, se suelen utilizar ficheros inversos de raíces y palabras clave, direcciones, ubicación y frecuencia de apariciones. Este enfoque, esencialmente morfológico y estadístico, basa la RI en la similitud formal de las palabras, y las estadísticas de su presencia en documentos y colecciones de documentos. Crean en sus índices de trabajo internos una entrada para cada una de las palabras únicas que figuran en la base de datos. Cada una de esas palabras tiene asociada la información sobre su frecuencia estadística y, un puntero para cada registro que describe cada una de las páginas donde ocurre, el cual a su vez incluye la dirección URL.

Las palabras vacías (artículos, preposiciones, conjunciones, etc. así como aquellas palabras que no son discriminatorias por su elevada frecuencia de aparición en el conjunto de documentos del Web) no suelen incluirse en el índice sino que son filtradas ayudándose generalmente de una lista de palabras vacías o anti - diccionario. Los términos asignados mediante la indexación automática se complementan en algunos casos con los términos de indexación que el autor del documento HTML incorpora en el *tag* <META>.

El problema de la asignación de descriptores o identificadores por parte del autor de páginas Web, es que, a veces éste no es honesto (generalmente por razones comerciales o ambición de estar en los primeros resultados devueltos por un motor de búsqueda) y otorga términos que tienen muchas posibilidades de ser interrogados por el usuario, aunque éstos no se correspondan con el contenido de su documento; es el caso típico de los términos *'sex'* o *'free'* o el uso de marcas comerciales muy buscadas, como por ejemplo 'Netscape', 'Microsoft', etc. como identificadores.

4.2.3 Indexación por conceptos.

Existen varios procedimientos para construir bases de datos basadas en conceptos, algunas de ellas muy complejas. En algunos casos estos métodos se basan en sofisticadas teorías lingüísticas y de inteligencia artificial. En otros casos, como Excite, se basan en una aproximación numérica, calculando la frecuencia de aparición de ciertas palabras significativas. A partir de análisis estadísticos el buscador determina qué conceptos aparecen juntos o relacionados en textos que se centran en un tema concreto. Mediante este sistema se pueden recuperar recursos que tratan un tema dado, incluso aunque las palabras incluidas en el documento no coincidan formalmente con las de la pregunta. Otros sistemas, como Dr-Link, realizan un análisis más profundo e indexan a nivel sintáctico y semántico. Sin embargo, el mayor nivel de análisis semántico, posiblemente sea el de los sistemas que ofrecen información evaluada, revisada e indexada por humanos, que se presenta en algunos directorios temáticos, como por ejemplo en Excite e Infoseek.

Algunos de los grandes buscadores del Web, como por ejemplo AltaVista, WebCrawler, u OpenText Web Index, no hacen una selección de términos para incluir en su base de datos sino que indexan el texto completo de los documentos, es decir almacenan *todas* las palabras que aparecen en los textos de las páginas Web que han recogido, a excepción de las palabras vacías.

Otros, como por ejemplo Jumpstation II incluyen en su base de datos el contenido de determinados *tag*s y además seleccionan como otros términos de indexación las 3 ó 4 palabras del texto que más se repiten. Además, algunos procedimientos otorgan mayor peso a las palabras que aparecen en ciertas zonas del documento. Lycos, por ejemplo, da mayor importancia a los términos que se encuentren en el título, en la dirección electrónica del documento, en las cabeceras, en los enlaces a otros documentos y en las primeras veinte líneas.

El título suele ser una de las partes más importantes de un texto, sin embargo los autores de páginas web muchas veces se olvidan de incluirlo, o ponen títulos que fuera de la estructura hipertextual del resto de sus páginas no tiene sentido, o es ambiguo o confuso.

4.2.4 Indexación por Hiperenlaces:

El Web, como conjunto de páginas html relacionadas mediante enlaces o hipervínculos, puede ser representado como un grafo, en el que cada página constituye un nodo, y cada enlace puede considerarse un arco. Dado que los enlaces o hipervínculos parten de páginas o nodos concretos y apuntan a otra página concreta, se puede hablar de grafo dirigido, de manera que los arcos tienen un sentido o dirección determinada, independientemente de que los navegadores puedan tener utilidades de vuelta atrás o mecanismos similares. Aunque las páginas html contienen, desde luego, más cosas, además de los enlaces o arcos, éstos constituyen elementos informativos de importancia, que pueden ayudar, además, a caracterizar dichas páginas. En efecto, los hipervínculos establecen una relación entre unas páginas y otras.

De manera intuitiva, se puede pensar que dos páginas que reciben enlaces desde los mismos nodos deben tratar acerca de los mismos o parecidos temas. *Páginas que apuntan o enlazan con los mismos nodos podrían ser más o menos similares en su temática o contenido.*

La utilización de los hipervínculos como términos de indexación de las páginas Web pueden ser usadas en la recuperación de dichas páginas, aplicando las mismas técnicas que se emplean habitualmente en Recuperación de Información basada en palabras o términos. Esto permitiría eliminar las diferencias idiomáticas, dado el carácter multilingüe del Web, y reducir significativamente la capacidad de proceso y almacenamiento necesarias.

El término *índice* es usado a menudo en el contesto de recuperación y ranking sin embargo este tiene un significado específico. Alguna definiciones propuestas por expertos son las siguientes:

"El índice es la herramienta más importante de recuperación de información, es una colección de términos con indicadores o referencias a los lugares donde se encuentra la información en documentos". "Indexar es construir una estructura de datos que permita buscar texto rápidamente". "el acto de asignar términos de índice a documentos que son los objetos a ser recuperados"; "El término índice es una palabra cuya semántica ayuda a grabar los principales temas de un documento". Cuatro aproximaciones de indexación de documentos en la Web son las siguientes:

- Indexación Manual o humana.
- Indexación automática.
- Indexación basada en agentes inteligentes.
- Indexación basada en Metadatos, RDF y anotaciones.

Las dos primeras son indexaciones clásicas de texto mientras que las dos últimas son relativamente nuevas y prometen áreas de estudio.

Indexar páginas Web para facilitar la recuperación rápida de esta información es una tarea muy compleja y un problema que tiene que ver con los clásicos de bases de datos. El enorme número de páginas Web existentes y el rápido incremento y actualización hacen que la indexación asistida por computadora sea francamente indispensable, de lo contrario esto sería una tarea imposible de realizar manualmente. De hecho muchos expertos afirman que una gran porción de la Web no es grabada por ninguna máquina de búsqueda.

En adición al gran volumen de documentos a ser procesados los indexadores deben tener en cuenta otro problema: la mayoría de las páginas sufren cambios y actualizaciones cada mes, mantener actualizada la información indexada es un problema más complejo que la indexación en sí.

Henzinger, ha sugerido un método para evaluar la calidad de las páginas en los indexadores de las máquinas de búsqueda. La idea de Henzinger es evaluar la calidad de las páginas de acuerdo a su *indegree* (mide cuantas otras páginas de la Web están apuntando a la página a ser evaluada) y a su *pagerank.* El desarrollo de indexadores ayuda en el filtrado y muchos otros problemas de búsqueda y obtención de información en la Web.

4.3 Tipos de indexadores

4.3.1 Manual

La indexación manual es usada por varias máquinas de búsqueda comerciales. En la práctica es improbable que este método siga siendo exitoso en los próximos años, debido a que el volumen de información en la Web crece a pasos agigantados. Otro problema de la indexación manual es la falta de consistencia entre dos diferentes indexadotes profesionales. Este método no es muy bueno comparado con métodos de indexación automática. Los avances tecnológicos están tratando de achicar el espacio en cuanto a la calidad de indexación manual y la indexación automática, de manera que en un futuro la indexación manual solo sea utilizada para pequeñas bases de datos de páginas Web en una corporación.

4.3.2 Robots/Crawlers

Recientemente, científicos están investigando el uso de agentes inteligentes para realizar tareas específicas como la indización en la Web. Estos agentes pueden ser vistos como **robots** o **crawlers**. Muchas máquina de búsqueda confían la generación de índices de manera automática por estos agentes entre ellos Google, Yahoo, Lycos, etc. Hay tres problemas que se asocian al uso de estos robots:

hay gente que tiene miedo de que estos agentes sean invasivos.

Estos robots pueden sobrecargar de trabajo a los servidores si no son manejados con cuidado.

Hay sitios que se actualizan con frecuencia diaria con lo cual estos agentes debería proporcionar una manera de actualizar la información con esta frecuencia.

4.3.3 Metabuscadores

Son sistemas capaces de buscar información acerca de un tema específico, pero en lugar de ser éste quien recolecta los datos de la red, deriva las consultas a varios motores de búsqueda existentes en Internet a la vez.

A pesar de que estos sistemas pueden tener una interfaz *Web*, tal como un buscador tradicional, su funcionamiento es muy diferente, y debido a que no tienen que manejar grandes cantidades de datos, los requerimientos de hardware no son tan grandes como en un buscador tradicional.

La implementación de estos sistemas puede ejecutarse del lado del cliente. Como se vio en las secciones anteriores acerca de las características que tienen los principales motores de búsqueda existentes en la actualidad, cada uno de estos tiene fortalezas que no se encuentran en los demás, y justamente los Metabuscadores se pensaron para aprovechar estas fortalezas de una forma sencilla para el usuario. Por otro lado se puede decir que en general ningún motor de búsqueda puede cubrir todos los contenidos que están publicados en la red de Internet, por lo que estos sistemas hacen posible que se aprovechen mejor los recursos en cuanto cantidad de datos recolectados por cada motor de búsqueda.

5. SISTEMAS DE RECOMENDACIÓN

En el contexto de Internet la gran cantidad de información origina que sus diferentes usuarios lleven a cabo una ardua tarea al momento de realizar una búsqueda. Muchas son las ocasiones en que se desconoce la existencia de documentos que en un determinado momento podrían ser de interés para determinados usuarios. Por lo tanto se omite información valiosa que serviría para enriquecer el contenido de investigaciones múltiples o simplemente de nuestro conocimiento.

Históricamente, los seres humanos han resuelto el problema mediante la solicitud de recomendaciones de viva voz por parte de expertos del área en la que nos vemos carentes de conocimiento y experiencia. Esto implica ciertamente un proceso social natural, por lo que los usuarios de Internet lo realizan con frecuencia en sus investigaciones, así como sus hábitos de lectura.

En estos años se han generado distintas soluciones involucrando a la tecnología de por medio. La más prometedora de ellas es la que plantea el concepto de bibliotecas digitales. Así alguno de los servicios deseados como organización, recuperación y almacenamiento de la información, son provistos en teoría por una biblioteca digital. De hecho una biblioteca digital no solo tiene como objetivo proporcionar a los usuarios los documentos fuentes (libros, revistas, tesis, etc.) en formato digital, sino además, brindar métodos (servicios) para acceder a ellos.

En efecto, tales servicios son importantes, sin embargo, como hemos visto, es necesario contar con un servicio adicional que sea capaz de recomendarnos material bibliográfico y que sustituya al experto al que solemos recurrir, ya que este no siempre se podrá contactar. A este tipo de servicios se lo denomina Sistemas de Recomendación.[Resnik y Varian 1997].

Los sistemas de recomendación surgen a partir de la necesidad de poder proveer a los usuarios información relevante y personalizada. Es decir, como usuario, a uno le gustaría que el periódico de cada mañana consistiera solamente de la clase de artículos que a uno le interesa. O bien que a partir de mis hábitos e historial de lectura, se pudieran recomendar que libros leer. Ejemplos como estos han movido a investigadores de distintas áreas de la computación a desarrollar nuevas herramientas que permitan la construcción de estos sistemas.

Wang, define a un sistema de recomendación como "aquél sistema que tiene como principal tarea seleccionar ciertos objetos, de acuerdo a los requerimientos del usuario, dado que estos objetos están almacenados y caracterizados, en base a sus atributos". Evidentemente, en un sistema rudimentario el usuario podría fácilmente consultar la información de alguna base de datos, dado un criterio de búsqueda, y pareciera que el problema ha sido resuelto. Sin embargo los usuario se encuentran ante un nuevo problema que es cómo expresar sus necesidades de información en términos de un criterio de búsqueda que en la mayoría de los casos no es del todo preciso [Belkin 2000; Dharap 1998; Wang 1998]. Podemos percatarnos entonces que se deben generar alternativas a este mecanismo de búsqueda de información. Y es precisamente un sistema de recomendación el que "oculta" ese criterio de búsqueda explícito, sin eliminarlo, y se trabaja ahora en el modelado del usuario que al final se traducirá en ese criterio de búsqueda que es generado automáticamente sin intervención del usuario final.

5.1 Modelado de las necesidades de información del usuario

Un reto que se presenta indudablemente al momento de crear un sistema de recomendación, es como modelar al usuario con sus necesidades de información que han de permanecer estáticas, y que por tanto habrá que simular ese muy posible dinamismo. [Balavanovic 1998a] sugiere que más que modelar al usuario, se tiende a modelar los objetos en los que este mostró, muestra y posiblemente mostrará interés. Sin embargo existen otros enfoques muy interesantes sobre este tema como lo es la teoría de los estereotipos [Balavanovic 1998[a]; Brajnik et al.; Rich 1979], que no solo definen las necesidades de información del usuario de una u otra forma, sino que además tomaría en cuenta datos como su nivel de educación, religión, edad, entre otros, llevando con esto a generar recomendaciones que posiblemente sean más efectivas. La investigación se encuentra abierta, mientras la propuesta de modelar la información del usuario es plausible y efectiva y por consiguiente muy usada.

5.2 Mecanismos de recomendación

En la actualidad existen dos mecanismos bien definidos para llevar a cabo la tarea de recomendación: el que se *basa en el contenido* y el *colaborativo.*

5.2.1 Recomendación basada en contenido

Balavanovic y Shoham describen a un sistema de recomendación basado en contenido, como "aquel sistema en el que las recomendaciones son realizadas basándose solamente en un perfil creado a partir del análisis del contenido de los objetos, que el usuario ha evaluado en el pasado".

Básicamente, un sistema de recomendación que use un mecanismo basado en contenido para generar recomendaciones, debe ser capaz de modelar y representar que objetos han sido de utilidad o agrado para el usuario, para que de esta forma este último reciba subsecuentemente una serie de objetos similares. Por ejemplo un sistema de recomendación puede resultar muy obvio, o incluso ingenuo, pero nos da la pauta que sigue el mecanismo que especificando.

En este momento surgen varios cuestionamientos: ¿Cómo representar los objetos a recomendar? ¿Qué métricas nos ayudan a saber el grado de similitud entre dos o más objetos? ¿Cómo analizar automáticamente el contenido de tales objetos? ¿Cómo mantener el perfil o historial del usuario con respecto a esos objetos?

De todos estos interrogantes se pueden destacar dos términos importantes: **representación** y **aprendizaje**.

Diversos trabajos trataron de responder estos cuestionamientos. Uno de los enfoques más empleados es el conocido como *Espacios Vectoriales* [Salton y McGill 1983] que representa el carácter temático de los objetos, como un *vector de términos*. Otro enfoque hace uso de diferentes técnicas del Area de Aprendizaje Automatizado como son las *redes Bayesianas* [Sahami 1998] y derivados de esta como son la *Clasificación(Classification)* y la *Agrupación(Clustering)*, Los dos enfoques son efectivos en general es usado el modelo de espacios vectoriales por su simplicidad.

5.2.1.1 Espacios Vectoriales

El modelo de espacios vectoriales [Sahami 1998; Balabonovic 1998[a]; Salton y McGill 1983] ha sido empleado en repetidas ocasiones para la representación de documentos en el campo de la recuperación de información. A pesar de que este modelo solo llega a representar el carácter temático de los documentos, su aplicación ha resultado exitosa en la mayoría de las ocasiones.

Fundamentalmente, cada documento se representa como un vector booleano o numérico, en un espacio de dimensión **p**, donde cada dimensión corresponde a un término diferente en el conjunto o colección de documentos a caracterizar. Un término se refiere a una secuencia alfanumérica delimitada por espacios en blanco o signos de puntuación, que no es otra cosa que una palabra. De manera más formal tenemos:

- Sea D = {d1,d2,dn} el conjunto o colección de documentos.
- Sea {t1,t2,tp} el conjunto de palabras o diccionario de toda la colección de documentos.

Cada documento d es representado como el vector de dimensión p, d = [dt1,dt2,.....,dtp] donde dti es el peso de la palabra ti en el documento d para toda i entre 1 y p.

Podemos percatarnos que el peso de cada término para cierto documento d puede variar. Este peso está determinado, en la mayoría delos casos, por la frecuencia con la que el término aparece en el documento. En la literatura encontramos distintas funciones que llevan a cabo tal tarea. La más difundida, y tal vez empleada, es la propuesta por Salton y Buckley [1998], mejor conocida como "TFIDF":

TFDF(ti,d) = TF(ti). IDF(ti)

esta función indica el peso asignado al término ti en el documento d, donde TF(ti) representa la frecuencia del término o palabra ti en el documento d, y IDF(ti) es la frecuencia inversa del término ti en el documento d.

$$TF(ti) = 0.5 + 0.5\ tfi\ /\ tfmax$$

$$IDF(ti) = \log (n / dfi\ )$$

Donde t*f*i es el número de ocasiones en que ti aparece en el documento d; dfi, es un número de documentos de la colección de documentos que contienen el término ti, *f*max, es la máxima frecuencia de un término por sobre todos los demás términos. La asignación de pesos, sin embargo, puede ser menos sofisticada y mucho más simple de los que parece. Alternativamente, podemos emplear tan solo valores boléanos, y así tener vectores en los que los pesos vendrían determinados por la ausencia o presencia del término ti en el documento d. Esto es, la función *f*(ti,d), que determina el peso del ti para el documento d. Está definido por el número de ocurrencias λ, en que ti se encuentra en d:

f(ti,d) = 1 si λ >= 1 o 0 si λ = 0

Ahora bien, para encontrar la similitud entre dos documentos d1 y d2, calculamos el coseno del ángulo entre los dos vectores de los documentos. Esto puede ser expresado de la siguiente forma:

SIM (d1,d2) = d1.d2

5.2.2 Recomendación colaborativa

Un sistema de recomendación colaborativo puede definirse como "aquel sistema en el que las recomendaciones son hechas basándose solamente en los términos de similitud entre los usuarios", es decir, no se realiza ningún tipo de análisis de los objetos a recomendar.

Algunos autores se refieren también a la recomendación colaborativa como filtrado colaborativo. La idea principal de este mecanismo se basa en que la tarea de realizar las recomendaciones, se lleva a cabo por los mismos usuarios. Se puede afirmar que este mecanismo logra adaptar fielmente el proceso natural de recomendar realizado por los seres humanos, logrando automatizarlo.

Pero ¿cómo lograr entonces automatizar ese proceso natural de recomendar? Para responder a esto, la mayoría de los investigadores parten de la suposición de que existen tendencias o patrones en los gustos o preferencias de las personas, esto es, que no se encuentran distribuidas de manera aleatoria. La meta es entonces encontrar patrones entre las preferencias de las personas, puesto que se ha observado que esto suele presentarse con mucha frecuencia. De esta forma, se realizarían comparaciones de preferencias o gustos de los usuarios en un campo de aplicación determinado, con el propósito de hallar grupos de personas con características similares.

El proceso de recomendación colaborativa puede resumirse según Shardanand y Maes [1995] en tres pasos:

- El sistema mantiene un perfil de usuario, que contiene tanto evaluaciones positivas como negativas sobre objetos que han sido de interés para el usuario.

- Se compara este perfil con los perfiles de los usuarios restantes, para determinar el grado de similitud que existe entre ellos.

- Habiendo encontrado los perfiles con más similitud, se recomiendan objetos que el usuario no haya evaluado.

Para entender mejor este mecanismos apliquémoslo a manera de un ejemplo, a un sistema de recomendación de películas. Entonces, supongamos que A y B son usuarios. A ha evaluado positivamente a "Los Ángeles al desnudo", "El silencio de los inocentes", "Expedientes X: la película", "Loco por Mary". Por su parte, el usuario B evaluó de igual forma todas las películas de A excepto "Loco por Mary". El sistema debería ser capaz de notar que A y B tienen gustos similares (o al menos en cierta medida), y recomendar al usuario B "Loco por Mary". Ahora bien cómo hacemos de manera automática para conocer el grado de similitud entre dos usuarios? Esto se puede realizar a través de métricas como el Coeficiente de Parson.

5.2.2.1 Coeficiente de Parson

Una de las métricas más empleadas para determinar la similitud entre dos usuario a e i, es el coeficiente de Parson [Good et. Al. 1999]

$$W(a,i) = (\sum j (Va,j - V'a) (Vi,j - V'i)) / SQRT (\sum j (Va,j - V'a)^2 \sum j(Vi,j - V'i)^2)$$

Donde

$$V'i = (1 / | Ii |) \sum_{(j \varepsilon I)} Vi,j$$

Donde Va,j representa el voto o calificación del objeto j por parte del usuario a; Vi,j, el voto o calificación del usuario i al objeto j; entendiéndose V'i, como el voto o calificación promedio del usuario i, por sobre los objetos que haya evaluado, siendo Ii este conjunto de objetos.

5.3 Sistemas de Recomendación enfocados a Internet

Existen diferentes sistemas de recomendación enfocados al material disponible en Internet. Resaltan sistemas que recomiendan páginas electrónicas como FAB y Sience Forum, y de contenido general en Web como el trabajo descrito en [Imundon y Neuman 1998]. Todos estos sistemas describen interesantes procedimientos, algoritmos y consideraciones a tomar en cuenta en el proceso de creación de un sistema de esta índole.

5.3.1 FAB [Balavanovic 1998a]

Es un sistema de recomendación de páginas electrónicas que alude a la concepción de un periódico virtual, que se adecua a los gustos de lectura propios de un usuario. En este estudio se detallan todos los pasos que se siguieron para la creación de FAB, desde la recolección de información hasta la evaluación de los elementos recomendados. Pero, sin temor a errar, el más grande aporte hecho es haber propuesto combinar los dos mecanismos existentes para la generación de recomendaciones: el basado en contenido y el colaborativo, para tratar de eliminar los problemas y desventajas que acarrearía el usar uno solo.

5.3.2 Science Forum

Es un sitio Web, que encuentra con un subsistema de recomendación de fuentes de información en la red, con la finalidad de ayudar a académicos, desde preescolar hasta el grado 12, en su actividad de enseñanza.

6. SISTEMAS DE CALIFICACION: PICS

6.1 Descripción

En agosto de 1995, los miembros principales de la comunidad de Internet vinieron juntos comenzar el desarrollo de las especificaciones técnicas que permitirían a los usuarios 1) encontrar fácilmente contenido apropiado y 2) evitar el contenido que considerara inadecuado o indeseado, para sí mismos o para sus hijos. Estas especificaciones fueron diseñadas para facilitar la creación y acceso a esquemas etiquetados (con mecanismos asociados de selección y filtrado), permitiendo que las organizaciones etiquete el contenido de la información en la web de manera que se pudieran tener en cuenta sus diversos puntos de vista. Las especificaciones del PICS no fueron pensadas para limitar el usos de contenido potencialmente ofensivo. Se esperaba que PICS fuera utilizado para muchos propósitos, tales como la graduación de una tercera persona en la puntualidad y la exactitud técnica del contenido de un sitio.

Las especificaciones técnicas finales fueron terminadas a principios de 1996. PICS se ha incorporado desde entonces en un número importante de productos, y una variedad de servicios de evaluación se han desarrollado por PICS para la Web, como así también un gran número de herramientas de filtrado independientes a partir de este momento se hicieron compatibles con PICS.

Muchos de los creadores de PICS reconocieron que la World Wide Web proporciona el acceso a una gama extraordinaria de contenidos, algunos de los cuales son considerados inadecuado, indeseado, perjudicial, etc. para algunos usuarios, especialmente los niños.

La naturaleza global de la Web, y el hecho de que sirve a comunidades numerosas con una gran diversidad de valores, hace que aún los leyes internacionales que restringen ciertas clases de discurso en la Web para ciertos países no sean eficaces ni necesariamente deseables para la Web en otros países, PICS fue desarrollado para tratar de homogenizar la amplia gama de comunidades en línea.

Las personas que propusieron PICS basaron su trabajo en un sistema general de principios, detallado más abajo. Las organizaciones originales que propusieron las especificaciones del PICS también adoptaron la siguiente declaración de principios para dirigir su trabajo:

Creemos que los individuos, los grupos y los negocios deben tener acceso fácil la gama más ancha posible de productos y contenidos de selección, y una diversidad de sistemas voluntarios de ratings . Para avanzar sus objetivos, PICS ideará un sistema de los estándares que facilitan lo siguiente:

Self-Rating :

permita a los proveedores de contenido etiquetar voluntariamente el contenido y distribuirlo.

Third-party Rating :

Habilita múltiples servicios de etiquetado independientes y asocia etiquetas adicionales al contenido creado y distribuido por otros. Los servicios pueden idear sus propios sistemas de etiquetado, y el mismo contenido puede recibir diversas etiquetas de diversos servicios.

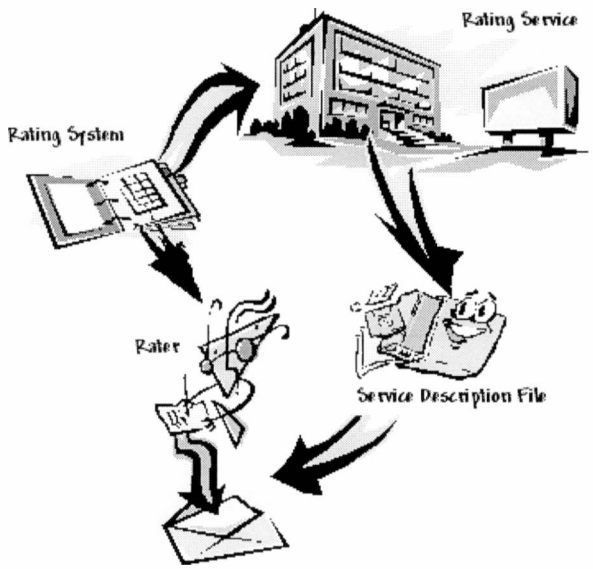
Ease-of-use :

Permita a los usuarios, a los padres y a los profesores utilizar grados y etiquetas de una diversidad de fuentes para controlar la información que llega o los niños.

Los miembros del PICS creen que una plataforma de etiquetado abierta que incorpora estas características proporciona la mejor manera de preservar la diversidad de contenido en la Web. El permitir acceso fácil a la tecnología y la graduación del contenido de terceros dará a los usuarios control máximo sobre el contenido que la Web recibe sin requerir nuevas restricciones proveedores de contenido.

6.2 Arquitectura PICS

La base de la infraestructura PICS es el Service Rating. El **Service Rating** elige un nuevo **System Rating** o bien desarrolla uno para utilizar el etiquetado de contenido (Labeling Content). El **System Rating**, es un sistema que pone valores al contenido de una URL. El **Service Rating** establece los criterios para determinar quién pueden etiquetar el contenido usando su nombre y cómo deben ser aplicadas las etiquetas. Esta combinación de criterios y valorización es identificada únicamente por el servicio de URL (**Service URL**). Este servicio se convierte en la marca de fábrica, del Service Rating. En resumen, el Service URL devolverá en una forma comprensible por el humano los criterios de valorización. El **Service Rating** es también responsable de entregar **un archivo de descripción del servicio** .



Un etiquetador, autorizado accede al Service Rating, utiliza los criterios establecidos por el junto con el System Rating para etiquetar contenidos. Estas etiquetas contienen una declaración sobre el contenido del recurso que es etiquetado y link al servicio de URL. Las etiquetas pueden venir en el propio contenido, o con el contenido o de terceros confiados a tales tareas como una oficina de etiquetado. Las políticas determinan las acciones a tomar basadas en las declaraciones específicas de las etiquetas. Si una etiqueta se basa en un Service URL desconocido, se debe recuperar el archivo apropiado de la descripción del servicio para entender qué declaraciones se están haciendo en la etiqueta.

6.3 Formato General

Una etiqueta PICS está formada por un *Service identifier, label options* y un *rating*. El Service identifier es la URL elegida por el Rating Service. Los Label options dan las características adicionales del documento que es clasificado así como las características de valor como por ejemplo la fecha de clasificación del documento. . El rating en sí mismo es un sistema de los pares del atributo-valor que describen un documento a lo largo de una o varias dimensiones. Una o varias etiquetas se pueden distribuir juntas como una lista. La forma general para una lista de etiqueta es:

(PICS-1.1
<*service url*> [*option...*]
labels [*option...*] ratings (<category> <value> ...)
[option...] ratings (<category> <value> ...)

<*service url*> [*option...*]
labels [*option...*] ratings (<category> <value> ...)
[option...] ratings (<category> <value> ...)

...)

Ejemplo:

```
(PICS-1.1 "http://www.gcf.org/v2.5"
    by "John Doe"
    labels on "1994.11.05T08:15-0500"
        until "1995.12.31T23:59-0000"
        for "http://w3.org/PICS/Overview.html"
        ratings (suds 0.5 density 0 color/hue 1)
        for "http://w3.org/PICS/Underview.html"
        by "Jane Doe"
        ratings (subject 2 density 1 color/hue 1))
```

7.1 Historia del desarrollo

Inicialmente, la decisión de desarrollo de la aplicación paso por un modelo clásico cliente/servidor en el que interactuaban agentes de software. Estos eran construidos en base al framework Aglets, donde se utilizaban todos los beneficios asociados a este tipo de agentes:

- Ciclo de vida.
- Movilidad.
- Capacidad de aprendizaje.

Posteriormente, a medida que avanzamos sobre el diseño, y considerando que no teníamos un servidor de base de datos serio, sino una versión de Oracle que corría en una sola plataforma, fuimos madurando la posibilidad de crear un agente que por un lado se conectara a la base de datos en el host donde se hallaba el servidor de base de datos personal, y que por el otro recibiera pedidos de acceso, consulta y modificación sobre la base de datos. Este concepto progreso rápidamente hasta llegar a un servicio completo formado por múltiples Aglets para el manejo de las transacciones sobre la base de datos.

Cuando encaramos el desarrollo de los agentes encargados de las acciones de adquisición de datos del Web e indexación de los mismos fue que pensamos en la posibilidad de que estos pudieran ser dos servicios independientes que dejaran la información procesada sobre la misma base de datos.

Es así que llegamos al modelo funcional final donde tenemos un conjunto de aplicaciones independientes, formada cada una por una cantidad de agentes implementados con Aglets que actúan en forma independiente para obtener la información, indexarla, procesarla y recuperarla. Esto logra por un lado que todas las aplicaciones colaboren con un bajo grado de cohesión. Al mismo tiempo, cada aplicación – llamada por nosotros **Servicio** -, puede correr en diferentes hosts, y moverse de un host a otro usando las propiedades de movilidad intrínsecas en los Aglets, facilitando las tareas de tener servidores de backup y aumentando la tolerancia a fallos del conjunto.

7.2 Diseño y modelo propuesto del aplicativo.

Basándonos en los requerimientos antes mencionados planteamos el desarrollo de ActiveFinder. Algunas de las características son:

1. **Es una aplicación diseñada para funcionar en el ambiente Distribuido de la Web.** ActiveFinder esta desarrollado como una serie de módulos independientes entre si que procesan las distintas etapas de la adquisición de la información para obtener el resultado deseado por el usuario.

2. **Es una aplicación altamente escalable.** Puede funcionar toda en un único Host que actué como Server de la aplicación o puede hacerse funcionar cada módulo en un Host separado. Más aun cada módulo puede implementar métodos de Balanceo de Carga que permitan correr los procesos del módulo en distintos entornos o Hosts.

3. **Portable:** Esta escrito en Java. Por lo tanto es ejecutable sobre cualquier plataforma que soporte este lenguaje.

4. **Diseño Orientado a Objetos:** Cada Módulo está compuesto por un conjunto de objetos que cooperan entre si.

5. **Utilización de Agentes de Software:** ActiveFinder está basado en agentes móviles de software. Estos agentes fueron construidos utilizando el ASDK (Aglets Software Deployment Kit). Los Aglets son objetos Java que pueden moverse de un host a otro dentro de una red. Esto significa que un Aglet que ejecuta en un host o puede parar su ejecución viajar a otro host e iniciar nuevamente su ejecución. Cuando los Aglets se mueven, tanto el código de programa de los mismos como el estado interno de sus objetos se mueven con él.

8. DISEÑO FUNCIONAL

Funcionalmente identificamos una serie de módulos que funcionan como aplicaciones independientes:

8.1 Cliente:

Este módulo controla la interfase con el cliente. El cliente podrá registrarse a través de este, registrar sus características personales y podrá en sus distintas sesiones crear búsquedas independientes unas de otras. Se comunica e interactúa con el Servidor de Búsquedas.

8.1.1 Servicios requeridos:

- Registro de usuario
- Registro de característica del usuario.
- Modificación de característica de usuario
- Creación de una búsqueda.
- Modificación de una búsqueda.
- Baja de una búsqueda.
- Baja de un usuario.

8.2 Servidor de Búsqueda:

Es el módulo de servicios que se encarga de atender las solicitudes de registro, conexión y búsqueda de los usuarios. En este módulo se alojan los objetos destinados a resolver las búsquedas de los eventos sobre la Web y guardar la información característica de los usuarios a medida que estos obtienen y procesan resultados.

8.2.1 Servicios prestados:

- Registro de usuario
- Registra de características de usuario.
- Baja de Usuarios
- Modificación de característica de usuario.
- Registro de búsquedas.
- Retorno de resultados de búsquedas.
- Baja de búsquedas.

8.3 Servidor de Base de Datos:

Esta aplicación atiende solicitudes del Servidor de Búsqueda que permiten guardar y procesar información de características de los distintos usuarios. Realiza procesos de información para las búsquedas. También atiende requerimientos del Servidor de Spiders para almacenar información de sitios que cumplen con el patrón de URLs definido por ActiveFinder, y solicitudes del Servidor de Indexación.

8.3.1 Servicios prestados:

- Conectar un usuario a la base
- Realiza una consulta.
- Procesa información.
- Realiza modificación de actualización.
- Realiza modificación de inserción.

8.4 Servidor de Indexación:

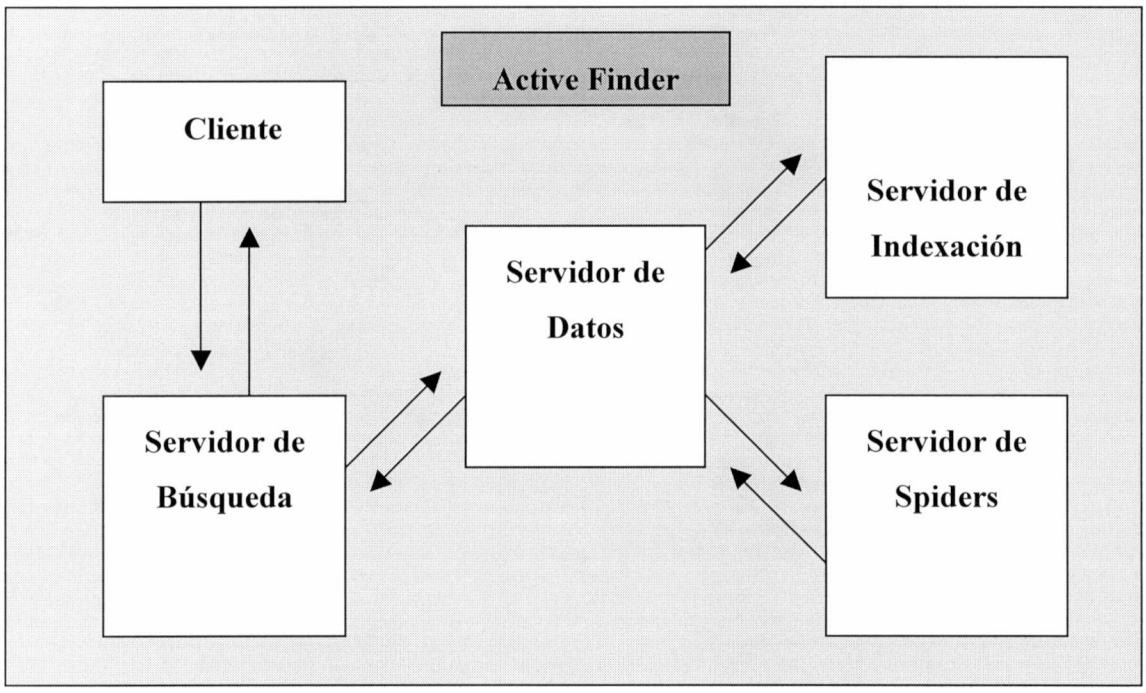
Este servidor realiza la tarea de indexar las URL que cumplen con el patrón ActiveFinder (detallado más adelante) y refresca la información en intervalos de tiempo configurados por el administrador.

8.5 Servidor de Adquisición de datos:

Permite suscribir las URL que cumplan con el patrón ActiveFinder y recorre URLs ligadas buscando que cumplan también este patrón.

8.6 Middleware

Objetos Message de los Aglets que permiten la comunicación e intercambio de información entre los distintos módulos. Si se tiene en cuenta un nivel inferior de abstracción podemos agregar como parte del Middleware el protocolo de comunicación que usan los Aglets, ATP (Aglet Transfer Protocol).


Active Finder
Cliente
Servidor de Indexación
Servidor de Datos
Servidor de Búsqueda
Servidor de Spiders

9. DISEÑO TÉCNICO.

9.1 Método de calificación

El proceso de calificación de una página, con el fin de encontrar el tipo de eventos buscado por el usuario - y que además cuente con las características deseadas por este - es una tarea compleja, para la que no basta con encontrar las *palabras claves* señaladas por el usuario. Un ejemplo muy sencillo es: un usuario desea ubicar todos los congresos acerca de algún tema puntual en una fecha. Cualquiera de los buscadores tradicionales le estaría entregando, con alta probabilidad, páginas como la del Subterráneo de Buenos Aires, donde se pueden encontrar referencias a la estación **Congreso;** Páginas de historia acerca del Congreso de Tucumán; Citas del Congreso de la nación, etc.; a la vez que omitiría toda página con contenido no castellano.

Se presentaron varias opciones. La primera era desarrollar algún método que no solo pudiera analizar sintáctica sino semánticamente las páginas para determinar que contenido se correspondía en mejor medida con los contenidos buscados por el usuario. Este análisis debía poder hacerse teniendo en cuenta los diversos lenguajes de publicación en Internet – al menos cuatro grandes lenguajes -. La segunda contemplaba una acotación del espacio de análisis semántico, reduciéndolo al reconocimiento de un solo lenguaje. La tercera, finalmente adoptada por consejo de nuestro director, consistía en la posibilidad de incorporar **MetaTags** a las páginas con contenido interesante al usuario.

En el último caso, el espacio de búsqueda se reducía a encontrar todas las páginas que contuvieran un metatag específico - llamado **Tag ActiveFinder** – el cual debería contener algunos de los conceptos buscados.

Por otra parte, el agente avatar del usuario en el sistema – el **Assistant** – mantuvo la capacidad de almacenar las preferencias del usuario y de mantener un perfil actualizado del mismo.

El método de calificación entonces tomó su forma. Se buscarían todas las páginas indexadas dentro del sistema donde se encontraran **si o si**, todas las características especificadas por el usuario para una búsqueda – tipo de evento, lenguaje, disertante, etc. – y luego, se calificarían todas las páginas recuperadas por este concepto de acuerdo a las características y preferencias del usuario.

El sistema de calificación usado fue el más sencillo de todos, ya que solo incrementa el valor de las características de usuario de acuerdo con un puntaje asignado por este a cada página luego de examinar su contenido. Este puntaje se aplica a las preferencias almacenadas del usuario estableciendo una escala de preferencias que es respetada y usada en cada búsqueda sucesiva.

Como resultado, con el correr de las búsquedas exitosas y una calificación consciente de parte del usuario, luego de un número de corrida de los algoritmos de búsqueda, se tendrá un perfil bastante aproximado de las preferencias de cada usuario del sistema.

Por supuesto, el sistema se dejo deliberadamente abierto a la posibilidad de cambiar este algoritmo por alguno de los expresados en el capitulo correspondiente.

9.2 Tag ActiveFinder

Como se expresó en el capítulo anterior, cada pagina que contenga información relevante para el Sistema de adquisición de datos, deberá estar armada contemplando la inclusión de un Tag que respete el siguiente formato:

```
<HEAD>
<META NAME="ACTIVEFINDER" CONTENT='
 (Nombre = Un_nombre descriptivo de la página opcional)
 (Description = Una descripción corta opcional)
 {nombre_categoria1=categoría_1[,categoría_2[,categoría_]] }
 {nombre_categoria2= categoría_1[,categoría_2[,]] }
'>
</HEAD>
<BODY>
```

9.2.1 Categorías

Se definen una serie de categorías fijas que deben respetarse. El contenido se agregara a medida que los indexadores descubren nuevos valores no antes agregados para la categoría.

Tenemos dos tipos de categorías:

1. **Categorías incrementales:** Crecen a medida que aparecen nuevos ítems. Algunas son las siguientes

- TipoEvento
- AreaProfesional
- País
- Ciudad
- Disertante

A las que se pueden agregar otras que se consideren convenientes.

2. **Categorías Libres:** Se toman los contenidos para usarlos en búsquedas por semejanza. No se agregan a la lista original. Algunas pueden ser:

- Fecha realización
- Lugar de realización.

9.3 Detalle técnico

9.3.1 Servidor de adquisición de datos – Spider –

Esta parte de la aplicación está formada por un conjunto de agentes (Aglets) que se muestran en la figura de la página 73 con el título de (Servidor de adquisición de datos – Spider –). Está formada por un servidor de aglets Tahiti en el cual se crea un agente que arranca el servidor (ActiveSpider). Dicho agente se encarga de la creación de un nuevo contexto de ejecución y crea otro agente coordinador llamado (SpiderCoordinator). El agente coordinador toma la tabla de repositorio de información que almacena las URL de los sitios que se registran y dispara un agente (Spider) por cada URL. El Spider parsea la URL asignada en busca del TAG Active Finder y lanza Spiders hijos por cada referencia que tenga la página parseada. La profundidad máxima, es configurada en la tabla de repositorio para cada URL. En caso de que la página parseada tenga el TAG se agrega la URL al repositorio de indexación para que luego sea indexada por el Servidor de Indexación.

Cada SpiderCoordinator se desactiva (se baja del servidor y se guarda en memoria persistente durante una unidad de tiempo (x días, x meses o x años), una vez que su último hijo (Spider) termina su trabajo. El tiempo mencionado se guarda también en la tabla de configuración de la URL. Una vez transcurrido ese tiempo el SpiderCoordinator se activa automáticamente y vuelve a realizar el parseo de la página. Esto posibilita el refresco de la información del repositorio de URLs haciendo que la información de las URL que caducaron o se modificaron pueda ser actualizada.

9.3.2 Indexador

Esta parte de la aplicación está constituida por un servidor Tahiti que crea un agente (ActiveIndexer) que dispara a su vez un agente coordinador (IndexerCoordinator). El coordinador recorre la tabla repositorio que alimenta el Servidor de adquisición de datos –Spider-. Por cada URL que se encuentra en esta tabla que no ha sido indexada, el (IndexerCoordinator) dispara un agente indexador (Indexer)

que toma el TAG Active Finder de la URL y lo parsea guardando las categorías y valores para cada URL de esta manera cada URL será luego identificada por una búsqueda de usuario.

9.3.3 Cliente y Servidor de Búsqueda

El cliente de esta aplicación es un el servidor Tahití que se encuentra en el lugar físico del usuario de la aplicación (terminal del usuario). En este servidor se crea el agente (ActiveFinderClient) encargado de abrir la ventana de login de la aplicación y creando el agente (Manager). El Manager obtiene primeramente el AgletProxy del agente Coordinator (agente que se halla en el **Servidor de Búsqueda),** luego envía al coordinador la información del usuario para establecer el la conexión con el servidor y arrancar la aplicación. De esta manera el Coordinador chequea la autenticidad del usuario y en tal caso **moviliza** a los Assistant (agentes representantes de cada búsqueda activa de usuario) la información de las búsquedas activas del usuario.

El Servidor de Búsqueda también se ejecuta en un servidor Tahití. Aunque está pensado para funcionar constantemente, contempla la posibilidad de una caída intencional o imprevista. Cuando se inicia/reinicia el Servidor, este crea un agente llamado ActiveFinderServer. Este agente inmediatamente dispara otro agente, el Coordinator, encargado de varias tareas, entre ellas:

a) Inicialmente recorre la tabla de asistentes que guarda información de cada una de las búsquedas activas que pueda tener cada usuario. Determina si en el momento de la caída el Assistant asignado a la búsqueda estaba en el servidor, en cuyo caso está muerto y se crea de nuevo. Caso contrario registra su posición para uso futuro.

b) Se queda a la espera permanente de solicitudes de conexión de usuarios existentes o mensajes de creación de nuevos usuarios.

Como vemos hicimos referencia dos veces el agente Assistant y no dimos demasiados detalles. Concretamente **es uno de los agentes más importantes de la aplicación** por el rol que tiene. El Assistant representa una búsqueda activa de un usuario, es decir que cada usuario puede tener "**N**" Assistants trabajando en el Servidor de aplicación.

Cada Assistant es creado en el momento que se inicia el Servidor de Búsqueda y cuando el usuario se registra en el Server todos los Assistants que le corresponden se **mueven** al servidor Tahití del cliente. En el momento que se le solicita al Assistant que ejecute una búsqueda, este crea automáticamente un agente que será el representante en su ausencia (recordemos que el se va al cliente) en el Servidor de Búsqueda llamado FinderCoordinator.

Luego de que el usuario se registra y el Assistant migra a la locación del cliente todo el intercambio de información correspondiente a la búsqueda que representan el Assistant y el FinderCoordinator fluirá por intercambio de mensajes entre estos agentes. A través de la interfaz del cliente que se muestra seguidamente el usuario podrá crear y configurar nuevas búsquedas. En la información de una búsqueda se puede configurar:

a) La fecha de inicio y fin de la búsqueda y la **periodicidad** que le indicará al Assistant a cada cuanto tiempo debe despertarse y volver a disparar la búsqueda operación que detallamos más adelante.

b) La cantidad máxima de resultados retornados.

c) Y en caso de resultados la dirección mail a la cual enviar un mensaje de aviso.

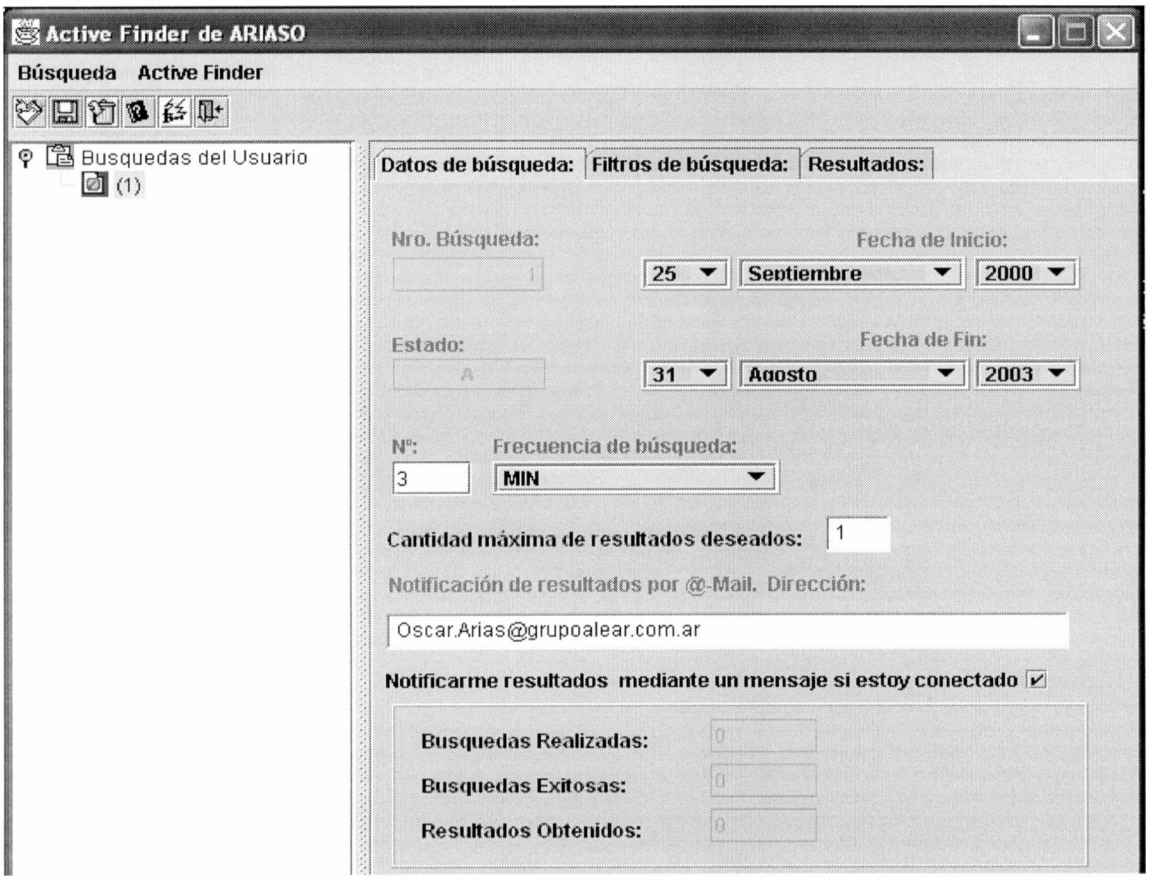


Ilustración 1

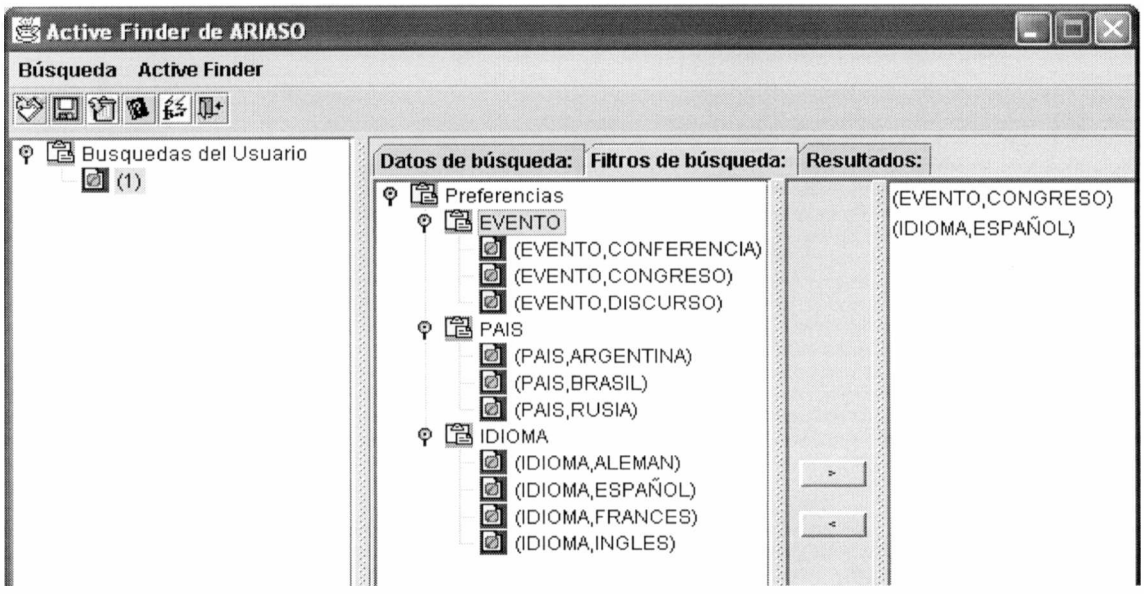


Ilustración 2

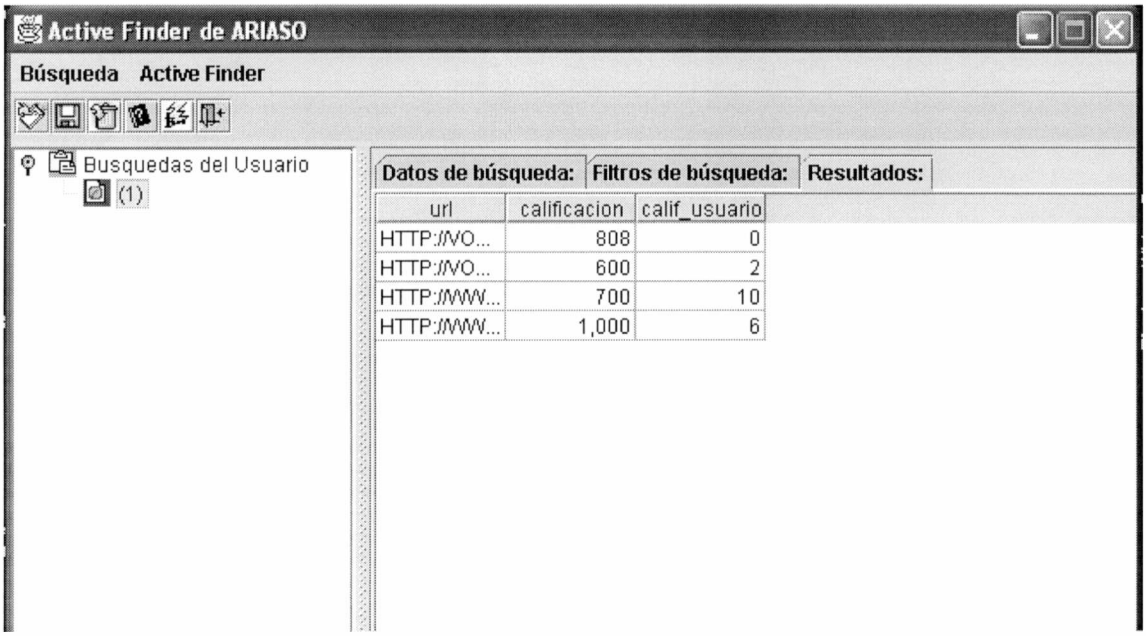


Ilustración 3

Cuando el usuario dispara la búsqueda, el Assistant envía un mensaje de aviso al FinderCoordinator que inmediatamente crea un conjunto de agentes Finder, uno por cada categoría de la búsqueda. Estos agentes Finder's se encargan de procesar la información residente en la base y obtener los resultados. Cada Finder una vez terminada su tarea retorna al FinderCoordinator padre los resultados y este a través de un mensaje de retorno envía los resultados al Assistant en el cliente donde serán visualizados por la interfaz.

En caso de que la búsqueda tenga configuradas las fechas de inicio y fin y el intervalo de reactivación, ejemplo: supongamos una búsqueda que se inicia el 1/1/2000 y finaliza el 1/6/2000 y se reactiva cada 24 Horas, esto le indicará al FinderCoordinator que inmediatamente después de retornados los resultados al Assistant, deberá desactivarse (guardarse en memoria persistente del servidor Tahiti donde corre el Servidor de Búsqueda). Transcurridas 24 horas el FinderCoordinator se activa nuevamente y vuelve a realizar la búsqueda, esto se repite hasta que la misma expire. Aún cuando el usuario este fuera de línea en cuyo caso el Assistant **vuelve a moverse** al Servidor de Búsqueda, los resultados son guardados y retornados al usuario en su próxima conexión. De aquí el nombre de la aplicación "ACTIVE FINDER". Si en la

búsqueda se configuró para que se avise por mail que el FinderCoordinator a encontrado resultados, el usuario podrá enterarse que su búsqueda obtuvo nuevos resultados fuera de línea.

9.3.4 Servidor de Datos

Esta parte de la aplicación también se basa en un servidor Tahití. Fue creada para administrar los datos de la aplicación pensando en la posible utilización de una base de datos distribuida. Inicialmente se crea un agente llamado DatabaseServer. Seguidamente este agente crea a otro llamado TransactionMonitor. El TransactionMonitor se comunica con distintos agentes en los demás servidores para brindar agentes Transaction. Estos agentes son utilizados mayormente por aquellos que se relacionan más directamente con el procesamiento de la información ejemplo: Spider's, Finder's, Indexer's. Ejemplo: cuando se crea una búsqueda el FinderCoordinator creado envía un mensaje al TransactionMonitor para que le devuelva un agente Transaction para posibilitar el manejo de datos en la base. El TransactionMonitor entonces crea un Transaction que inicialmente correrá en el servidor Taihiti correspondiente a la aplicación Servidora de Datos, pero nada imposibilita a este objeto migrar a otro servidor Tahiti en un lugar remoto donde se encuentra una replica o instancia distribuida de la base de datos y **de esta manera una misma búsqueda podría manejar transacciones en paralelo** con las evidentes consecuencias positivas en cuanto a la performance.

9.3.5 Interfaz del monitoreo de Servidores y Clientes en línea.

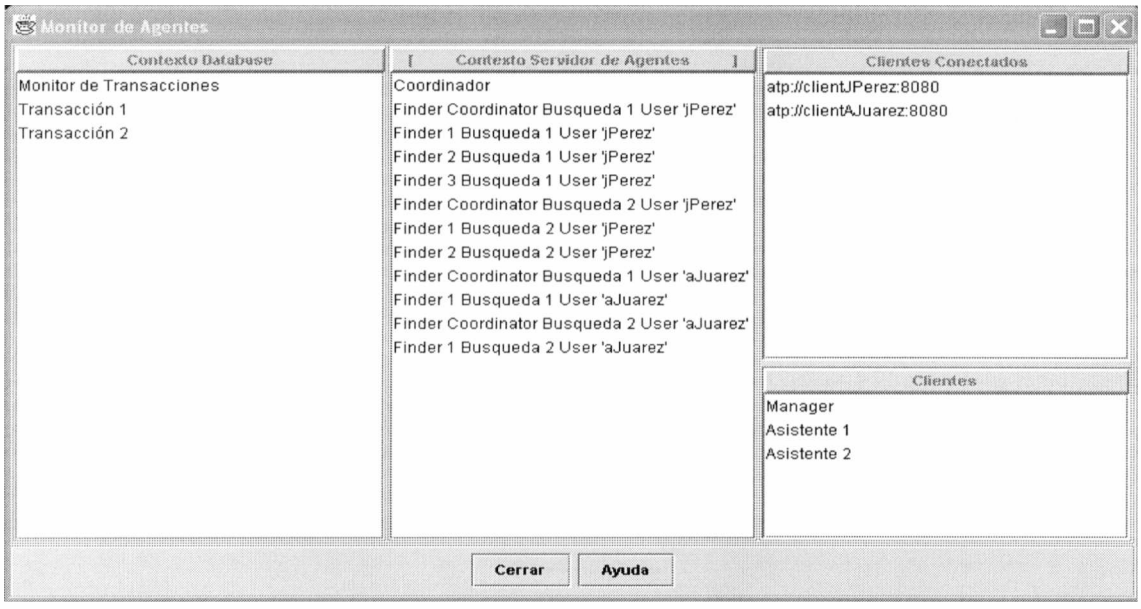


Ilustración 4

9.3.6 Diagrama técnico de interacción de los servicios y el cliente

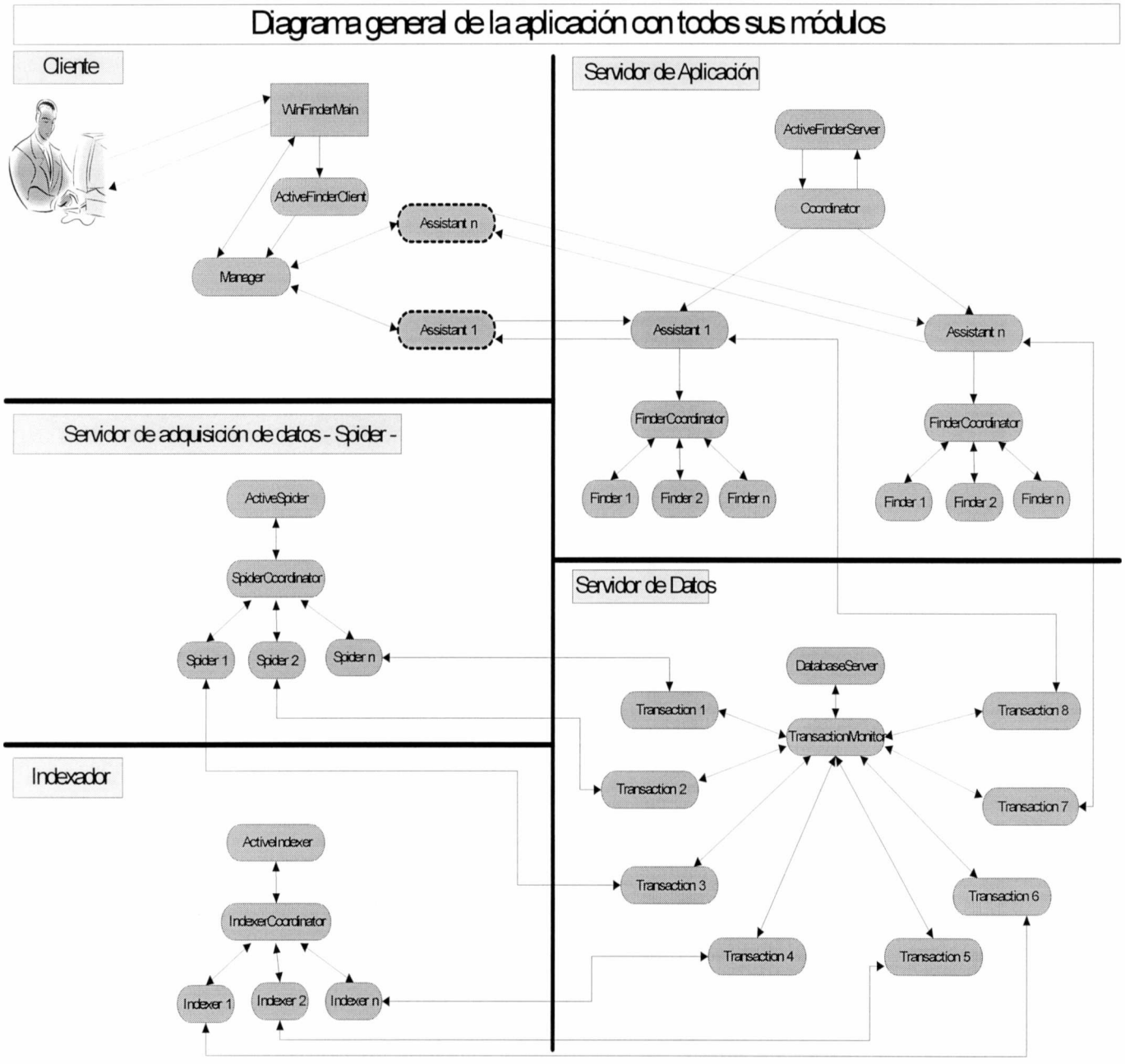


Ilustración5

10. COMPARACIÓN CON OTROS BUSCADORES

La gran mayoría de los buscadores actuales han efectuado una mezcla entre la búsqueda dentro de metatags relevantes y la búsqueda por contenidos directos. Vale decir, miran si una palabra tal aparece dentro de la pagina Web x que generalmente se encuentra almacenada indexada dentro de las bases de datos del buscador. ActiveFinder basa su proceso de indexado y búsqueda sobre un MetaTag donde el creador/publicador de la página vuelca las características que él cree relevantes del evento. Periódicamente el sistema de Spidering recorre las direcciones cabeceras donde se presume encontrará nuevas páginas de Eventos, las verifica, agrega a un repositorio las nuevas o modificadas. Posteriormente, otro sistema totalmente independiente del anterior – aunque dependiente claro está, de sus resultados -, llamado Sistema de Indexado, toma los datos del repositorio, verifica los metatags ActiveFinder y procede a indexar, es decir, separar las características definidas en el MetaTag ActiveFinder, y a almacenar esta información de indexado en la base de datos del sistema de donde es extraído por el Sistema de Búsqueda.

La forma de búsqueda mediante metatags basan sus resultados en la fiabilidad del diseñador/publicador de la página catalogada. Si este hace trampa agregando información mentirosa acerca de la misma, los resultados no serán correctos.

Por otro lado, aún si el diseñador es confiable, la cantidad de páginas encontradas usando un buscador tradicional y ActiveFinder serán distintas. En primer lugar porque no todas las páginas de Internet llevan definidos el MetaTag ActiveFinder.

Supongamos el siguiente ejemplo: Un usuario desea buscar información referente a Congresos de Matemática en idioma ingles. Toma su buscador tradicional, ingresa una expresión tal como:

"congresos matemática ingles"

El buscador le entregará miles de páginas con resultados, de las cuales solo una pequeña cantidad contendrán la información buscada. La gran mayoría serán equivocadas, habiendo sido traídas solo por contener una o más de las palabras ingresadas. ActiveFinder, en tanto, entregará solo aquellas páginas que efectivamente cumplan con las categorías seleccionada. Una búsqueda en ActiveFinder se haría desde la IDE seleccionando

Categoría	**SubCategoría**
Evento	Congreso
Idioma	Ingles
Area	Matemática

El Assistant se encargará de buscar todos los eventos que se encuentren registrados en la base de datos y que cumplan las características.

En resumen, la dependencia de Metatags de ActiveFinder origina beneficios:

- Evita contenidos erróneos.
- Reduce el numero de resultados.
- Aligera las búsquedas.
- Bases de datos más sencillas.

Y contras:

- Dependencia del diseñador.
- Perdida de páginas con contenido por no estar registradas.

- Necesidad de estandarizar el diseño de las páginas en lo atinente a la inclusión del nombrado MetaTag.

Otra cosa a tener en cuenta es la capacidad de ActiveFinder de realizar la misma búsqueda un número determinado de veces. Esto, asociado a un algoritmo de calificación adecuado y retroalimentado por la interacción con el usuario brinda la posibilidad de que cada búsqueda sucesiva sea más precisa colocando los resultados más deseados por el usuario en los primeros lugares.

ActiveFinder es capaz de guardar esa información alimentando a todos los Asistentes del usuario con lo aprendido de este en una búsqueda. Lo que origina búsquedas subsiguientes más y más eficientes y personalizadas.

Esta forma de seguimiento de sesión es poco usada en los buscadores tradicionales. El recuerdo de búsquedas anteriores y trabajo desatendido es apenas explorados al cierre de esta tesis.

Resta mencionar el punto más fuerte de este sistema y que fue motivo original de la tesis:

El sistema está totalmente basado en agentes de software inteligentes que "viven" en un contexto servidor que los mantiene hasta que son llamados por su usuario y, eventualmente mueren cuando no se los necesita más. Es altamente modular. Es totalmente escalable. Es fácilmente distribuible a través de la red intra/internet.

11. REFERENCIAS

- Jennings M. Wooldridge M. Agents Technology – Foundations, Applications and Markets. Springer – UNICOM 1998.
- Nwana M. - Software Agents On Overview.
- Smith, Jerry – The Aglets Framework.
- Oshima M., Karjoth G., Ono K. – Aglets Specification 1.1 Draft
- Chong Xu, Dongbin Tao – Building Distributed Application with Aglet.
- W3C – Platform for Internet Content Selection
- Jim Miller, Paul Resnick, David Singer – Rating Service and Rating Systems.
- Yan Hua Chu, Philip DesAutels, etc. – PICS Signed Labels (DSig) 1.0 Specification
- Crasswell N., Haines J., Humpreys B. – Aglets: a good idea for Spidering?
- Sahuguet Arnaud – About Agents and databases.
- Aglets Home IBM Japon – http://www.trlibm.co.jp/aglets.
- Bill Venners, - Architecture of Aglets. http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html.
- Bill Venners, - Solving Real Problems with Aglets. 1997 http://www.javaworld.com/javaworld/jw-04-1997/jw-04-hood.html
- Jeffrey Bradshaw – Software Agents.
- Hyacinth S., Nwana and Divine T., Ndumu – A perspective on Software Agent Research.